130f17

SCIENTIFIC NOTEBOOK 115E

BY

Charles B. Connor

SCIENTIFIC NOTEBOOK 115E

BY

Charles B. Connor

Southwest Research Institute
Center for Nuclear Waste Regulatory Analyses
San Antonio, Texas

SCIENTIFIC NOTEBOOK 115E

CHUCK CONNOR

PROJECT

IGNEOUS ACTIVITY

PROJECT NUMBER 20-1402-461

July, 1999 - Oct, 1999

PARTS IN THIS NOTEBOOK:

Magnetic Units discussion
Direction cosine code for magnetic units
Plouff code for magnetic modeling
Magnetic Dike calculation code
Magnetic dike plotting code
Revision of "Geologic factors controlling patterns of small - volume basaltic volcanism"
Plume tank experiments by David Alvarado
Numerical modeling of magma repository interactions by Onno Bokove
Cerro Negro magnetic data and gmt scripts
Magnetic cube program
Revisions of the USGS volcanic hazards program review
trip report on UNAVCO geodetic gps meeting

Scientific Notebook
20-1402-461
Inititals: CC

Chuck Connor
July 4,1999

We are calculating the expected magnetic field anomalies produced by buried geologic structures. This is for modeling of magnetic data collected in the YMR. See summaries in Connor and Sanders (1994, Geophysics review topical report: application of seismic tomography and magnetic methods to issues in basaltic volcanism, CNWRA 94-013). And Connor et al.(1997, Magnetic surveys help reassess volcanic hazards at Yucca Mountain, Nevada, Eos vol 78: 73, 77-78.)

First a note about units.....

The Earth's field strength (H) can be reported in Oe or Amp/m. It is infinitely bogus to report it in gamma or nT because those are units of magnetization (J) or induction (B), not field strength - H. People get away with this by talking about the "measured field strength" of the Earth's field, which is a magnetization.

0.5 Oe = 0.5 * 1/(4*PI) * 1000 Amp/m = 39.78 Amp/m

Suppose you have a susceptibility of k = 0.3 SI

39.78 Amp/m * 0.03 SI = 1.193 Amp/m

but this is now a magnetization, not a field strength, so convert it to nT,

J = 1.193 Amp/m * 400 * PI = 1,500 nT

Use J, in nT, for the magnitude of the vector of magnetization produced by a rock with susceptibility k in a magnetic field with strength H.

alternatively, 0.5 Oe is equivalent to a "measured magnetic field strength" of 50,000 nT, so

J = 50,000 * 0.3 = 1,500 nT

which is a shortcut that works because the factor 4*PI cancels, but you lose track of the units and the physical meaning.

—

In pmag, the intensity of magnetization (J) of a sample is measured, usually in gauss (G), which equals 4 * Pi * 10,000 nT. Note 1 G = 1 emu /cc. Also note, gauss is also the unit of the magnetic dipole moment per unit volume (M) and 1

G (magnetic dipole moment / unit volume) = 1000 Amp/m (magnetic dipole moment / unit volume).

Pmag data are usually reported in magnetic dipole moment / unit volume, M. For example M = 0.001 G (cgs) = 1.0 Amp/m (SI), even though the magnetometers measure J, not M.

So, to convert a reported "remanent magnetization" of units magnetic dipole moment /unit volume (M) in SI back to a magnetization (J) in SI units

1.0 Amp/m = 1.0 * 400 * PI nT

To convert 1 G or 1 emu/cc (J or M) to magnetization (J) in SI units:

1 G = 4 * PI * 10,000 nT

Invariably, magnetic anomaly calculations use the components of the vector of magnetization. If I = inclination of the Earth's magnetic field, D = declination of the Earth's magnetic field, Ir = inclination of the vector of remanent magnetization, Dr = declination of the vector of remanent magnetization, then the direction cosines are

m = cos(D) *cos(I)
l = cos(I) * sin(D)
n = sin (D)

M = cos(Dr) * cos(Ir)
L = cos(Ir)*sin(Dr)
N = sin(Ir)

If remanent magnetization is reported as a magnetic dipole moment / unit volume (Mr) and in SI units (Amp/m) then the magnitude of the vector of remanent magnetization is:

Jr = 400 * PI * Mr (nT)

If the susceptibility, k, is reported in SI units and the magnetic field strength,H, in Oe:

Ji = k*H*10,000 (nT, using the shortcut)

and the components of the vector of magnetization are:

Jx = Ji * l + Jr * L
Jy = Ji * m + Jr * M
Jz = Ji * n + Jr*N

Using these values of the components of the vector of magnetization will result in an anomaly with magnitude in nT - which is where we want to be.

Here is an object class that adds the vecotrs of magnetization (remanent and induced) to get the total vector of magnetization.

```
void  dir_cos(float xdec, float xinc, float sus, float H,
        float rdec, float rinc, float nrm, float *dirl,
        float *dirm, float *dirn, int mm)  {
  // calculates the direction cosines for the magnetic field for
  // a given polygon
  //
  // by C. Connor, 10/95
  //
  // Input:
  // xdec - declination of the regional field in degrees (E of N is positive)
  // xinc - inclination of the regional field in degrees (down is positive)
  // sus - susceptibility of the polygon (uniform in the
  //      polygon) in SI units , 1 SI = 1/(4*pi) emu
  // H - Earth's magnetic field strength in Oe (e.g., 0.5)
  // rdec - declination of the remanent vector of magnetization
  //       in degrees (E of N is positive)
  // rinc - inclination of the remanent vector of magnetization
  //       in degrees (down is positive)
  // nrm - intensity of remanent magnetization in emu/cc
  //
```

```
// Output:
// returns the direction cosines for the regional field
// dirl, dirn, dirm
//
// returns the components of the vector of magnetization for the polygon
// tjx[mm],tjy[mm],tjz[mm]
// where tjx[mm], and tjy[mm] are horizontal components
//   - Jx, Jy in the literature
// and tjz[mm] is the vertical component of magnetization Jz
//
float xdec_rad = 3.14159265/180.0*xdec;
float xinc_rad = 3.14159265/180.0*xinc;
*dirm = cos(xdec_rad)*cos(xinc_rad);
*dirl = cos(xinc_rad)*sin(xdec_rad);
*dirn = sin(xinc_rad);

float rdec_rad = 3.14159265/180.0*rdec;
float rinc_rad = 3.14159265/180.0*rinc;
float Rdirm = cos(rdec_rad)*cos(rinc_rad);
float Rdirl = cos(rinc_rad)*sin(rdec_rad);
float Rdirn = sin(rinc_rad);

float JL = nrm*Rdirl;
float JM = nrm*Rdirm;
float JN = nrm*Rdirn;

tjx[mm] = sus*H*(*dirl) + JL;
tjy[mm] = sus*H*(*dirm) + JM;
tjz[mm] = sus*H*(*dirn) + JN;

}  // end dir_cos
```

The above code can be used to calculate the vector of magnetization input into the following code that calculates the magnetic anomaly due to a buried prism. This is the algorithm of Rao and Babu, 1991, A rapid method for three dimensional modeling of magnetic anomalies, Geophysics 56, no.11, 1729-1737.

It is written in True Basic by C. Connor

```
dim t(73,73), t1(73,73), t2(73,73), t3(73,73), t4(73,73)
dim gx2(73,73),gx3(73,73),gx4(73,73)
dim gy2(73,73),gy3(73,73),gy4(73,73)
dim gz2(73,73),gz3(73,73),gz4(73,73)
open #1: name "magN1", create newold
erase #1

!print "input the width (a1,a2)"
!input a1,a2
let a1 = 25.0001
let a2 = 50.5001

!print "input the length (b1,b2)"
!input b1,b2

let b1 = 25.00001
let b2 = 50.50001

!print "input the depth (h1,h2)"
!input h1,h2

let h1 = 10
let h2 = 11

!print "Input the angle from north, in degrees"
!input thet

let thet = 0
let thet = thet *pi/180


!Print "Input the inclination and declination of the Earth's field"
!print "I,D"
!input I,D
let I = 62
```

```
let d = 14
let I = I*pi/180
let D = D*pi/180

let p = cos(I)*cos(D-thet)
let q = cos(I)*sin(D-thet)
let r = sin(I)

!print "input the inclination and declination of the vector of magnetization"
!input Io,Do
!print "input the intensity of magnetization"
!input EI

let Io = 62
let Do = 45
let EI = 1256
let Io = Io*pi/180
let Do = Do*pi/180

let L = cos(Io)*cos(Do-thet)
let M = cos(Io)*sin(Do-thet)
let N = sin(Io)

let g1 = EI*(M*r + N*q)
let g2 = EI*(L*r + N*p)
let g3 = EI*(L*q + M*p)
let g4 = EI*(N*r - M*q)
let g5 = EI*(N*r - L*p)

for k1 = 1 to 72

for k2 = 1 to 72


let x = k1
let y = k2

let xp = x*cos(thet) + y *sin(thet)
let yp = -x*sin(thet) + y*cos(thet)
```

```
let ap1 = a1-xp
let ap2 = a2 -xp
let bp1 = b1 - yp
let bp2 = b2 - yp

let r1 = sqr(ap1^2 + bp1^2 + h1^2)
let r2 = sqr(ap1^2 + bp1^2 + h2^2)
let r3 = sqr(ap2^2 + bp1^2 + h1^2)
let r4 = sqr(ap2^2 + bp1^2 + h2^2)
let r5 = sqr(ap1^2 + bp2^2 + h1^2)
let r6 = sqr(ap1^2 + bp2^2 + h2^2)
let r7 = sqr(ap2^2 + bp2^2 + h1^2)
let r8 = sqr(ap2^2 + bp2^2 + h2^2)

let f1 = (r2+ap1)*(r3+ap2)*(r5+ap1)*(r8+ap2)/((r1+ap1)*(r4+ap2)*(r6+ap1)*(r7+ap2))
let f2 = (r2+bp1)*(r3+bp1)*(r5+bp2)*(r8+bp2)/((r1+bp1)*(r4+bp1)*(r6+bp2)*(r7+bp2))
let f3 = (r2+h2)*(r3+h1)*(r5+h1)*(r8+h2)/((r1+h1)*(r4+h2)*(r6+h2)*(r7+h1))
let f4 =
atn(ap2*h2/(r8*bp2))-atn(ap1*h2/(r6*bp2))-atn(ap2*h2/(r4*bp1))+atn(ap1*h2/(r2*bp1))-
atn(ap2*h1/(r7*bp2))+atn(ap1*h1/(r5*bp2))+atn(ap2*h1/(r3*bp1))-atn(ap1*h1/(r1*bp1))
let f5 =
atn(bp2*h2/(r8*ap2))-atn(bp2*h2/(r6*ap1))-atn(bp1*h2/(r4*ap2))+atn(bp1*h2/(r2*ap1))-
atn(bp2*h1/(r7*ap2))+atn(bp2*h1/(r5*ap1))+atn(bp1*h1/(r3*ap2))-atn(bp1*h1/(r1*ap1))

let T(k1,k2) = g1*log(f1) + g2*log(f2) + g3*log(f3) + g4*(f4) + g5*(f5)
!print #1: x,y,h1,T(k1,k2)
next k2
next k1

mat t1 = T

! And now print it out

end
```

This code, originally from Plouff, calculates the magnetic anomaly for a multi-sided polygon.

```
void calc(int kk, int mm, float shortSide, float zdel,
      double *xp, double *yp, float *zp, float *xb,
      float *yb, int *ll, float *dstnc, float *zee, float *zm)  {
  // calculates the components of the magnetic anomaly resulting from
  // one or more uniformly magnetized, vertical sided polygons.
  //
  // ancient fortran code of Plouff translated by Connor, 10/95
  //
  // Input:
  // kk - number of field points - where the anomaly is calculated
  // mm - number of polygons
  // shortSide - the shortest length of any side of any polygon
  // zdel - the thinnest (vertically) polygon
  // tjx[],tjy[],tjz[] - direction cosine of the magnetization vector
  //                for each polygon. commonly reffered to as
  //                Jx, Jy, Jz in the literature
  //                should have dimension mm
  // xp[], yp[],zp[] - easting,northing, and elevation of
  //                field points (meters)
  // xb[],yb[] - easting and northing of polygon coordinates (meters)
  // ll[] - number of sides in a given polygon
  // dstnc[] - length of the sides of polygons (meters)
  // zee[] - depth to the top of the polygon(s) (meters)
  // z[] - depth to the bottom of the polygon(s) (meters)
  //
  //Output:
  // delta_X, delta_Y, and delta_Z - magnitudes of the
  //                      components of the magnetic field
  //                anomaly in the horizontal (X,Y) and vertical
  //                (Z) directions, dimension kk
  // v(,) - volume integrals for the polygon geometries, (kk,5)
  //

  float s[6];   // spare parts matrix for the volume integrals
  float aa1,aa2,aa3,aa4,aa5,aa6;
  float d1,d2,dd1,dd2,d12,dx,dy,dds,delx,dely,delz;
  float x1,y1,z1,x2,y2,z2,zf,zz,z1s,z2s;
  float r2s,r2z1,r2z2,gl,r1z1,rdz2,rdz1;
  float r1z2,zf1,zf2,fact,dtest,lk,ns;
```

```
float r1s,r1sf,rz1f,rz2f,x1f,y1f,p,ds;
float cost,sint,sncos,coss,ph;
float tjxm,tjym,tjzm,t12,fl;
int   n,nm,nend;
double xf,yf;

// part  of shortest length of body edges and thinnest layer.
float stest = shortSide*2.0e-4;
float ztest = zdel*2.0e-4;
float ztem = 10.0*ztest;

// for each field point
for (int k = 0 ; k < kk ; ++k)  {       // k <= kk
  xf = xp[k];
  yf = yp[k];
  zf = zp[k];
  delx = 0.0;
  dely = 0.0;
  delz = 0.0;
  for (int j = 0 ; j < 5 ; ++j)  {
    // Fortran Statement number 10
    s[j] = 0.0;
  } // end for (j)
  nend = 0;
  // number of bodies is mm
  for (int m = 0 ; m < mm ; ++m)  {
    zf1 = zee[m];
    zf2 = zm[m];
    z1 = zf1-zf;
    z2 = zf2-zf;
    if (abs(z1) < ztest)
      z1 = 0.0;
    if (abs(z2) < ztest)
      z2 = 0.0;
    fact = 1.0;
    zz = z1+z2;
    z1 = abs(z1);
    z2 = abs(z2);
    if ( zz < 0 )
```

```
  fact = -1.0;
if ( zz == 0 )
  break;       // end for (m)
if ( z1 > z2 )  {
  dtest = z2;
  z2 = z1;
  z1 = dtest;
}  // end if
lk = ll[m];
ns = nend+1;
nend = ns+lk-1;
z1s = z1*z1;
z2s = z2*z2;
aa1 = 0.0;
aa2 = 0.0;
aa3 = 0.0;
aa5 = 0.0;
aa6 = 0.0;
n = ns;

ns--;
n--;

x1 = xb[n]-xf;
y1 = yb[n]-yf;
if (abs(x1) < stest)
  x1 = 0.0;
if (abs(y1) < stest)
  y1 = 0.0;
r1s = x1*x1+y1*y1;
r1z1 = sqrt(r1s+z1s);
r1z2 = sqrt(r1s+z2s);
r1sf = r1s;
rz1f = r1z1;
rz2f = r1z2;
x1f = x1;
y1f = y1;

// number of line segments (vertical sides) or vertices =lk-1
```

```
for (int l = 0 ; l < lk ; ++l)  {
  nm = n;
  if (l == lk-1)  { // last side of the polygon
    x2 = x1f;
    y2 = y1f;
    r2s = r1sf;
    r2z1 = rz1f;
    r2z2 = rz2f;
  } else  {
    n = n+1;
    x2 = xb[n]-xf;
    y2 = yb[n]-yf;
    if (abs(x2) < stest)
      x2 = 0.0;
    if (abs(y2) < stest)
      y2 = 0.0;
    r2s = x2*x2+y2*y2;
    r2z1 = sqrt(r2s+z1s);
    r2z2 = sqrt(r2s+z2s);
  }  // end if (l == lk-1)
  dx = x2-x1;
  dy = y2-y1;
  dds = dx*dx+dy*dy;
  if ( dds > 0 )  {
    ds = dstnc[nm];
    cost = dy/ds;
    sint = dx/ds;
    sncos = sint*cost;
    coss = cost*cost;
    p = (x1*y2-x2*y1)/ds;
    // test if perpendicular distance to line
    // exceeds 1/5000 segment length
    if ( abs(p) < stest  )
      p = 0.0;

    // d1 and d2 are the lengths of the polygon side, bisected
    // by a line perpendicular to the side and through the
    // field point
    d1 = x1*sint+y1*cost;
```

```
d2 = x2*sint+y2*cost;
dd1 = abs(d1);
dd2 = abs(d2);
if (dd1 < stest)
  d1 = 0.0;
if (dd2 < stest)
  d2 = 0.0;
d12 = d1*d2;
// log(r+d) terms
if ( p != 0 ) {
  ph = p*p+z1s;
  rdz1 = r1z1+d1;
  if (d1 >= 0) {
    rdz2 = r2z1+d2;
    if (d2 < 0) {
      dtest = ph/(d2*d2);
      if (dtest < 0.01)
        rdz2 = dd2*0.5*dtest*(1.0-0.25*dtest);
    }  // end if (d2 < 0)
    gl = rdz1/rdz2;
  } else  {
    if (d2 < 0)  {        //d1<0
      gl = (r2z1-d2)/(r1z1-d1);
    } else  {     //D2=>0
      dtest = ph/(d1*d1);
      if (dtest < 0.01)
        rdz1 = dd1*0.5*dtest*(1.0-0.25*dtest);
      rdz2 = r2z1+d2;
      gl = rdz1/rdz2;
    }  // end if (d2 < 0)
  }  // end if (d1 => 0)

} else  {  //p = 0
  if (z1 != 0)  {
    ph = p*p+z1s;
    rdz1 = r1z1+d1;
    if (d1 >= 0) {
      rdz2 = r2z1+d2;
```

```
    if (d2 < 0)  {
      dtest = ph/(d2*d2);
      if (dtest < 0.01)
        rdz2 = dd2*0.5*dtest*(1.0-0.25*dtest);
    }  // end if (d2 < 0)

    gl = rdz1/rdz2;

  } else  {     //d1<0
    if (d2 < 0)  {
      gl = (r2z1-d2)/(r1z1-d1);
    } else  {   //D2=>0
      dtest = ph/(d1*d1);
      if (dtest < 0.01)
        rdz1 = dd1*0.5*dtest*(1.0-0.25*dtest);
      rdz2 = r2z1+d2;
      gl = rdz1/rdz2;
    }  // end if (d2 < 0)
  }  // end if (d1 => 0)

} else
  if (d12 <= 0)  {
    // raise z of field point slightly
    z1 = ztem;
    z1s = z1*z1;
    r1z1 = sqrt(r1s+z1s);
    r2z1 = sqrt(r2s+z1s);
    if (l == 0)
      rz1f = r1z1;
    ph = p*p+z1s;
    rdz1 = r1z1+d1;

    if (d1 >= 0)  {
      rdz2 = r2z1+d2;
      if (d2 < 0)  {
        dtest = ph/(d2*d2);
        if (dtest < 0.01)
          rdz2 = dd2*0.5*dtest*(1.0-0.25*dtest);
      }  // end if (d2 < 0)
```

```
        gl = rdz1/rdz2;

      } else {
        if (d2 < 0) {     //d1<0
          gl = (r2z1-d2)/(r1z1-d1);
        } else {  //D2=>0
          dtest = ph/(d1*d1);
          if (dtest < 0.01)
            rdz1 = dd1*0.5*dtest*(1.0-0.25*dtest);
          rdz2 = r2z1+d2;
          gl = rdz1/rdz2;
        } // end if (d2 < 0)
      } // end if (d1 => 0)

    } else if (d1 < 0) {
        gl = d2/d1;
    } else {
      gl = d1/d2;
    } // end if (d12 <= 0)
  } // end if (p != 0)


  gl = log(gl*(r2z2+d2)/(r1z2+d1));

  // log(r+z) terms
  fl = (z2+r2z2)/(z2+r1z2);
  if (0.0 < r1z1)
    fl = fl*(z1+r1z1);
  if (0.0 < r2z1)
    fl = fl/(z1+r2z1);
  fl = log(fl);
  // arctangent(zd/pr) terms
  t12 = 0.0;
  if (p != 0.0)
    tanget (p,z2*d2,r2z2,z1*d2,r2z1,z1*d1,r1z1,z2*d1,r1z2,&t12);
  aa1 = aa1-coss*t12+sncos*fl;
  aa2 = aa2+sncos*t12+coss*fl;
  aa3 = aa3+cost*gl;
  aa5 = aa5-sint*gl;
```

```
            aa6 = aa6+t12;

          } // end if (dds > 0)
          x1 = x2;
          y1 = y2;
          r1s = r2s;
          r1z1 = r2z1;
          r1z2 = r2z2;
          // end, corner loop
        } // end for (l)
        s[0] = s[0]+fact*aa1;
        s[1] = s[1]+fact*aa2;
        s[2] = s[2]+fact*aa3;
        s[3] = s[3]+fact*aa5;
        s[4] = s[4]+fact*aa6;
        aa4 = -(aa1+aa6);

        tjxm = tjx[m];
        tjym = tjy[m];
        tjzm = tjz[m];
        delx = delx+fact*(tjxm*aa1+tjym*aa2+tjzm*aa3);
        dely = dely+fact*(tjxm*aa2+tjym*aa4+tjzm*aa5);
        delz = delz+fact*(tjxm*aa3+tjym*aa5+tjzm*aa6);

      } // end for (m)
      // just saves the five volume integrals
      for (int n = 0 ; n < 5 ; ++n)  {
        // Fortran Statement number 240
        v[k][n] = s[n];
      } // end for (n)
      delta_X[k] = delx;
      delta_Y[k] = dely;
      delta_Z[k] = delz;
    } // end for (k)
    // EXIT SUB
  } // end calc


void tanget (float p, float zd1, float r1, float zd2, float r2,
```

```
        float zd3, float r3, float zd4, float r4, float *t12)  {
// arctangent addition with alternating terms negative
float zd[5], r[5];

r[1] = r1;
r[2] = r2;
r[3] = r3;
r[4] = r4;
zd[1] = zd1;
zd[2] = zd2;
zd[3] = zd3;
zd[4] = zd4;
float i = 1;
float sum = 0.0;
double tg = 0.0;
for (int j = 1 ; j <= 4 ; ++j)  {
  float fnum = zd[j];
  if (fnum != 0.0)  {
    // subroutine by-passed for p=0
    float f = fnum/(p*r[j]);
    if (i == -1)
      f = -f;
    fnum = tg+f;
    float den = 1.0e0-tg*f;
    if (den == 0)  {
      if (fnum < 0)  {
        sum = sum - 1.570796;
        tg = 0;
      } else  {
        sum = sum + 1.570796;
        tg = 0;
      }  // end if (den == 0)

    } else if (den < 0)  {
       tg = fnum/den;
       if (fnum < 0)  {
         sum = sum-3.14159265;
       } else if (fnum == 0)  {
         //just continue
```

```
            break;
          } else  {          //fnum>0
            sum = sum+3.14159265;
          }  // end if (fnum < 0)

        } else  {            // den>0
          tg = fnum/den;
        }  // end if (den < 0)

      }  // end if (fnum != 0.0)
      i = -i;
    }  // end for (j)
    *t12 = atan(tg)+sum;
}  // end tanget
```

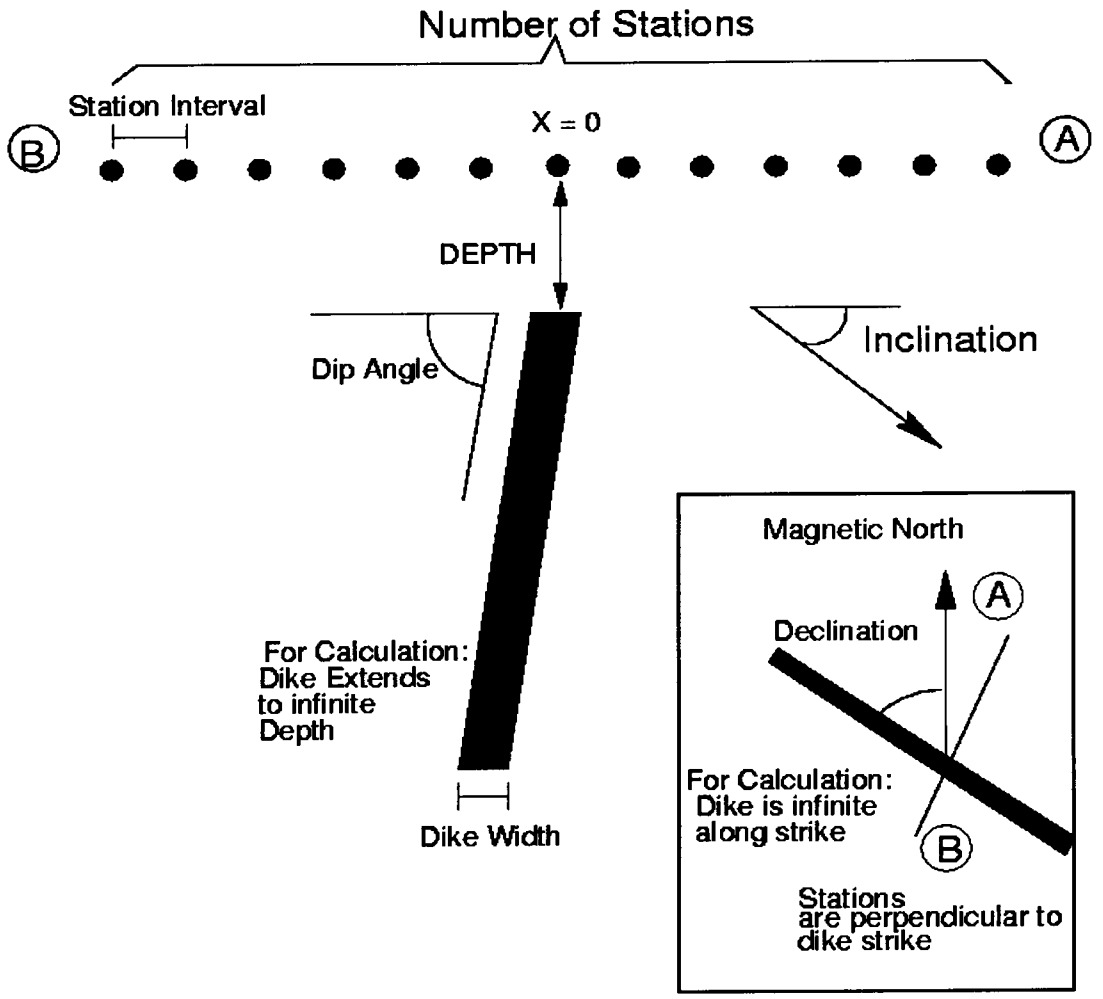
A method for calculating the magnetic anomaly over a thin igneous dike was originally proposed by Talwani and
Heirtzler (1964) and re-implemented by Ku and Sharp (1983, Geophysics vol 48, p 754-774). The following Java
Applet is used to calculate the magnetic anomaly associated with a thin, dipping, magnetized dike. The magnetic
anomaly is calculated along a profile perpendicular to the dike. The number of stations (points where the magnetic
anomaly is calculated) and distance between stations can be changed.

Additional Parameters can be altered, including: dike strike relative to magnetic north dike dip, dike depth, dike width,
inclination of vector of magnetization, and intensity of magnetization.

This code has been tested using data found in Gay (1961, Standard curves for interpretation of aeromagnetic
anomalies over long tabluar bodies Geophysics 28(2), 512-548.)

Scientific Notebook
20-1402-461
Inititals: CC

Chuck Connor
July 4,1999



The above graphic illustrates the physical meaning of parameters used to calculate the magnetic anomaly due to a dike. The
following parameters can be altered:

Number of stations - half of the stations are plotted left of the dike, and half to the right; increasing the number of stations
increases the resolution of the anomaly; note that the stations are arranged orthogonal to

the dike
Sample interval - distance, in meters, between stations
Dike depth - the depth, in meters, to the top of the dike
Dike Dip - the dip of the dike with respect to the horizontal, in degrees. Keep dike dips between 10 and 170 degrees
Intensity of Mag. - the intensity of magnetization of the dike, commonly a positive number between < 3 amp/m and
occasionally < 10 amp/m
Inclination - inclination of the Earth's magnetic field, in degrees
Declination, or azimuth, is the angle between the dike and magnetic north, in degrees. Note it is not the declination of the
Earth's magnetic field with respect to true north

The following is the java code to make the graphs

```
import java.awt.*;
import java.applet.*;

import graph.*;
/*
************************************************************************
**
**
**                Applet maggraph6
**
************************************************************************
**
**   Written by Chuck Connor July, 1999
**
**   This program calculates the magnetic anomaly over a thin vertical dike
************************************************************************
**
*
* This applet uses the MagneticDike class
```

```
* and plots the result.
*
**************************************************************************
*/

public class maggraph6 extends Applet {

    G2Dint graph        = new G2Dint();  // Graph class to do the plotting
    Axis xaxis;
    Axis yaxis;
    DataSet data;

    TextField sampleinput    = new TextField(5);      // station spacing input
    TextField depthinput  = new TextField(10);     // depth to top of dike input
    TextField widthinput  = new TextField(10);     // width of dike  input
    TextField stationinput    = new TextField(10);     // number of stations input
    TextField dipinput    = new TextField(10);     // dip of dike input
    TextField jinput    = new TextField(10);     // intensity of mag of dike input
    TextField incinput    = new TextField(10);     // magnetic field inclination input
    TextField decinput    = new TextField(10);     // magnetic field dec vis dike  input
    Button plot         = new Button("Plot It!"); // Button to plot it.




    public void init() {
      Label title          = new Label("Magnetic Anomaly Over a Thin
Dike",Label.CENTER);
      Panel panel          = new Panel();
      GridBagLayout gridbag  = new GridBagLayout();
      GridBagConstraints  c  = new GridBagConstraints();
      Font font            = new Font("Helvetica",Font.PLAIN,14);

      title.setFont(new Font("TimesRoman",Font.PLAIN,24));


      setLayout(new BorderLayout() );
      add("North",title);
```

```
add("Center",panel);

stationinput.setText(getParameter("STATIONS"));
sampleinput.setText(getParameter("SAMPLE"));
depthinput.setText(getParameter("DEPTH"));
widthinput.setText(getParameter("DWIDTH"));
dipinput.setText(getParameter("DDIP"));
jinput.setText(getParameter("JINTENSITY"));
incinput.setText(getParameter("MAGINC"));
decinput.setText(getParameter("MAGDEC"));



panel.setLayout(gridbag);

Label samplelabel   = new Label("Sample Interval");
Label depthlabel = new Label("Dike Depth");
Label widthlabel = new Label("Dike Width");
Label stationlabel   = new Label("Number of Stations");
Label diplabel   = new Label("Dike Dip");
Label jlabel   = new Label("Intensity of Mag.");
Label inclabel   = new Label("Inclination");
Label declabel   = new Label("Declination");

samplelabel.setFont(font);
stationlabel.setFont(font);
depthlabel.setFont(font);
widthlabel.setFont(font);
diplabel.setFont(font);
jlabel.setFont(font);
inclabel.setFont(font);
declabel.setFont(font);

sampleinput.setFont(font);
sampleinput.setBackground(Color.lightGray);
stationinput.setFont(font);
stationinput.setBackground(Color.lightGray);
depthinput.setFont(font);
depthinput.setBackground(Color.lightGray);
```

```
widthinput.setFont(font);
widthinput.setBackground(Color.lightGray);
dipinput.setFont(font);
dipinput.setBackground(Color.lightGray);
jinput.setFont(font);
jinput.setBackground(Color.lightGray);
incinput.setFont(font);
incinput.setBackground(Color.lightGray);
decinput.setFont(font);
decinput.setBackground(Color.lightGray);
plot.setFont(font);
plot.setBackground(Color.green);

c.weightx = 1.0;
c.weighty = 1.0;
c.gridwidth = 4;
c.gridwidth=GridBagConstraints.REMAINDER;
c.fill = GridBagConstraints.BOTH;

gridbag.setConstraints(graph,c);

c.fill = GridBagConstraints.NONE;
c.weightx=0.0;
c.weighty=0.0;
c.gridheight=1;

c.gridwidth=1;
c.anchor = GridBagConstraints.EAST;
gridbag.setConstraints(stationlabel,c);

c.anchor = GridBagConstraints.CENTER;
c.gridwidth=GridBagConstraints.RELATIVE;
c.fill = GridBagConstraints.HORIZONTAL;
gridbag.setConstraints(stationinput,c);

c.fill = GridBagConstraints.NONE;
c.gridwidth=GridBagConstraints.REMAINDER;

gridbag.setConstraints(plot,c);
```

```
c.gridwidth=1;
c.anchor = GridBagConstraints.EAST;
gridbag.setConstraints(samplelabel,c);
c.gridwidth=2;
c.anchor = GridBagConstraints.WEST;
c.gridwidth=GridBagConstraints.REMAINDER;
gridbag.setConstraints(sampleinput,c);

c.gridwidth=1;
c.anchor = GridBagConstraints.EAST;
gridbag.setConstraints(depthlabel,c);
c.gridwidth=2;
c.anchor = GridBagConstraints.WEST;
c.gridwidth=GridBagConstraints.REMAINDER;
gridbag.setConstraints(depthinput,c);

c.gridwidth=1;
c.anchor = GridBagConstraints.EAST;
gridbag.setConstraints(widthlabel,c);
c.gridwidth=2;
c.anchor = GridBagConstraints.WEST;
c.gridwidth=GridBagConstraints.REMAINDER;
gridbag.setConstraints(widthinput,c);

c.gridwidth=1;
c.anchor = GridBagConstraints.EAST;
gridbag.setConstraints(diplabel,c);
c.gridwidth=2;
c.anchor = GridBagConstraints.WEST;
c.gridwidth=GridBagConstraints.REMAINDER;
gridbag.setConstraints(dipinput,c);

c.gridwidth=1;
c.anchor = GridBagConstraints.EAST;
gridbag.setConstraints(jlabel,c);
c.gridwidth=2;
c.anchor = GridBagConstraints.WEST;
c.gridwidth=GridBagConstraints.REMAINDER;
```

```
gridbag.setConstraints(jinput,c);

c.gridwidth=1;
c.anchor = GridBagConstraints.EAST;
gridbag.setConstraints(inclabel,c);
c.gridwidth=2;
c.anchor = GridBagConstraints.WEST;
c.gridwidth=GridBagConstraints.REMAINDER;
gridbag.setConstraints(incinput,c);

c.gridwidth=1;
c.anchor = GridBagConstraints.EAST;
gridbag.setConstraints(declabel,c);
c.gridwidth=2;
c.anchor = GridBagConstraints.WEST;
c.gridwidth=GridBagConstraints.REMAINDER;
gridbag.setConstraints(decinput,c);


panel.add(graph);
panel.add(stationlabel);
panel.add(stationinput);
panel.add(plot);
panel.add(samplelabel);
panel.add(sampleinput);
panel.add(depthlabel);
panel.add(depthinput);
panel.add(widthlabel);
panel.add(widthinput);
panel.add(diplabel);
panel.add(dipinput);
panel.add(jlabel);
panel.add(jinput);
panel.add(inclabel);
panel.add(incinput);
panel.add(declabel);
panel.add(decinput);
```

```
    xaxis = graph.createXAxis();
    xaxis.setTitleText("Distance From Dike (m)");
   xaxis.setTitleFont(new Font("Helvetica",Font.PLAIN,18));
   xaxis.setLabelFont(new Font("Helvetica",Font.PLAIN,14));


    yaxis = graph.createYAxis();
    yaxis.setTitleText("nT");
    yaxis.setTitleFont(new Font("Helvetica",Font.PLAIN,18));
    yaxis.setLabelFont(new Font("Helvetica",Font.PLAIN,14));


    data = new DataSet();

    xaxis.attachDataSet(data);
    yaxis.attachDataSet(data);
    graph.attachDataSet(data);


    graph.setDataBackground(new Color(255,200,175));
    graph.setBackground(new Color(200,150,100));

    plot();

  }



  void plot() {
    int total_stations;
    double sample_interval;
    double ddepth, halfwidth, ddip, dintensity, maginc, magdec;
    boolean error = false;

    try {
       sample_interval = Double.valueOf(sampleinput.getText()).doubleValue();
    } catch(Exception e) {
       this.showStatus("Error with sample interval!");
       System.out.println("Sample interval error "+e.getMessage());
```

```
            return;
        }

        try {
            total_stations  = Integer.parseInt(stationinput.getText());
        } catch(Exception e) {
            this.showStatus("Error with number of stations!");
            System.out.println("number of stations  error "+e.getMessage());
            return;
        }

        try {
          halfwidth = (Double.valueOf(widthinput.getText()).doubleValue())/2.0;
        } catch(Exception e) {
            this.showStatus("Error with dike width  value!");
            System.out.println("dike width error "+e.getMessage());
            return;
        }

        try {
          ddepth = Double.valueOf(depthinput.getText()).doubleValue();
        } catch(Exception e) {
            this.showStatus("Error with depth value!");
            System.out.println("dike depth error "+e.getMessage());
            return;
        }
      try {
          ddip= Double.valueOf(dipinput.getText()).doubleValue();
        } catch(Exception e) {
            this.showStatus("Error with dip value!");
            System.out.println("dip error "+e.getMessage());
            return;
        }

      try {
          dintensity= 400.0*3.1415927*Double.valueOf(jinput.getText()).doubleValue();
        } catch(Exception e) {
            this.showStatus("Error with magnetization!");
            System.out.println("magnetization error "+e.getMessage());
```

```
            return;
          }

        try {
            maginc = Double.valueOf(incinput.getText()).doubleValue();
          } catch(Exception e) {
            this.showStatus("Error with inclination value!");
            System.out.println("inclination error "+e.getMessage());
            return;
          }

        try {
            magdec = Double.valueOf(decinput.getText()).doubleValue();
          } catch(Exception e) {
            this.showStatus("Error with declination value!");
            System.out.println("declination error "+e.getMessage());
            return;
          }



          this.showStatus("Calculating points!");


          int j, kount;

          double d[] = new double[2*total_stations];

          MagneticDike dike1 = new
    MagneticDike(maginc,magdec,ddepth,halfwidth,ddip,dintensity);

        for (kount=j = 0; kount <= total_stations; kount++, j +=2) {
          try {
            d[j] = ((double)kount - (double)total_stations/2.0)*sample_interval;
            d[j+1] = dike1.anomaly2(d[j]);

          } catch (Exception e) {error = true;}
```

```
    }


    /**
     if(total_stations <= 2) {
        this.showStatus("Error NO POINTS to PLOT!");
        System.out.println("Error NO POINTS to PLOT!");
        return;
     } else
     if( error ) {
        this.showStatus("Error while Calculating points!");
        System.out.println("Error while calculating points!");
     }
    **/


     data.deleteData();

     try {
         data.append(d,total_stations);
     } catch(Exception e) {
        this.showStatus("Error while appending data!");
        System.out.println("Error while appending data!");
        return;
     }


     graph.repaint();
   }


  public boolean action(Event e, Object a) {

     if(e.target instanceof Button) {
        if( plot.equals(e.target) ) {
            plot();
            return true;
```

```
        }
      }

      return false;
    }



}
```

```
import java.io.*;
import java.math.*;

/*****************************************************
* Dike calculates the magnetic anomaly expected from *
* a magnetized dike                                  *
*                                                    *
* Abstract                                           *
*   This code uses the method originally proposed    *
* by Talwani and Heirtzler (1964) and re-implmented  *
* by Ku and Sharp (1983, Geophysics vol 48, p 754-774)*
* to calculate the magnetic anomaly associated with a *
* thin dike, dipping, magnetized dike. The magnetic  *
* anomaly is calculated along a profile perpendicular *
* to the dike.                                       *
*                                                    *
* variables can be altered as described below        *
* including: dike strike relative to magnetic north  *
* dike dip, dike depth, dike half-width,             *
* inclination of vector of magnetization, and        *
* intensity of magnetization                         *
*                                                    *
* The code outputs x, total field magnetic anomaly,  *
```

```
* horizontal component of magnetic anomaly, and      *
* vertical component of magnetic anomaly.           *
*                                                    *
* This code has been tested using data found in      *
* Gay (1961, Standard curves for interpreation       *
* of aeromagnetic anomalies over long tabluar bodies *
* Geophysics 28(2), 512-548.)                        *
*                                                    *
* Written by C.B. Connor in May, 1996                *
*        Div. 20                                     *
*        Southwest Res. Inst.                        *
*        6220 Culebra Rd                             *
*        San Antonio, TX 78238                       *
*        210-522-6649                                *
*        cconnor@swri.edu                            *
/*****************************************************/
public class MagneticDike {
  public double incl_vec, dec_vec,depth, half_width, dip_angle, J_intensity;

  /** where incl_vec -  is the inclination of the vector of magnetization
    in degrees, positiv downward (degrees)
    dec_vec =  is the strike of the dike wrt declination (degrees)
    of the vector of magnetization
    depth - depth to the top of the dike (meters)
    half_width = 1/2 the thickness of the dike (meters)
    dip_angle = the dip of the dike (degrees) dip_angle
    is the dip of the dike, dip_angle should vary from 0 to 180,
    with values less than 90 in the direction of magnetization and
    values greater than 90 dipping away from the
    direction of the magnetization vector, j, 90.0 is a  vertical dike
    J_intensity - intensity of magnetization of the dike (nt) divide
    by 400 * pi to obtain amp/m.
  **/

  public MagneticDike (double incl_vec, double dec_vec, double depth,
            double half_width, double dip_angle, double J_intensity){
    this.incl_vec = incl_vec;
    this.dec_vec = dec_vec;
    this.depth = depth;
```

```
    this.half_width = half_width;
    this.dip_angle = dip_angle;
    this.J_intensity = J_intensity;

  }

  // this method calculates the magnetic anomaly and prints a list
  public void anomaly1 () {

    double jx;
    double jz,  a,  b, x;
    double T, vmag, hmag, rad_change;
    double ia, theta,kount;

    rad_change = 3.14159/180.0;

    // change degrees to radians
    incl_vec *= rad_change;
    dec_vec *= rad_change;
    dip_angle *= rad_change;

    /* alpha == 0 results in undefined angle ia */
    if (dec_vec == 0.0) dec_vec = 0.001;

    /* ia is the component of the inclination of the
       vector of magnetization in the direction of x */
    ia = Math.atan2(Math.tan(incl_vec),Math.sin(dec_vec));

    /* theta is the angle between the dipping
       dike and the inclination of the vector of
       magnetization projected in the x direction */
    theta = dip_angle - ia;

    /* compute the components of magnetization wrt theta */
    jx = J_intensity*Math.sin(theta);
    jz = J_intensity*Math.cos(theta);


    /* calculate the total, horizontal, and vertical
```

```
        magnetic field at points along the profile */
      for (kount = 0; kount <= 2000; kount++) {

        x = ((double) kount - 1000.0) * 10.0;
        a = -2.0*half_width*(jx*Math.sin(incl_vec) +
jz*Math.cos(incl_vec)*Math.sin(dec_vec));
        b = 2.0*half_width*(-jx*Math.cos(incl_vec)*Math.sin(dec_vec) +
jz*Math.sin(incl_vec));

        T = (a*x + b*depth)/(x*x + depth*depth);
        hmag = -2*half_width*(jx*depth + jz*x)/(x*x+depth*depth);
        vmag = 2*half_width*(jz*depth - jx*x)/(x*x+depth*depth);

        /* print the results */
        if (x < 100.0 && x > -100.0)
        System.out.println(x + " ," + T + ", " + hmag + "' " + vmag);


    } /* end of the kount loop */

  }
  // this method calculates the magnetic anomaly and returns
  // the value of the total magnetic field for a passed value x
  public double anomaly2 (double x) {

    double jx;
    double jz,  a,  b;
    double T, vmag, hmag, rad_change;
    double ia, theta,kount;
    double ivec,dvec,dipa;

    rad_change = 3.14159/180.0;

    // change degrees to radians
    ivec = incl_vec * rad_change;
    dvec = dec_vec * rad_change;
    dipa = dip_angle * rad_change;

    /* alpha == 0 results in undefined angle ia */
```

```
        if (dvec == 0.0) dvec = 0.001;

        /* ia is the component of the inclination of the
           vector of magnetization in the direction of x */
        ia = Math.atan2(Math.tan(ivec),Math.sin(dvec));

        /* theta is the angle between the dipping
           dike and the inclination of the vector of
           magnetization projected in the x direction */
        theta = dipa - ia;

        /* compute the components of magnetization wrt theta */
        jx = J_intensity*Math.sin(theta);
        jz = J_intensity*Math.cos(theta);


        /** calculate the total, horizontal, and vertical
          magnetic field at points along the profile */



        a = -2.0*half_width*(jx*Math.sin(ivec) + jz*Math.cos(ivec)*Math.sin(dvec));
        b = 2.0*half_width*(-jx*Math.cos(ivec)*Math.sin(dvec) + jz*Math.sin(ivec));

        T  = (a*x + b*depth)/(x*x + depth*depth);
        hmag = -2*half_width*(jx*depth + jz*x)/(x*x+depth*depth);
        vmag = 2*half_width*(jz*depth - jx*x)/(x*x+depth*depth);

        return T;



    }
}
```

Scientific Notebook
20-1402-461
Inititals: CC

Chuck Connor
July 4,1999

Program maggraph6.java, version 1.0

Written in ansi C by C.B. Connor in May, 1996
Translated to Java by C.B. Connor, June, 1999
This code uses the graph package for Java developed by Leigh Brookshaw

This code was developed at the Center for Nuclear Waste Regulatory Analyses (CNWRA) with funding from the U.S. Nuclear Regulatory Commission (NRC) under contract NRC-02-97-009. This code is an independent product of the CNWRA and does not necessarily reflect the views or regulatory position of the NRC.

```
import java.io.*;
import java.math.*;

/*******************************************************
 * Dike calculates the magnetic anomaly expected from  *
 *  a magnetized dike                                  *
 *                                                     *
 * Abstract                                            *
 *     This code uses the method originally proposed   *
 *  by Talwani and Heirtzler (1964) and re-implmented  *
 *  by Ku and Sharp (1983, Geophysics vol 48, p 754-774)*
 *  to calculate the magnetic anomaly associated with a *
 *  thin dike, dipping, magnetized dike. The magnetic  *
 *  anomaly is calculated along a profile perpendicular *
 *  to the dike.                                       *
 *                                                     *
 *  variables can be altered as described below        *
 *  including: dike strike relative to magnetic north  *
 *  dike dip, dike depth, dike half-width,             *
 *  inclination of vector of magnetization, and        *
 *  intensity of magnetization                         *
 *                                                     *
 *  The code outputs x, total field magnetic anomaly,  *
 *  horizontal component of magnetic anomaly, and      *
 *  vertical component of magnetic anomaly.            *
 *                                                     *
 *  This code has been tested using data found in      *
 *  Gay (1961, Standard curves for interpreation       *
 *  of aeromagnetic anomalies over long tabluar bodies *
 *  Geophysics 28(2), 512-548.)                        *
 *                                                     *
 *  Written by C.B. Connor in May, 1996                *
 *              Div. 20                                *
 *              Southwest Res. Inst.                   *
 *              6220 Culebra Rd                        *
 *              San Antonio, TX 78238                  *
 *              210-522-6649                           *
 *              cconnor@swri.edu                       *
 *******************************************************/
public class MagneticDike {
    public double incl_vec, dec_vec,depth, half_width, dip_angle, J_intensity;

    /** where incl_vec -  is the inclination of the vector of magnetization
        in degrees, positiv downward (degrees)
        dec_vec =  is the strike of the dike wrt declination (degrees)
        of the vector of magnetization
        depth - depth to the top of the dike (meters)
        half_width = 1/2 the thickness of the dike (meters)
        dip_angle = the dip of the dike (degrees) dip_angle
        is the dip of the dike, dip_angle should vary from 0 to 180,
        with values less than 90 in the direction of magnetization and
        values greater than 90 dipping away from the
        direction of the magnetization vector, j, 90.0 is a  vertical dike
        J_intensity - intensity of magnetization of the dike (nt) divide
        by 400 * pi to obtain amp/m.
    **/

    public MagneticDike (double incl_vec, double dec_vec, double depth,
                         double half_width, double dip_angle, double J_intensity){
        this.incl_vec = incl_vec;
        this.dec_vec = dec_vec;
        this.depth = depth;
        this.half_width = half_width;
        this.dip_angle = dip_angle;
```

```
        this.J_intensity = J_intensity;

    }

    // this method calculates the magnetic anomaly and prints a list
    public void anomaly1 () {

        double jx;
        double jz,  a,  b, x;
        double T, vmag, hmag, rad_change;
        double ia, theta,kount;

        rad_change = 3.14159/180.0;

        // change degrees to radians
        incl_vec *= rad_change;
        dec_vec *= rad_change;
        dip_angle *= rad_change;

        /* alpha == 0 results in undefined angle ia */
        if (dec_vec == 0.0) dec_vec = 0.001;

        /* ia is the component of the inclination of the
           vector of magnetization in the direction of x */
        ia = Math.atan2(Math.tan(incl_vec),Math.sin(dec_vec));

        /* theta is the angle between the dipping
           dike and the inclination of the vector of
           magnetization projected in the x direction */
        theta = dip_angle - ia;

        /* compute the components of magnetization wrt theta */
        jx = J_intensity*Math.sin(theta);
        jz = J_intensity*Math.cos(theta);


        /* calculate the total, horizontal, and vertical
           magnetic field at points along the profile */
        for (kount = 0; kount <= 2000; kount++) {

            x = ((double) kount - 1000.0) * 10.0;
            a = -2.0*half_width*(jx*Math.sin(incl_vec) + jz*Math.cos(incl_vec)*Math.sin(dec_v
+ ec));
            b = 2.0*half_width*(-jx*Math.cos(incl_vec)*Math.sin(dec_vec) + jz*Math.sin(incl_v
+ ec));

            T = (a*x + b*depth)/(x*x + depth*depth);
            hmag = -2*half_width*(jx*depth + jz*x)/(x*x+depth*depth);
            vmag = 2*half_width*(jz*depth - jx*x)/(x*x+depth*depth);

            /* print the results */
            if (x < 100.0 && x > -100.0)
            System.out.println(x + " ," + T + ", " + hmag + "' " + vmag);


        } /* end of the kount loop */

    }
    // this method calculates the magnetic anomaly and returns
    // the value of the total magnetic field for a passed value x
    public double anomaly2 (double x) {

        double jx;
        double jz,  a,  b;
```

```
        double T, vmag, hmag, rad_change;
        double ia, theta,kount;
        double ivec,dvec,dipa;

        rad_change = 3.14159/180.0;

        // change degrees to radians
        ivec = incl_vec * rad_change;
        dvec = dec_vec * rad_change;
        dipa = dip_angle * rad_change;

        /* alpha == 0 results in undefined angle ia */
        if (dvec == 0.0) dvec = 0.001;

        /* ia is the component of the inclination of the
           vector of magnetization in the direction of x */
        ia = Math.atan2(Math.tan(ivec),Math.sin(dvec));

        /* theta is the angle between the dipping
           dike and the inclination of the vector of
           magnetization projected in the x direction */
        theta = dipa - ia;

        /* compute the components of magnetization wrt theta */
        jx = J_intensity*Math.sin(theta);
        jz = J_intensity*Math.cos(theta);


        /** calculate the total, horizontal, and vertical
          magnetic field at points along the profile */



        a = -2.0*half_width*(jx*Math.sin(ivec) + jz*Math.cos(ivec)*Math.sin(dvec));
        b = 2.0*half_width*(-jx*Math.cos(ivec)*Math.sin(dvec) + jz*Math.sin(ivec));

        T  = (a*x + b*depth)/(x*x + depth*depth);
        hmag = -2*half_width*(jx*depth + jz*x)/(x*x+depth*depth);
        vmag = 2*half_width*(jz*depth - jx*x)/(x*x+depth*depth);

        return T;



    }
}
```

```
import java.io.*;
import java.math.*;

 **
 * This is an attempt to translate a program written by
 * Charles Connor in True Basic.
 **/
public class MagneticCube {
  public double a1, a2, b1, b2, h1, h2, theta, I, D, Io, Do, EI;

  /** a1 and a2 are the width (a1,a2) a1=300.0001, and a2=600.5001
   *  b1 and b2 are the length (b1,b2) b1=300.00001, and b2=600.50001
   *  h1 and h2 are the depth (h1,h2) h1=100, and h2=5000
   *  theta is the angle from north, in degrees theta=0
   *  I is the inclination of the Earth's field I=45
   *  D is the declination of the Earth's field D=0
   *  Io is the inclination of the vector of magnetization Io=45
   *  Do is the declination of the vector of magnetization Do=0
   *  EI is the intensity of magnetization EI=500
   **/

  public MagneticCube(double a1, double a2, double b1, double b2,
                      double h1, double h2, double theta, double I,
                      double D, double Io, double Do, double EI) {
    this.a1 = a1;
    this.a2 = a2;
    this.b1 = b1;
    this.b2 = b2;
    this.h1 = h1;
    this.h2 = h2;
    this.theta = theta;
    this.I = I;
    this.D = D;
    this.Io = Io;
    this.Do = Do;
    this.EI = EI;
  }

  // This method calculates the magnetic anomaly and prints...
  // ...a list
  public void anomaly1 () {

  // Initialize some variables
  double a1=300.0001;
  double b1=300.00001;
  double h1=100;
  double h2=5000;
  double theta=0;
  double I=45;
  double D=0;
  double Io=45;
  double Do=0;
  double EI=500;
  double pi = 3.14159, rad_change;

  rad_change = pi/180;

  // Change degrees to radians
  theta *= rad_change;
  I *= rad_change;
  D *= rad_change;
  Io *= rad_change;
  Do *= rad_change;
```

```
   // Initialize some more variables
   double p, q, r;

   // Calculate the earth's inclination and declination wrt theta
   p = Math.cos(I)*Math.cos(D-theta);
   q = Math.cos(I)*Math.sin(D-theta);
   r = Math.sin(I);

   // Initialize some more variables
   double L, M, N;

   // Calculate the vector of magnetization's inclination and...
   //...declination
   L = Math.cos(Io)*Math.cos(Do-theta);
   M = Math.cos(Io)*Math.sin(Do-theta);
   N = Math.sin(Io);

   // Initialize some more variables
   double g1, g2, g3, g4, g5;

   // Calculates the direction cosines
   g1 = EI*(M*r + N*q);
   g2 = EI*(L*r + N*p);
   g3 = EI*(L*q + M*p);
   g4 = EI*(N*r - M*q);
   g5 = EI*(N*r - L*p);

   // Initialize more variables
   int kount, x, y;

   /**
   * Calculates the magnetic anomaly and returns the value
   * of the total magnetic field.
   **/
   for(kount = 1; kount <= 1024; kount++) {

   //Initialize some variables
   x = kount;
   y = 500;

   double xp = x*Math.cos(theta) + y*Math.sin(theta);
   double yp = -x*Math.sin(theta) + y*Math.cos(theta);

   double ap1 = a1 - xp;
   double ap2 = a2 - xp;
   double bp1 = b1 - yp;
   double bp2 = b2 - yp;

   double r1 = Math.sqrt((ap1*ap1) + (bp1*bp1) + (h1*h1));
   double r2 = Math.sqrt((ap1*ap1) + (bp1*bp1) + (h2*h2));
   double r3 = Math.sqrt((ap2*ap2) + (bp1*bp1) + (h1*h1));
   double r4 = Math.sqrt((ap2*ap2) + (bp1*bp1) + (h2*h2));
   double r5 = Math.sqrt((ap1*ap1) + (bp2*bp2) + (h1*h1));
   double r6 = Math.sqrt((ap1*ap1) + (bp2*bp2) + (h2*h2));
   double r7 = Math.sqrt((ap2*ap2) + (bp2*bp2) + (h1*h1));
   double r8 = Math.sqrt((ap2*ap2) + (bp2*bp2) + (h2*h2));

   double f1 = (r2 + ap1)*(r3 + ap2)*(r5 + ap1)*(r8 + ap2)/
               ((r1 + ap1)*(r4 + ap2)*(r6 + ap1)*(r7 + ap2));
   double f2 = (r2 + bp1)*(r3 + bp1)*(r5 + bp2)*(r8 + bp2)/
               ((r1 + bp1)*(r4 + bp1)*(r6 + bp2)*(r7 + bp2));
   double f3 = (r2 + h2)*(r3 + h1)*(r5 + h1)*(r8 + h2)/
               ((r1 + h1)*(r4 + h2)*(r6 + h2)*(r7 + h1));
   double f4 = Math.atan(ap2*h2/(r8*bp2))-Math.atan(ap1*h2/(r6*bp2))-
```

```
                Math.atan(ap2*h2/(r4*bp1))+Math.atan(ap1*h2/(r2*bp1))-
                Math.atan(ap2*h1/(r7*bp2))+Math.atan(ap1*h1/(r5*bp2))+
                Math.atan(ap2*h1/(r3*bp1))-Math.atan(ap1*h1/(r1*bp1));
    double f5 = Math.atan(bp2*h2/(r8*ap2))-Math.atan(bp2*h2/(r6*ap1))-
                Math.atan(bp1*h2/(r4*ap2))+Math.atan(bp1*h2/(r2*ap1))-
                Math.atan(bp2*h1/(r7*ap2))+Math.atan(bp2*h1/(r5*ap1))+
                Math.atan(bp1*h1/(r3*ap2))-Math.atan(bp1*h1/(r1*ap1));

    /** The Math.log function returns a natural (base e) log.  Convert
      *  natural logs to base 10 logs with code like this:
      *  double nat_log = ...
      *
      *  double base10log = nat_log / Math.log(10.0);
      **/

    double T = g1*(Math.log(f1)/Math.log(10.0))+g2*(Math.log(f2)/Math.log(10.0))+
               g3*(Math.log(f3)/Math.log(10.0))+g4*(Math.log(f4)/Math.log(10.0))+
               g5*(Math.log(f5)/Math.log(10.0));

    // Print the results
    System.out.println(x + "," + T + ",");
    } // End of kount loop
}

    // This method calculates the magnetic anomaly and returns...
    // ...the value of the total magnetic field
    public double anomaly2 (double x) {

    // Initialize some variables
    double a1=300.0001;
    double b1=300.00001;
    double h1=100;
    double h2=5000;
    double theta=0;
    double I=45;
    double D=0;
    double Io=45;
    double Do=0;
    double EI=500;
    double pi = 3.14159, rad_change;

    rad_change = pi/180;

    // Change degrees to radians
    theta *= rad_change;
    I *= rad_change;
    D *= rad_change;
    Io *= rad_change;
    Do *= rad_change;

    // Initialize some more variables
    double p, q, r;

    // Calculate the earth's inclination and declination wrt theta
    p = Math.cos(I)*Math.cos(D-theta);
    q = Math.cos(I)*Math.sin(D-theta);
    r = Math.sin(I);

    // Initialize some more variables
    double L, M, N;

    // Calculate the vector of magnetization's inclination and...
    //...declination wrt theta
    L = Math.cos(Io)*Math.cos(Do-theta);
```

```
   M = Math.cos(Io)*Math.sin(Do-theta);
   N = Math.sin(Io);

   // Initialize some more variables
   double g1, g2, g3, g4, g5;

   // Calculates ????
   g1 = EI*(M*r + N*q);
   g2 = EI*(L*r + N*p);
   g3 = EI*(L*q + M*p);
   g4 = EI*(N*r - M*q);
   g5 = EI*(N*r - L*p);

   // Initialize more variables
   int kount, y;

   /**
   * Calculates the magnetic anomaly and returns the value
   * of the total magnetic field.
   **/

   //Initialize some variables
   y = 500;

   double xp = x*Math.cos(theta) + y*Math.sin(theta);
   double yp = -x*Math.sin(theta) + y*Math.cos(theta);

   double ap1 = a1 - xp;
   double ap2 = a2 - xp;
   double bp1 = b1 - yp;
   double bp2 = b2 - yp;

   double r1 = Math.sqrt((ap1*ap1) + (bp1*bp1) + (h1*h1));
   double r2 = Math.sqrt((ap1*ap1) + (bp1*bp1) + (h2*h2));
   double r3 = Math.sqrt((ap2*ap2) + (bp1*bp1) + (h1*h1));
   double r4 = Math.sqrt((ap2*ap2) + (bp1*bp1) + (h2*h2));
   double r5 = Math.sqrt((ap1*ap1) + (bp2*bp2) + (h1*h1));
   double r6 = Math.sqrt((ap1*ap1) + (bp2*bp2) + (h2*h2));
   double r7 = Math.sqrt((ap2*ap2) + (bp2*bp2) + (h1*h1));
   double r8 = Math.sqrt((ap2*ap2) + (bp2*bp2) + (h2*h2));

   double f1 = (r2 + ap1)*(r3 + ap2)*(r5 + ap1)*(r8 + ap2)/
               ((r1 + ap1)*(r4 + ap2)*(r6 + ap1)*(r7 + ap2));
   double f2 = (r2 + bp1)*(r3 + bp1)*(r5 + bp2)*(r8 + bp2)/
               ((r1 + bp1)*(r4 + bp1)*(r6 + bp2)*(r7 + bp2));
   double f3 = (r2 + h2)*(r3 + h1)*(r5 + h1)*(r8 + h2)/
               ((r1 + h1)*(r4 + h2)*(r6 + h2)*(r7 + h1));
   double f4 = Math.atan(ap2*h2/(r8*bp2))-Math.atan(ap1*h2/(r6*bp2))-
               Math.atan(ap2*h2/(r4*bp1))+Math.atan(ap1*h2/(r2*bp1))-
               Math.atan(ap2*h1/(r7*bp2))+Math.atan(ap1*h1/(r5*bp2))+
               Math.atan(ap2*h1/(r3*bp1))-Math.atan(ap1*h1/(r1*bp1));
   double f5 = Math.atan(bp2*h2/(r8*ap2))-Math.atan(bp2*h2/(r6*ap1))-
               Math.atan(bp1*h2/(r4*ap2))+Math.atan(bp1*h2/(r2*ap1))-
               Math.atan(bp2*h1/(r7*ap2))+Math.atan(bp2*h1/(r5*ap1))+
               Math.atan(bp1*h1/(r3*ap2))-Math.atan(bp1*h1/(r1*ap1));

   /** The Math.log function returns a natural (base e) log.  Convert
      *  natural logs to base 10 logs with code like this:
      *  double nat_log = ...
      *
      *  double base10log = nat_log / Math.log(10.0);
      **/

   double T = g1*(Math.log(f1)/Math.log(10.0))+g2*(Math.log(f2)/Math.log(10.0))+
```

```
            g3*(Math.log(f3)/Math.log(10.0))+g4*(Math.log(f4)/Math.log(10.0))+
            g5*(Math.log(f5)/Math.log(10.0));

    return T;
    }
}
```

```
import java.awt.*;
import java.applet.*;

port graph.*;

/**************************************************************************
**
**                      Applet maggraph6
**
**************************************************************************
**    Written by Chuck Connor July, 1999
**
**    This program calculates the magnetic anomaly over a thin vertical dike
**************************************************************************
*
* This applet uses the MagneticDike class
* and plots the result.
*
**************************************************************************/

public class maggraph6 extends Applet {

      G2Dint graph         = new G2Dint();   // Graph class to do the plotting
      Axis xaxis;
      Axis yaxis;
      DataSet data;

      TextField sampleinput     = new TextField(5);       // station spacing input
      TextField depthinput   = new TextField(10);      // depth to top of dike input
      TextField widthinput   = new TextField(10);      // width of dike  input
      TextField stationinput     = new TextField(10);      // number of stations input
      TextField dipinput     = new TextField(10);      // dip of dike input
      TextField jinput     = new TextField(10);      // intensity of mag of dike input
      TextField incinput     = new TextField(10);      // magnetic field inclination input
      TextField decinput     = new TextField(10);      // magnetic field dec vis dike  input
      Button plot          = new Button("Plot It!"); // Button to plot it.




      public void init() {
         Label title            = new Label("Magnetic Anomaly Over a Thin Dike",Label.CENTER)
+ ;
         Panel panel            = new Panel();
         GridBagLayout gridbag  = new GridBagLayout();
         GridBagConstraints  c  = new GridBagConstraints();
         Font font              = new Font("Helvetica",Font.PLAIN,14);

         title.setFont(new Font("TimesRoman",Font.PLAIN,24));


         setLayout(new BorderLayout() );
         add("North",title);
         add("Center",panel);

         stationinput.setText(getParameter("STATIONS"));
         sampleinput.setText(getParameter("SAMPLE"));
         depthinput.setText(getParameter("DEPTH"));
         widthinput.setText(getParameter("DWIDTH"));
         dipinput.setText(getParameter("DDIP"));
         jinput.setText(getParameter("JINTENSITY"));
         incinput.setText(getParameter("MAGINC"));
         decinput.setText(getParameter("MAGDEC"));
```

```
        panel.setLayout(gridbag);

        Label samplelabel   = new Label("Sample Interval");
        Label depthlabel = new Label("Dike Depth");
        Label widthlabel = new Label("Dike Width");
        Label stationlabel   = new Label("Number of Stations");
        Label diplabel   = new Label("Dike Dip");
        Label jlabel   = new Label("Intensity of Mag.");
        Label inclabel   = new Label("Inclination");
        Label declabel   = new Label("Declination");

        samplelabel.setFont(font);
        stationlabel.setFont(font);
        depthlabel.setFont(font);
        widthlabel.setFont(font);
        diplabel.setFont(font);
        jlabel.setFont(font);
        inclabel.setFont(font);
        declabel.setFont(font);

        sampleinput.setFont(font);
        sampleinput.setBackground(Color.lightGray);
        stationinput.setFont(font);
        stationinput.setBackground(Color.lightGray);
        depthinput.setFont(font);
        depthinput.setBackground(Color.lightGray);
        widthinput.setFont(font);
        widthinput.setBackground(Color.lightGray);
        dipinput.setFont(font);
        dipinput.setBackground(Color.lightGray);
        jinput.setFont(font);
        jinput.setBackground(Color.lightGray);
        incinput.setFont(font);
        incinput.setBackground(Color.lightGray);
        decinput.setFont(font);
        decinput.setBackground(Color.lightGray);
        plot.setFont(font);
        plot.setBackground(Color.green);

        c.weightx = 1.0;
        c.weighty = 1.0;
        c.gridwidth = 4;
        c.gridwidth=GridBagConstraints.REMAINDER;
        c.fill   =  GridBagConstraints.BOTH;

        gridbag.setConstraints(graph,c);

        c.fill   =  GridBagConstraints.NONE;
        c.weightx=0.0;
        c.weighty=0.0;
        c.gridheight=1;

        c.gridwidth=1;
        c.anchor = GridBagConstraints.EAST;
        gridbag.setConstraints(stationlabel,c);

        c.anchor = GridBagConstraints.CENTER;
        c.gridwidth=GridBagConstraints.RELATIVE;
        c.fill   =  GridBagConstraints.HORIZONTAL;
        gridbag.setConstraints(stationinput,c);
```

```
        c.fill = GridBagConstraints.NONE;
        c.gridwidth=GridBagConstraints.REMAINDER;

        gridbag.setConstraints(plot,c);


        c.gridwidth=1;
        c.anchor = GridBagConstraints.EAST;
        gridbag.setConstraints(samplelabel,c);
        c.gridwidth=2;
        c.anchor = GridBagConstraints.WEST;
        c.gridwidth=GridBagConstraints.REMAINDER;
        gridbag.setConstraints(sampleinput,c);

        c.gridwidth=1;
        c.anchor = GridBagConstraints.EAST;
        gridbag.setConstraints(depthlabel,c);
        c.gridwidth=2;
        c.anchor = GridBagConstraints.WEST;
        c.gridwidth=GridBagConstraints.REMAINDER;
        gridbag.setConstraints(depthinput,c);

        c.gridwidth=1;
        c.anchor = GridBagConstraints.EAST;
        gridbag.setConstraints(widthlabel,c);
        c.gridwidth=2;
        c.anchor = GridBagConstraints.WEST;
        c.gridwidth=GridBagConstraints.REMAINDER;
        gridbag.setConstraints(widthinput,c);

        c.gridwidth=1;
        c.anchor = GridBagConstraints.EAST;
        gridbag.setConstraints(diplabel,c);
        c.gridwidth=2;
        c.anchor = GridBagConstraints.WEST;
        c.gridwidth=GridBagConstraints.REMAINDER;
        gridbag.setConstraints(dipinput,c);

        c.gridwidth=1;
        c.anchor = GridBagConstraints.EAST;
        gridbag.setConstraints(jlabel,c);
        c.gridwidth=2;
        c.anchor = GridBagConstraints.WEST;
        c.gridwidth=GridBagConstraints.REMAINDER;
        gridbag.setConstraints(jinput,c);

        c.gridwidth=1;
        c.anchor = GridBagConstraints.EAST;
        gridbag.setConstraints(inclabel,c);
        c.gridwidth=2;
        c.anchor = GridBagConstraints.WEST;
        c.gridwidth=GridBagConstraints.REMAINDER;
        gridbag.setConstraints(incinput,c);

        c.gridwidth=1;
        c.anchor = GridBagConstraints.EAST;
        gridbag.setConstraints(declabel,c);
        c.gridwidth=2;
        c.anchor = GridBagConstraints.WEST;
        c.gridwidth=GridBagConstraints.REMAINDER;
        gridbag.setConstraints(decinput,c);


        panel.add(graph);
```

```
        panel.add(stationlabel);
        panel.add(stationinput);
        panel.add(plot);
        panel.add(samplelabel);
        panel.add(sampleinput);
        panel.add(depthlabel);
        panel.add(depthinput);
        panel.add(widthlabel);
        panel.add(widthinput);
        panel.add(diplabel);
        panel.add(dipinput);
        panel.add(jlabel);
        panel.add(jinput);
        panel.add(inclabel);
        panel.add(incinput);
        panel.add(declabel);
        panel.add(decinput);


        xaxis = graph.createXAxis();
        xaxis.setTitleText("Distance From Dike (m)");
       xaxis.setTitleFont(new Font("Helvetica",Font.PLAIN,18));
       xaxis.setLabelFont(new Font("Helvetica",Font.PLAIN,14));


        yaxis = graph.createYAxis();
        yaxis.setTitleText("nT");
        yaxis.setTitleFont(new Font("Helvetica",Font.PLAIN,18));
        yaxis.setLabelFont(new Font("Helvetica",Font.PLAIN,14));


        data = new DataSet();

        xaxis.attachDataSet(data);
        yaxis.attachDataSet(data);
        graph.attachDataSet(data);


        graph.setDataBackground(new Color(255,200,175));
        graph.setBackground(new Color(200,150,100));

        plot();

    }



    void plot() {
        int total_stations;
        double sample_interval;
        double ddepth, halfwidth, ddip, dintensity, maginc, magdec;
        boolean error = false;

        try {
             sample_interval = Double.valueOf(sampleinput.getText()).doubleValue();
        } catch(Exception e) {
             this.showStatus("Error with sample interval!");
             System.out.println("Sample interval error "+e.getMessage());
             return;
        }

        try {
             total_stations  = Integer.parseInt(stationinput.getText());
        } catch(Exception e) {
```

```
            this.showStatus("Error with number of stations!");
            System.out.println("number of stations  error "+e.getMessage());
            return;
        }

        try {
           halfwidth = (Double.valueOf(widthinput.getText()).doubleValue())/2.0;
        } catch(Exception e) {
            this.showStatus("Error with dike width  value!");
            System.out.println("dike width error "+e.getMessage());
            return;
        }

        try {
           ddepth = Double.valueOf(depthinput.getText()).doubleValue();
        } catch(Exception e) {
            this.showStatus("Error with depth value!");
            System.out.println("dike depth error "+e.getMessage());
            return;
        }
      try {
           ddip= Double.valueOf(dipinput.getText()).doubleValue();
        } catch(Exception e) {
            this.showStatus("Error with dip value!");
            System.out.println("dip error "+e.getMessage());
            return;
        }

      try {
           dintensity= 400.0*3.1415927*Double.valueOf(jinput.getText()).doubleValue();
        } catch(Exception e) {
            this.showStatus("Error with magnetization!");
            System.out.println("magnetization error "+e.getMessage());
            return;
        }

      try {
           maginc = Double.valueOf(incinput.getText()).doubleValue();
        } catch(Exception e) {
            this.showStatus("Error with inclination value!");
            System.out.println("inclination error "+e.getMessage());
            return;
        }

      try {
           magdec = Double.valueOf(decinput.getText()).doubleValue();
        } catch(Exception e) {
            this.showStatus("Error with declination value!");
            System.out.println("declination error "+e.getMessage());
            return;
        }




        this.showStatus("Calculating points!");


        int j, kount;

        double d[] = new double[2*total_stations];

        MagneticDike dike1 = new MagneticDike(maginc,magdec,ddepth,halfwidth,ddip,dintensity
+ );
```

```
        for (kount=j = 0; kount <= total_stations; kount++, j +=2) {
           try {
               d[j] = ((double)kount - (double)total_stations/2.0)*sample_interval;
               d[j+1] = dike1.anomaly2(d[j]);

           } catch (Exception e) {error = true;}

        }

        /**


         if(total_stations <= 2) {
             this.showStatus("Error NO POINTS to PLOT!");
             System.out.println("Error NO POINTS to PLOT!");
             return;
         } else
         if( error ) {
             this.showStatus("Error while Calculating points!");
             System.out.println("Error while calculating points!");
         }
        **/


         data.deleteData();

         try {
               data.append(d,total_stations);
         } catch(Exception e) {
             this.showStatus("Error while appending data!");
             System.out.println("Error while appending data!");
             return;
         }


         graph.repaint();
      }



    public boolean action(Event e, Object a) {

         if(e.target instanceof Button) {
             if( plot.equals(e.target) ) {
                  plot();
                  return true;
             }
         }


         return false;
       }



}
```

GEOLOGIC FACTORS CONTROLLING PATTERNS OF SMALL-VOLUME BASALTIC VOLCANISM: APPLICATION TO A VOLCANIC HAZARDS ASSESSMENT AT YUCCA MOUNTAIN, NEVADA

CHARLES B. CONNOR[1], JOHN A. STAMATAKOS[1], DAVID A. FERRILL[1], BRITTAIN E. HILL[1], GOODLUCK I. OFOEGBU[1], F. MICHAEL CONWAY[2], BUDHI SAGAR[1], and JOHN TRAPP[3]

1. Center for Nuclear Waste Regulatory Analyses, Southwest Research Institute, 6220 Culebra Rd, San Antonio, TX, 78238-5166, USA

2. Department of Physical Sciences, Western Arizona College, Yuma, AZ

3. U.S. Nuclear Regulatory Commission, Office of Materials Safety and Safeguards, Division of Waste Management, Washington, D.C.

submitted to

Journal of Geophysical Research

August, 1998

revision of Sept, 1999

Accepted

& in press

Information potentially subject to copyright protection was redacted from this location. The redacted material is from the information listed above.

Scientific Notebook
20-1402-461
Inititals: CC

Chuck Connor
Oct. 14,1999

The following plume tank experiments were performed by Daniel Alvarado on Jun15 - August 15, 1999

GENERAL EXPERIMENTAL PROCEDURE

Preparation of vials and rack:

Each vial in each rack will be identified with two numbers. The first number will be the rack number, and the second will be the vial number within that rack.

The racks will be aligned radially away from the vent in 8 different directions. Rack number 1 will be aligned straight towards the wall, and the others will be laid out clockwise from that line. For runs 1-3 included, the only rack present was held in the bottom by a wrench located on the opposite side from the rack, spanning from positions 6-8. For later runs, nothing was used to hold the racks in the bottom.

The vial closest to the vent will be number one, the next one number two, and so on. These numbers will be written on tape that will be adhered to the vials.

The distance between each vial and the vent will be measured to millimeter accuracy, with an error of plus or minus 1 mm.

Each vial will have the cap shaved off, so it does not interfere

with other vials' space.

Each vial, once tagged, will be weighed on an electronic scale, model MC1 Research RC 210 P and its mass will be recorded with an accuracy of one tenth of a milligram. Although the procedure described in the oven drying of samples document stated that the mass of the vials should be recorded to the milligram, it was thought that it would be better to record the tenths of milligrams for better accuracy.

Preparation of tank:

The tank will be filled with tap water via a tube or hose to a depth of 26.5 cm. The valve beneath the vent should be shut off to avoid leaks into the lower reservoir.

Commercial sugar or salt solution will be added to the water in order to increase its density. If salt is used, ****3 Kg will be added. If sugar is used, 5.43 Kg will be added. Either salt or sugar will be stirred until totally dissolved.

Preparation of lower reservoir:

Silicon glass beads or copper powder will be dumped in the flask. The quantities do not have to be accurate, but will range between 250-350 g of silicon beads or 7-12 g of copper powder. A magnetic stirring bar will be also dumped into the flask. Once tubing system is connected, faucet should be open almost to the max, allowing for

enough pressure to build up to drive the fluid. The flow will be controlled with the flowmeter knob. The rubber stopper on top of the flask will be loosened or the bleed off valve will be opened so air can escape the flask. The lower reservoir will be filled to the top as the contents are mixed with the magnetic stirrer operating at its maximum. Once the lower reservoir is full, the valve connecting the lower reservoir and the tank will be opened, allowing the plume to rise up to the tank.

Preparation of Samples:

Once the experiment has concluded, the vials will be capped when they are in the tank. The rack will be pulled off, and each vial will have most of its water siphoned out with a syringe.

The vials will be taken to the oven where they will be introduced (without the caps) for 1-3 hours until the water has evaporated. The temperature of the oven oscillates between 104-105 degrees Celcius. After heating, the dry vials will be introduced in a sealed container with anhydride crystals for about 1 hour until cooled down. The vials and their contents will be then weighed in the same electronic balance used before.

The data will be introduced into Kaleidagraph software where the mass of the sediment will be calculated for every vial. A graph will be plotted showing the mass of the sediment divided by the area of

each vial in relationship with the distance of each vial from the vent.

Calculations for area of base of vial:

Diameter of base of vial (inside diameter) = 0.625 cm

Area of base of vial = 3.1416 * (0.625 * 0.625) = 1.227 cm sq.

Calculations for Initial density of the fluid mixture in tank

SUGAR AS SOLUTE

Depth of mixture = 26.5 cm

Length * Width of base of the tank = 57.5 * 57.5 cm

Volume = 26.5 *57.5 *57.5 = 87615.625 cc

Mass of sugar = 5430 g

Depth of water = 25.7 cm

Volume of water = 25.7 *57.5 *57.5 = 84970.6 cc

Mass of water = Density (assumed 1 g/cc) * Volume = 84970.6 g

Total mass of mixture = 84790.6 + 5430 = 90400.6 g

Density of mixture = (90400.6 / 87615.625) = 1.03178 g/cc

Scientific Notebook
20-1402-461
Inititals: CC

Chuck Connor
Oct. 14,1999

FLOWMETER CALIBRATION *6/14/99 - 6/17/99*

Purpose: A flowmeter that will be used for our experiment was to be calibrated

Specifications of the flowmeter:

COLE PARMER, 150 mm Direct Reading, variable area for air and

water

Max. pressure it withstands: 200 psi.

Adjustable valve for flow control, reads "H"

Ser. No: 137793

Flow range measured: 2 mL/min to 90 mL/min

Experiment set-up:

A DI water container was used as a gravity feed to pump water through the flowmeter and into a 1000 mL pipette. Water from the container was let run though the tubing until a steady flow rate could be achieved and all air bubbles were driven out of the tube. Once this was achieved, the outlet was held into the pipette and a stopwatch was started. Times (for the water to fill the pipette) were recorded for flowrates (flowmeter readings) between 6 mL/min and 16 mL/min.

Data should be read as follows:

The number for flowmeter reading is the number of mL/min read by the uncalibrated flowmeter.

For the actual flow, each bracket represents one run or test. The first number in the bracket is the number of mL filled in the pipette. The second number is the number of seconds it took to fill such quantity.

For flowmeter readings of 6 mL/min and 4 mL/min, a Volume/time bracket was recorded every 250 mL (for flowmeter readings of 250 mL/min) or 500 mL (for flowmeter readings of 6 mL/min). Since the flow was very slow, this extra recordings give an idea of the evolution of the flow during a single run or test. The first number is the number of mL for each recording period, not the total. The second number represents the time it took for that volume to be filled. At the end of the four (or two) recordings, the totals (volume/time) are given, which were the only piece of data for that run to be used in the calculations for the calibration.

Notes about the experiment:

Some runs were better than others. In the bad ones, bubbles distorted the flow continuously and the meniscus jumped up and down too much. In some cases, it was obvious not only from experimental observation but also form results that a run was bad, in which case it was marked with an asterisk after the bracket and excluded from the calculations.

The flowmeter reading of 4 mL/min was achieved with valve ¾ of the way closed with H (height difference between source and outlet) of about 1 m. Minimum H enough for flow is about 0.5 m.

DATA

Flowmeter Reading (mL/s)	**Actual flow (mL/s)**
16	(1000/275)*(1000/252) (1000/250) (1000/253)
14	(1000/280) (1000/285) (1000/284) (1000/287) (1000/300) (1000/280)
12	(1000/355) (1000/397)*(1000/332) (1000/331) (1000/327) (1000/338)
10	(1000/455)*(1000/392) (1000/391) (1000/381) (1000/398) (1000/463)*(1000/403) (1000/394)
8	(1000/590)*(1000/525) (1000/531) (1000/484) (1000/488) (1000/480) (1000/503)
6	[(500/366) (500/364) = (1000/730)] [(500/340) (500/355) = (1000/695)]
	[(500/390) (500/388) = (1000/778)]*
	[(500/368) (500/355) = (1000/723)]
	[(500/345) (500/340) = (1000/685)]
	[(500/380) (500/635) = (1000/745)]
	[(500/673) (500/639) = (1000/712)]
4	[(250/270)(250/270)(250/255)(250/247) = (1000/1042)] [(250/313)(250/312)(250/318)(250/317) = (1000/1260)]
	[(250/265)(250/265)(250/263)(220/220) = (970/1013)]
	[(250/286)(250/272)(250/254)(250/243) = (1000/1055)]

[(250/270)(250/283)(250/317)(250/260) = (1000/1130)]

SAMPLE DRYING IN OVEN *06/16/99 - 06-22/99*

Purpose: This experiment was done to test whether the approach of drying samples in the oven to get rid of water and thus be able to weigh the sediment is appropriate and accurate enough for our experiment.

General Procedure:

For every run, the same basic procedure was used, and if any particular changes were done, these were noted for that run. The general procedure goes as follows:

Measured mass of container with particles in it (in some cases, the mass of the container itself was also noted)

Added DI water until container was almost filled, and waited for particles to settle (one to two minutes)

Siphoned water with a syringe until water level in container was about 2 or 3 mm over the sediment layer.

The samples were put in the oven at 105 C for a certain amount of time (depending on the run; see results), along with *Drierite* (anhydrous calcium sulfate, a drying agent that picks up moisture in

the oven.

The samples were extracted from oven after heating time and put in a sealed container to cool down for a certain time (depending on the run; see results). The sealed container also had some drying agent (*Drierite*) for the same purposes as in the oven.

The samples were dried and weighed after they had cooled down, and the mass of the sediment plus the container was compared before and after the experiment.

DATA

RUN 1 *06/16/99*

Vials	Mass V	Mass V+S	Mass V+S (dry)	Mass V+S (20 h. later)	Mass V+S (28 h. later)	Mass V+S (48 h. later)
plastic	1.272 g	1.306 g	1.306 g	1.304 g	1.302 g	1.301 g
Glass	6.409 g	6.445 g	6.409 g	6.407 g	6.407 g	6.407 g

*Mass of the glass vial with sediment 3 days later = 6.407 g

Heating time: 4 h.

Plastic vial was mishandled, which probably caused loss of mass

Scientific Notebook
20-1402-461
Inititals: CC

Chuck Connor
Oct. 14,1999

Plastic vials were uncapped during heating and cooling

Glass vials' tops were unscrewed but left on top during heating, and screwed back on during cooling

Plastic vials showed good results, but they were deformed due to the heat in the oven

RUN 2 *06/17/99*

Vials	Mass V+S	Mass V+S (repeat)	Mass V+S (dry)	Mass V+S (19 h. later)	Mass V+S (50 h. later)
1	6.4029 g	6.4031 g	6.532 g	6.532 g	6.353 g ??*
2	6.4462 g	6.4462 g	6.395 g	6.395 g	6.396 g
3	6.3787 g	6.3787 g	6.3265 g	6.327 g	6.327 g
4	6.3898 g	6.3898 g	6.337 g	6.338 g	6.3385 g
5	6.4226 g	6.4229 g	6.374 g	6.374 g	6.375 g

Heating time: 7 h.

All vials were glass, since plastic vials from run 1 were deformed

Capping procedures were the exact same as in run 1

From repeated weighing of V+S (Vial and sediment), it was determined that weight should be recorded only to 3 decimal digits, based on discrepancies of weight for same sample

Results for glass vials were not good enough. Plastic vials form run 1 gave good results, so these were used again in the next run to determine whether it was true that plastic vials always gave better results than glass vials.

*This number might have been misrecorded

RUN 3 *06/18/99 (morning)*

Vials	Mass V+S	Mass V+S (dry)	Mass V+S (4 h. later)
Plastic 1	1.307 g	1.037 g	1.308 g

Plastic 2	1.324 g	1.324 g	1.324 g
Glass 1	6.508 g	6.545* g	
Glass 2	6.498 g	6.481 g	6.482 g

*Glass vial 1 was still wet when pulled out of the oven. It was reheated with the samples of the next run during an hour, yielding a mass of 6.489 g.

Heating time: 1 h.

Filter paper approach was attempted in this run, but the samples blew away while being transported.

Plastic vials gave almost perfect results, while, glass vials gave errors that were too large. It was thought that maybe differences in capping during heating made the difference in results, so for the afternoon run, all samples were uncapped during heating, and only glass vial samples were capped during cooling.

RUN 4 *06/18/99 (afternoon)*

Scientific Notebook
20-1402-461
Inititals: CC

Chuck Connor
Oct. 14,1999

Vials	Mass		Mass		Mass	Mass V+S
Mass S	Mass V+S		Mass V+S			
	(w/o tag)	(w/ tag)			(dry)	(3 days later)
Plastic A	1.287 g	1.351 g	1.383 g	0.032 g	1.381 g	1.381 g
Plastic B	1.271 g	1.341 g	1.385 g	0.039 g	1.383 g	1.383 g
Plastic C	1.271 g	1.327 g	1.339 g	0.012 g	1.338 g	1.338 g
Glass A	6.426 g	6.472 g	6.536 g	0.064 g	6.525 g	6.5095 g
Glass B	6.334 g	6.383 g	6.410 g	0.027 g	6.412 g	6.396 g
Glass C	6.368 g	6.436 g	6.453 g	0.017 g	6.448 g	6.433 g

Heating time: 1 h.

Results for glass vials were better, indicating that uncapping the vials during heating may be the solution for inaccuracy. Nevertheless, the results are not accurate enough for our purposes. Also, plastic vials show better, but not perfect results. It was thought that uncapping the vials during cooling also would increase the accuracy for glass vials, which would be tested in the next run.

Scientific Notebook
20-1402-461
Inititals: CC

Chuck Connor
Oct. 14,1999

RUN 5 *06/21/99*

Vials	Mass vials		Mass V+S		Mass
S	Mass V+S		Mass V+S		
	(w/o tag)	(w/ tag)		(dry)	(20 h. after)
Glass X	6.417 g	6.352 g	6.372 g	0.0195 g	6.368 g	6.361 g
Glass Y	6.304 g	6.471 g	6.552 g	0.051 g	6.518 g	6.512 g
Glass Z	6.338 g	6.394 g	6.477 g	0.083 g	6.472 g	6.466 g
Plastic X	1.275 g	1.324 g	1.366 g	0.042 g	1.363 g	1.364 g
Plastic Y	1.264 g	1.322 g	1.3625 g	0.0405 g	1.360 g	1.360 g
Plastic Z	1.276 g	1.3495 g	1.464 g	0.1145 g	1.461 g	1.461 g

Heating time: 3 and ½ h.

It was concluded that the max error for plastic vials, which show better results than glass vials, was about 3 mg. This error in mass corresponds to an error of thickness in the plastic vials of about 20 micrometers (see calculations below), which is almost negligible. It is also thought that (from results) every time there is an error, that error seems to be almost the same for every sample for that run, which would allow (if this is true) us to add a constant number for the error of mass to every sample in a run in the experiment. For the

next run, samples with great quantity of sediment (such as those we will have close to the vent) are going to be tested in the oven to check whether there is any difference in behavior.

RUN 6 *06/22/99*

Vials	**Mass vials**		**Mass V+S**	**Mass S**	**Mass V+S**
	(w/o tag)	**(w/tag)**		**(dry)**	**(h. later)**
Plastic J	1.274 g	1.346 g	3.748 g	2.402 g	3.749 g
Plastic K	1.270 g	1.350 g	3.135 g	1.785 g	3.133 g
Glass J	6.308 g	6.351 g	9.1615 g	2.8105 g	9.1855 g
Glass K	6.2795 g	6.381 g	9.450 g	3.069 g	9.395* g

*Glass vial K had a water bubble in the middle of the sediment layer, and upon heating, the water shot up and the sediment above it was pushed out of the vial and dispersed around, accounting for the major loss of mass

Heating time: 1 h, 20 min

Plastic vials give good results, with a mass error in this run of 2 mg for a 1.785 g sediment sample. This represents an error of

0.112%. The glass vials give erroneous results, but in this run the errors were enlarged due to the loss of sediment of glass vial K (explained above) which apparently caused sediment to enter glass vial J (instead of slight mass loss, there was a mass gain of 16.5 mg)

CALCULATIONS FOR HEIGHT RESOLUTION FOR A MASS ERROR OF 3 mg

Density of particles = 1.53 mg/mm^3

Mass of particles = 3 mg

Volume of particles = Mass/density = 2 mm^3

Diameter of vial = 12 mm

Radius of vial = 6 mm

Area of the base of the vial = 3.14*36 = 113.0 mm^2

Volume = Area * Height

Height = Volume/Area = 2 (mm^3)/113 (mm^2) = *17.7 microns*

CONCLUSIONS:

The results show that plastic vials are appropriate for any sediment thickness. The maximum error in percentage for a plastic vial occurred in RUN 4, for plastic vial C, with a sediment mass of 12 mg and an error of 1 mg, giving a percentage error of 8.33%. The maximum

error in mass reported is of 3 mg, which occurred in two samples of RUN 5, yielding percentage error of no more than 7.1%. Although the mass loss is minimal, it is unpredictable. For this reason, it is suggested that two or more control vials with known sediment are introduced in the oven with the rest of the samples to give an idea of how much mass loss might be taking place in the rest of the samples.

RESULTS

Run 1:

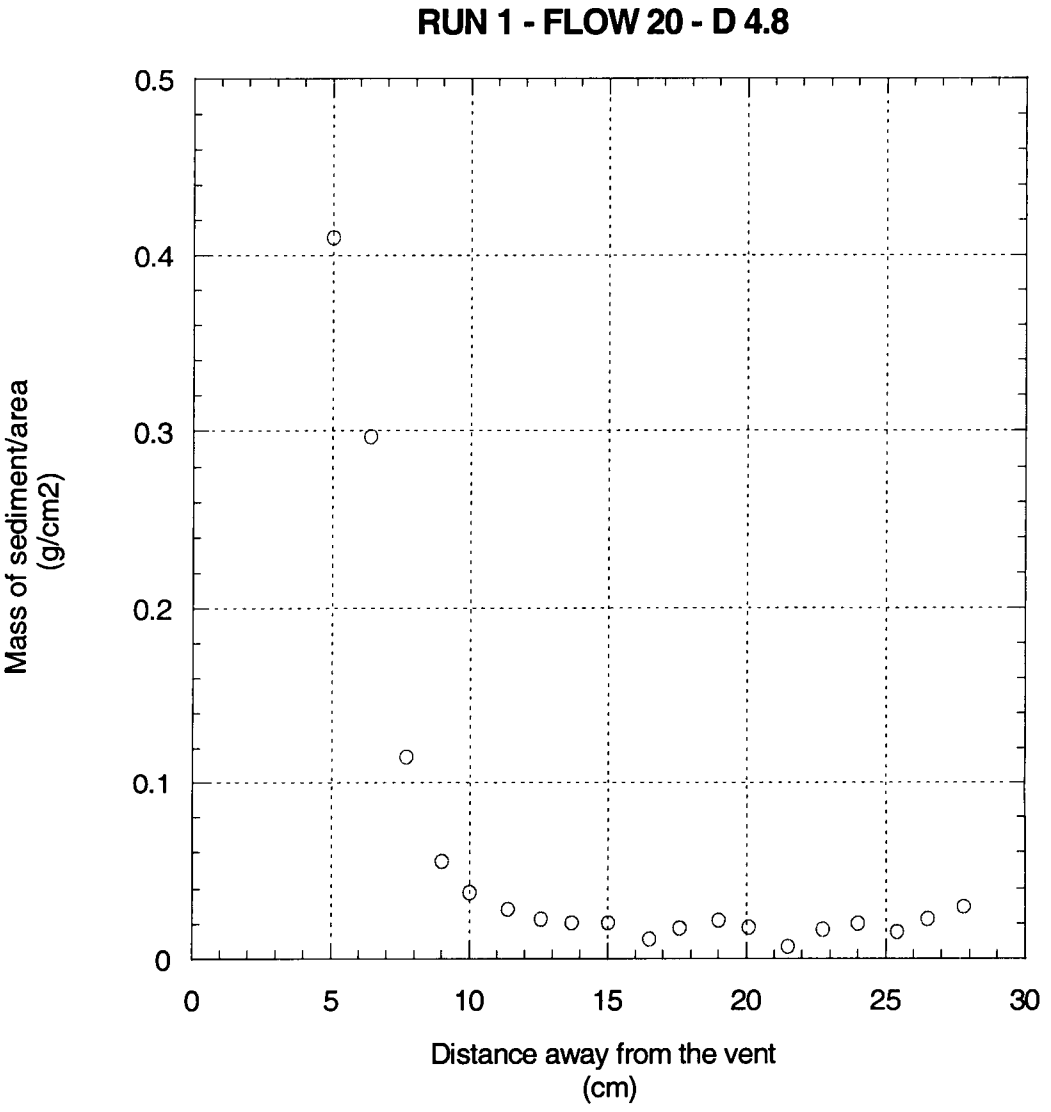
Format of the data is:

Vial # Dist (cm) Vial Mass (g) Mass vial + sediment (g) Mass of Sediment (g) Mass of sedimet (g) /Area of vial (cm2)

Data

1.0000	27.800	1.0023	1.0382	0.035900	0.029258
2.0000	26.500	1.0145	1.0423	0.027800	0.022657
3.0000	25.400	1.0111	1.0297	0.018600	0.015159

4.0000	24.000	1.0068	1.0310	0.024200	0.019723
5.0000	22.750	1.0190	1.0390	0.020010	0.016308
6.0000	21.500	1.0129	1.0211	0.0082000	0.0066830
7.0000	20.100	1.0116	1.0332	0.021600	0.017604
8.0000	19.000	1.0221	1.0485	0.026400	0.021516
9.0000	17.600	1.0194	1.0404	0.021000	0.017115
10.000	16.500	1.0103	1.0242	0.013900	0.011328
11.000	15.000	1.0108	1.0358	0.025000	0.020375
12.000	13.700	0.99820	1.0231	0.024900	0.020293
13.000	12.600	1.0161	1.0438	0.027700	0.022575
14.000	11.400	1.0188	1.0534	0.034600	0.028199
15.000	10.000	1.0127	1.0588	0.046100	0.037571
16.000	9.0000	1.0124	1.0798	0.067400	0.054931
17.000	7.7000	1.0233	1.1646	0.14130	0.11516
18.000	6.4000	1.0082	1.3725	0.36430	0.29690
19.000	5.0000	1.0100	1.5133	0.50330	0.41019


RUN 1 - FLOW 20 - D 4.8
Mass of sediment/area (g/cm2)
0.5
0.4
0.3
0.2
0.1
0
0
5
10
15
20
25
30
Distance away from the vent (cm)

Scientific Notebook
20-1402-461
Inititals: CC

Chuck Connor
Oct. 14,1999

Notes on Run 1

8/11/99

Sediment type: Silicon glass beads

Vent diameter: 4.8 mm

Flow rate: 20 mL/min (flowmeter reading) or 5 mL/min (real flow)

Initial depth of water: 26.5 cm

Solute type and amount: 5.43 Kg of sugar

Initial density of mixture in tank: 1.03178 g/cc

Run time: 10 minutes

Total number of racks and alignment: 1 rack at position 2

Total number of vials: 19

Heating time in oven: 2 hours

Cooling time in sealed case: 1 hour

The first two tries for this run failed as the faucet/tube connection snapped off because of excessive pressure. For both times, the water in the lower reservoir was drained and an estimate of silicon particles that were lost was added into the reservoir before restarting the experiment.

The rack was pulled out of the tank, and the vials were cleaned to eliminate remaining particles from the failed run. The rack was laid out on the bottom of the tank on the same spot where it was before, and its correct location was assured by measuring again some of the distances between vent and vial.

The sediment on the bottom was swept to a side away from the rack. The tank mixture was never drained

The third try worked correctly. When filling up the lower reservoir, the rubber stopper was loosened in order to release air from the flask. The plume grew clearly asymmetrically, trending towards line or position number 8 (line where rack number 8 would be laid out; see general experimental procedure). The plume height slowly increased from about 10 cm until hitting the surface of the water.

The outlet was opened a few minutes into the experiment. It was later closed when the water level dropped to 24.3 cm.

Run 2

Data

1.0000	5.0000	1.0174	1.2106	0.19320	0.15746
2.0000	6.1000	1.0133	1.1521	0.13880	0.11312

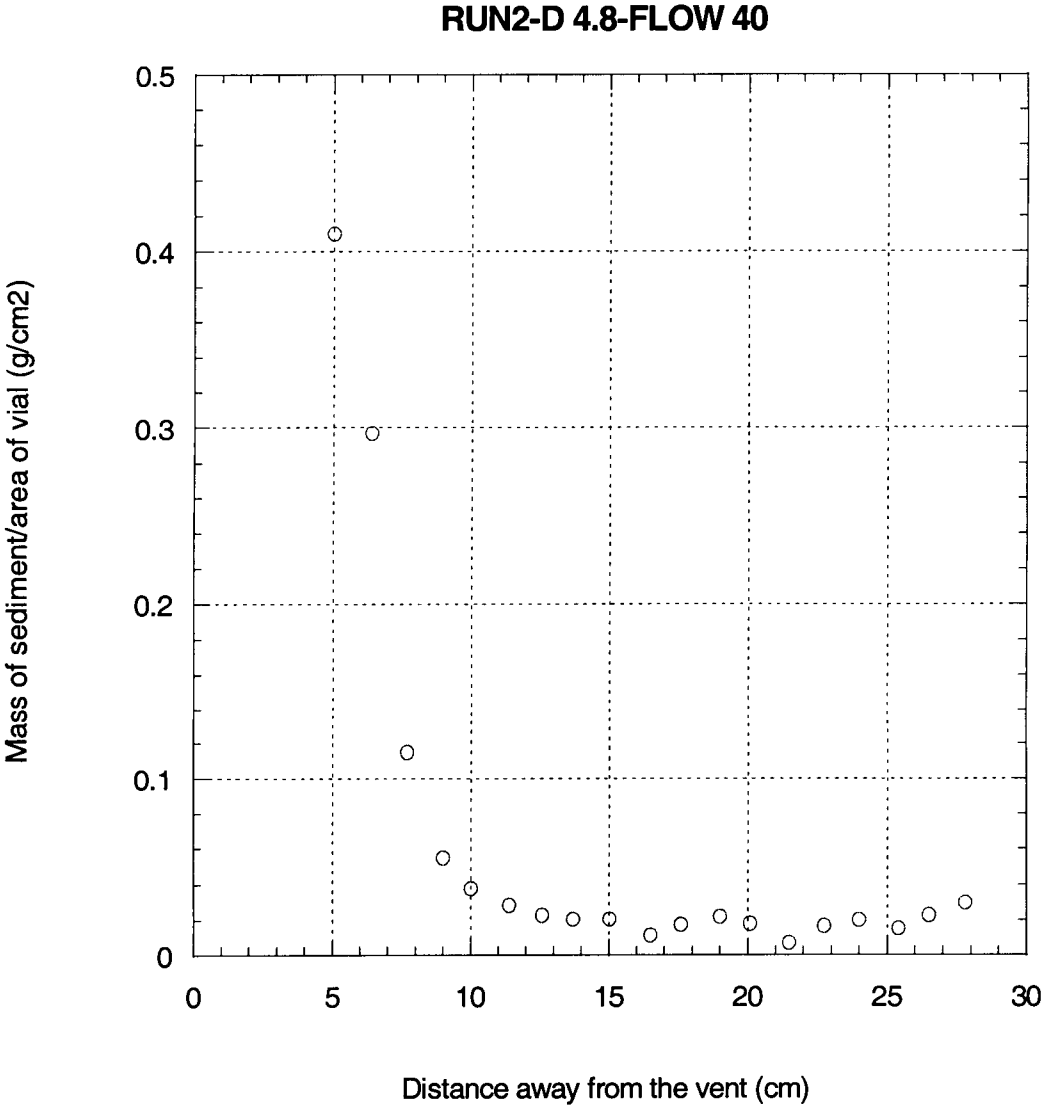
3.0000	7.2500	1.0219	1.1551	0.13320	0.10856
4.0000	8.5000	1.0075	1.0878	0.080300	0.065444
5.0000	10.000	1.0156	1.0878	0.072200	0.058843
6.0000	11.600	1.0107	1.0562	0.045500	0.037082
7.0000	12.300	1.0113	1.0484	0.037100	0.030236
8.0000	13.600	1.0259	1.0535	0.027600	0.022494
9.0000	15.000	1.0154	1.0377	0.022300	0.018174
10.000	16.200	1.0112	1.0293	0.018100	0.014751
11.000	17.500	1.0196	1.0409	0.021300	0.017359
12.000	18.750	1.0110	1.0234	0.012400	0.010106
13.000	20.000	1.0113	1.0176	0.0063001	0.0051345
14.000	21.300	1.0168	1.0345	0.017700	0.014425
15.000	22.600	1.0177	1.0288	0.011100	0.0090465
16.000	23.800	1.0196	1.0298	0.010200	0.0083130
17.000	25.000	1.0201	1.0417	0.021600	0.017604
18.000	26.400	1.0144	1.0425	0.028100	0.022901
19.000	27.500	1.0263	1.0357	0.0094000	0.0076610

Graph

Scientific Notebook
20-1402-461
Inititals: CC

Chuck Connor
Oct. 14,1999

Notes



8/12/99

Sediment type: Silicon glass beads

Vent diameter: 4.8 mm

Flow rate: 40 mL/min (flowmeter reading) or 10 mL/min (real flow)

Initial depth of water: 26.5 cm

Solute type and amount: 5.43 Kg of sugar

Initial density of mixture in tank: 1.03178 g/cc

Run time: 10 minutes

Total number of racks and alignment: 1 rack at position 2

Total number of vials: 19

Heating time in oven: 2 hours

Cooling time in sealed case: 1 hour

As in run 1, the air from the flask that was compressed as water was coming in was let out by loosening the rubber stopper. The plume grew asymmetrically again towards the same direction as in run 1. The plume quickly reached the surface of the water in the tank. The water level in the tank rose up to 27.8 cm before the outlet was activated. The flow was kept around 40* mL/min, with a high of 42* mL/min and a low of 36* mL/min.
Heating time was around 2 hours. Cooling time was around 1 hour.

During siphoning of water from vials, too much water was extracted from vial 13, and maybe others.

*These are flowmeter readings, which do not correspond to real flow rates; See flowmeter calibration.

Run 3

Data

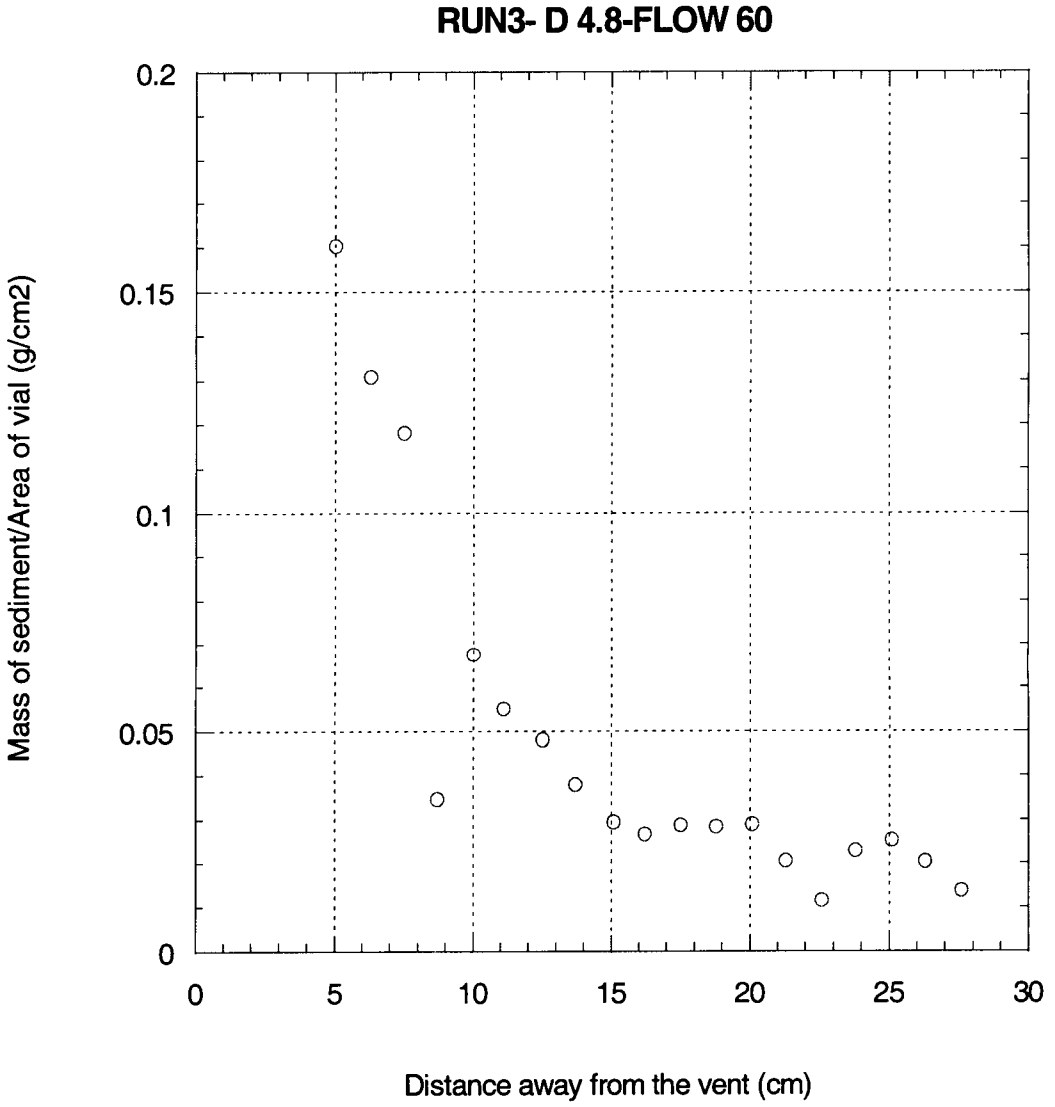
1.0000	5.0000	1.0100	1.2068	0.19680	0.16039
2.0000	6.3000	1.0095	1.1700	0.16050	0.13081
3.0000	7.5000	1.0153	1.1602	0.14490	0.11809
4.0000	8.7000	1.0146	1.0572	0.042600	0.034719
5.0000	10.000	1.0223	1.1054	0.083100	0.067726
6.0000	11.100	1.0154	1.0831	0.067700	0.055175
7.0000	12.500	1.0059	1.0649	0.059000	0.048085
8.0000	13.700	1.0116	1.0582	0.046600	0.037979
9.0000	15.100	1.0126	1.0488	0.036200	0.029503
10.000	16.200	1.0263	1.0590	0.032700	0.026650
11.000	17.500	1.0148	1.0501	0.035300	0.028769
12.000	18.800	1.0254	1.0603	0.034900	0.028443
13.000	20.100	1.0068	1.0422	0.035400	0.028851
14.000	21.300	1.0132	1.0385	0.025300	0.020619
15.000	22.600	0.99990	1.0142	0.014300	0.011654
16.000	23.800	1.0070	1.0351	0.028100	0.022901
17.000	25.100	1.0182	1.0493	0.031100	0.025346

18.000	26.300	1.0238	1.0489	0.025100	0.020456
19.000	27.600	1.0033	1.0204	0.017100	0.013937

Scientific Notebook
20-1402-461
Inititals: CC

Chuck Connor
Oct. 14,1999

Graph



Scientific Notebook
20-1402-461
Inititals: CC

Chuck Connor
Oct. 14,1999

Notes

8/13/99

Sediment type: Silicon glass beads

Vent diameter: 4.8 mm

Flow rate: 60 mL/min (flowmeter reading) or 15 mL/min (real flow)

Initial depth of water: 26.5 cm

Solute type and amount: 5.43 Kg of sugar

Initial density of mixture in tank: 1.03178 g/cc

Run time: 10 minutes

Total number of racks and alignment: 1 rack at position 2

Total number of vials: 19

Heating time in oven: 2 hours

Cooling time in sealed case: 1 hour

Since this run on, the air from the flask was let out by using the bleed off valve, after shortening the tube that goes into the flask. Since the tube only hangs about 1 or 2 cm from the rubber stopper, the bleed off valve can let air out while the flask is being filled up. Only when the flask is full will the bleed off valve let the mixture of sediment and water go out.
The outlet was activated 1 or 2 minutes into the experiment. The depth of the water in the tank never exceeded 28 cm. The flow was kept

between 58 and 62 mL/min (flowmeter readings).

The plume came out very strong from the very beginning, but it was asymmetrical again, this time towards line or position 6. The high flow caused the sediment in the lower reservoir to be emptied out of the reservoir quite quickly. It only lasted for 6 or 7 minutes. The other 3 or 4 minutes, the plume kept coming out but barely any sediment was visible.

Run 4

Data

Rack A

2.1000 5.0000 1.0080 1.2066 0.198600.16186

2.2000 6.0000 1.0202 1.1545 0.134300.10945

2.3000 7.3000 1.0180 1.1383 0.120300.098044

2.4000 8.4000 1.0092 1.1166 0.107400.087531

2.5000 9.7000 1.0137 1.0608 0.047100 0.038386

2.6000 11.100 1.0121 1.0473 0.035200 0.028688

2.7000 12.300 1.0038 1.0317 0.027900 0.022738

2.8000 13.500 1.0011 1.0305 0.029400 0.023961

2.9000 14.600 1.0137 1.0527 0.039000 0.031785

2.1000 15.900 1.0173 1.0401 0.022800 0.018582

2.1100 17.300 1.0075 1.0286 0.021100 0.017196

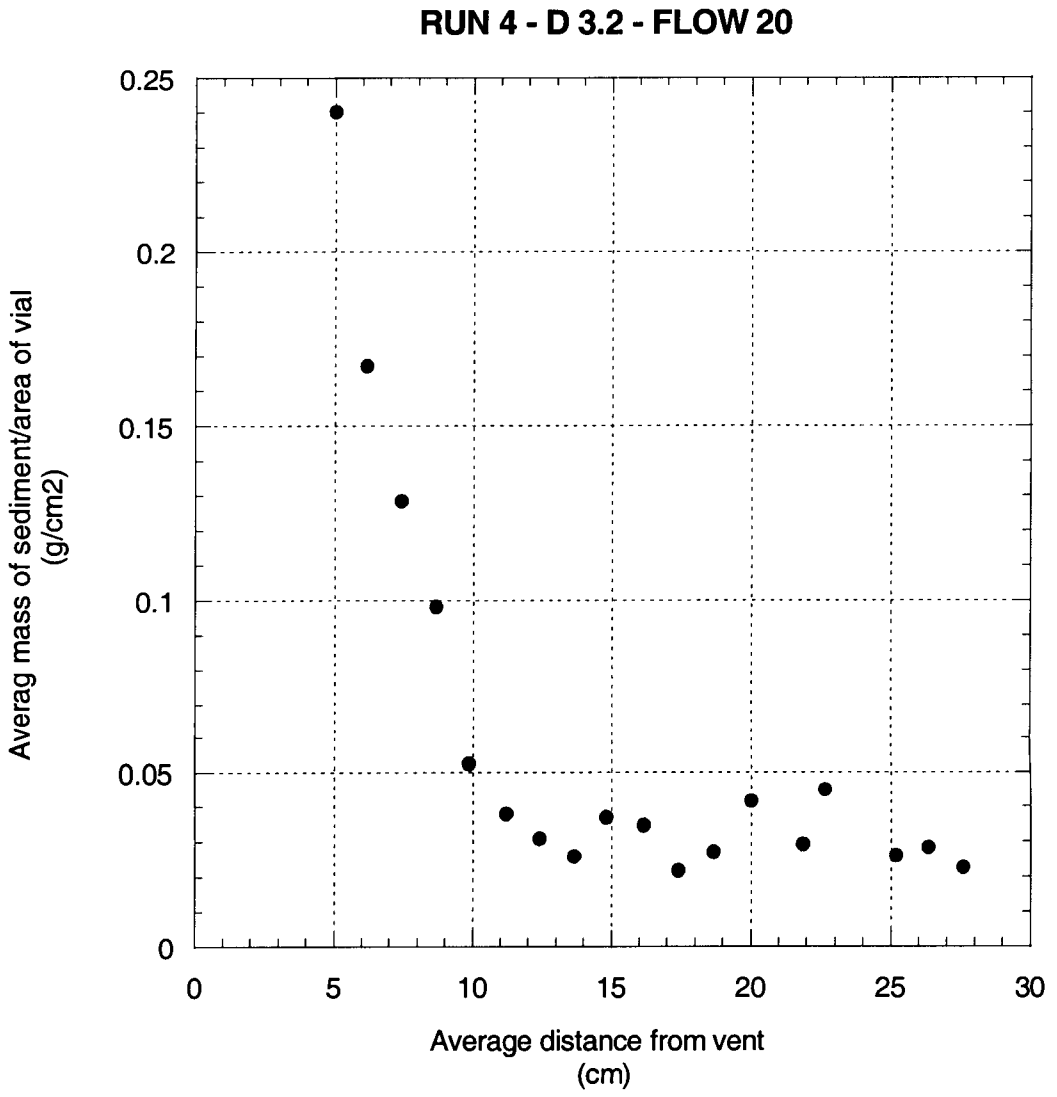
2.1200	18.500	1.0081	1.0505	0.042400	0.034556	
2.1300	19.900	0.999601	1.0337	0.034100	0.027791	
2.1400	22.200	1.0123	1.0476	0.035300	0.028769	
2.1500	22.500	1.0113	1.0457	0.034400	0.028036	
2.1600	23.800	1.0108	1.0428	0.032000	0.026080	
2.1700	25.000	1.0154	1.0419	0.026500	0.021597	
2.1800	26.300	1.0294	1.0617	0.032300	0.026324	
2.1900	27.500	1.0250	1.0470	0.022000	0.017930	

Rack B and Averages

6.1000	5.0000	1.0228	1.4135	0.39070	0.31842	5.0000	0.24014
6.2000	6.3000	1.0205	1.2965	0.27600	0.22494	6.1500	0.16720
6.3000	7.5000	1.0296	1.2243	0.19470	0.15868	7.4000	0.12836
6.4000	8.9000	1.0166	1.1504	0.13380	0.10905	8.6500	0.098289
6.5000	10.000	1.0086	1.0903	0.081700	0.066585	9.8500	0.052486
6.6000	11.300	1.0050	1.0627	0.057700	0.047025	11.200	0.037857
6.7000	12.500	1.0197	1.0680	0.048300	0.039364	12.400	0.031051
6.8000	13.800	1.0237	1.0575	0.033800	0.027547	13.650	0.025754
6.9000	15.000	1.0149	1.0661	0.051200	0.041728	14.800	0.036756
6.1000	16.400	1.0231	1.0853	0.062200	0.050693	16.150	0.034637
6.1100	17.500	0.997901	1.0300	0.032100	0.026161	17.400	0.021679
6.1200	18.800	1.0161	1.0399	0.023800	0.019397	18.650	0.026976

6.1300	20.100	1.0102	1.0783	0.068100	0.055501	20.000	0.041646
6.1400	21.500	1.0087	1.0449	0.036200	0.029503	21.850	0.029136
6.1500	22.800	1.0142	1.0904	0.076200	0.062103	22.650	0.045069
6.1600	24.000	1.0191	*	*	23.900		
6.1700	25.400	1.0231	1.0604	0.037300	0.030399	25.200	0.025998
6.1800	26.400	1.0144	1.0518	0.037400	0.030481	26.350	0.028403
6.1900	27.700	1.0129	1.0461	0.033200	0.027058	27.600	0.022494

*indicates that the vial had to be discarded


RUN 4 - D 3.2 - FLOW 20
Averag mass of sediment/area of vial (g/cm2)
0.25
0.2
0.15
0.1
0.05
0
0
5
10
15
20
25
30
Average distance from vent (cm)

Scientific Notebook
20-1402-461
Inititals: CC

Chuck Connor
Oct. 14,1999

Notes

Run4 *8/13/99*

Sediment type: Silicon glass beads

Vent diameter: 3.2 mm

Flow rate: 20 mL/min (flowmeter reading) or 5 mL/min (real flow)

Initial depth of water: 26.5 cm

Solute type and amount: 5.43 Kg of sugar

Initial density of mixture in tank: 1.03178 g/cc

Run time: 11 minutes

Total number of racks and alignment: 2 racks at positions 2 and 6

Total number of vials: 38

Heating time in oven: 2 hours

Cooling time in sealed case: 1 hour

The plume slowly grew in height until it reached the water surface after 4 or 5 minutes into the experiment. Also, the plume showed a clear asymmetry towards positions 4 or 5, which changed direction towards positions 9 or 10 9 minutes after the experiment began.
The flow was kept between 18 and 22 mL/min (flowmeter reading; see flowmeter calibration).
The outlet was started within 1 to 2 minutes into the experiment, and

it was stopped when the water depth got down to 26.2 cm.

Run 5

Data

Rack A

2.1000	5.0000	1.0144	1.3730	0.35860	0.29226
2.2000	6.3000	1.0104	1.3329	0.32250	0.26284
2.3000	7.6000	1.0109	1.2994	0.28850	0.23513
2.4000	8.9000	1.0001	1.2124	0.21230	0.17302
2.5000	10.100	1.0247	1.2190	0.19430	0.15835
2.6000	11.500	1.0171	1.1702	0.15310	0.12478
2.7000	12.600	1.0157	1.1426	0.12690	0.10342
2.8000	14.000	1.0057	1.1184	0.11270	0.091850
2.9000	15.300	1.0121	1.0903	0.078200	0.063733
2.1000	16.600	1.0147	1.0856	0.070900	0.057783
2.1100	17.800	1.0264	1.0804	0.054000	0.044010
2.1200	19.200	1.0132	1.0745	0.061300	0.049959
2.1300	20.400	1.0236	1.0638	0.040200	0.032763
2.1400	21.600	1.0036	1.0621	0.058500	0.047677
2.1500	22.800	1.0244	1.0584	0.034000	0.027710
2.1600	24.600	1.0076	1.0380	0.030400	0.024776
2.1700	25.400	1.0197	1.0511	0.031400	0.025591

2.1800	26.700	1.0250	1.0619	0.036900		0.030073		
2.1900	27.900	1.0181	1.0460	0.027900		0.022738		

Rack B and Averages

6.1000	5.0000	1.0180	1.2872	0.26920	0.21940	5.0000	0.25583	
6.2000	6.3000	1.0187	1.2476	0.22890	0.18655	6.3000	0.22469	
6.3000	7.7000	1.0146	1.2245	0.20990	0.17107	7.6500	0.20310	
6.4000	8.7000	1.0068	1.1661	0.15930	0.12983	8.8000	0.15143	
6.5000	10.100	1.0205	1.1600	0.13950	0.11369	10.100	0.13602	
6.6000	11.200	1.0007	1.1190	0.11830	0.096414	11.350	0.11059	
6.7000	12.500	1.0254	1.1333	0.10790	0.087938	12.550	0.095681	
6.8000	13.600	1.0090	1.0846	0.075600	0.061614	13.800	0.076732	
6.9000	15.200	1.0145	1.0790	0.064500	0.052567	15.250	0.058150	
6.1000	16.400	1.0182	1.0693	0.051100	0.041646	16.500	0.049715	
6.1100	17.500	1.0256	1.0733	0.047700	0.038875	17.650	0.041443	
6.1200	18.900	1.0118	1.0415	0.029700	0.024205	19.050	0.037082	
6.1300	20.300	1.0130	1.0502	0.037200	0.030318	20.350	0.031540	
6.1400	21.400	1.0097	1.0392	0.029500	0.024042	21.500	0.035860	
6.1500	22.700	1.0184	1.0485	0.030100	0.024531	22.750	0.026121	
6.1600	24.000	1.0279	1.0579	0.030000	0.024450	24.300	0.024613	
6.1700	25.700	1.0145	1.0443	0.029800	0.024287	25.550	0.024939	
6.1800	26.500	1.0108	1.0360	0.025200	0.020538	26.600	0.025306	
6.1900	27.600	1.0189	1.0474	0.028500	0.023227	27.750	0.022983	

Scientific Notebook
20-1402-461
Inititals: CC

Chuck Connor
Oct. 14,1999

Graph

6.1000 5.0000 1.0180 1.2872 0.269200.219405.0000 0.25583

6.2000 6.3000 1.0187 1.2476 0.228900.186556.3000 0.22469

6.3000 7.7000 1.0146 1.2245 0.209900.171077.6500 0.20310

6.4000 8.7000 1.0068 1.1661 0.159300.129838.8000 0.15143

6.5000 10.100 1.0205 1.1600 0.139500.1136910.100 0.13602

6.6000 11.200 1.0007 1.1190 0.118300.096414 11.350 0.11059

6.7000 12.500 1.0254 1.1333 0.107900.087938 12.550 0.095681

6.8000 13.600 1.0090 1.0846 0.075600 0.061614 13.800 0.076732

6.9000 15.200 1.0145 1.0790 0.064500 0.052567 15.250 0.058150

6.1000 16.400 1.0182 1.0693 0.051100 0.041646 16.500 0.049715

6.1100 17.500 1.0256 1.0733 0.047700 0.038875 17.650 0.041443

6.1200 18.900 1.0118 1.0415 0.029700 0.024205 19.050 0.037082

6.1300 20.300 1.0130 1.0502 0.037200 0.030318 20.350 0.031540

6.1400 21.400 1.0097 1.0392 0.029500 0.024042 21.500 0.035860

6.1500 22.700 1.0184 1.0485 0.030100 0.024531 22.750 0.026121

6.1600 24.000 1.0279 1.0579 0.030000 0.024450 24.300 0.024613

6.1700 25.700 1.0145 1.0443 0.029800 0.024287 25.550 0.024939

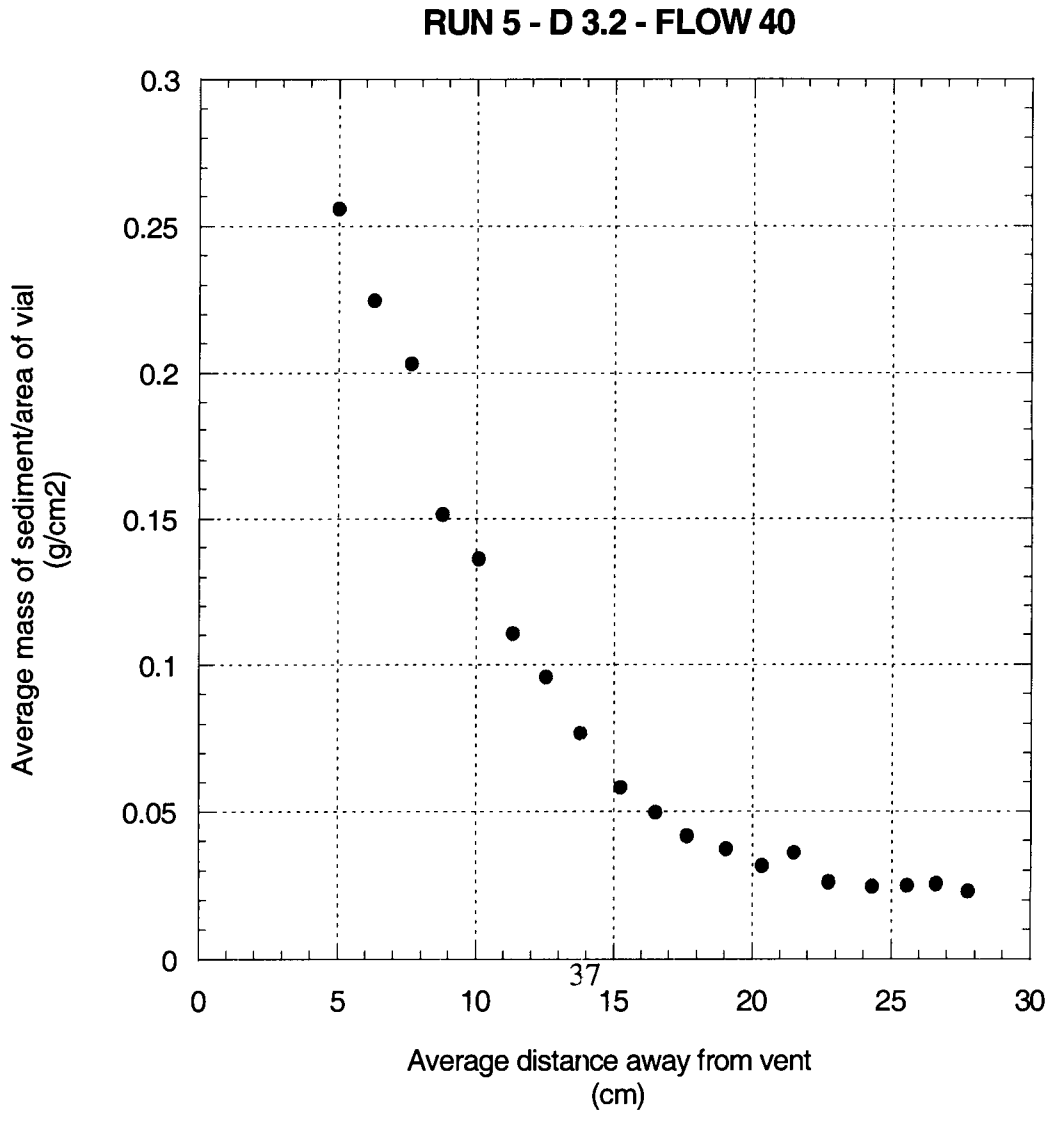
6.1800 26.500 1.0108 1.0360 0.025200 0.020538 26.600 0.025306

6.1900 27.600 1.0189 1.0474 0.028500 0.023227 27.750 0.022983

Scientific Notebook
20-1402-461
Inititals: CC

Chuck Connor
Oct. 14,1999

Graph



Scientific Notebook
20-1402-461
Inititals: CC

Chuck Connor
Oct. 14,1999

Notes

8/16/99

Sediment type: Silicon glass beads

Vent diameter: 3.2 mm

Flow rate: 40 mL/min (flowmeter reading) or 10 mL/min (real flow)

Initial depth of water: 26.5 cm

Solute type and amount: 5.43 Kg of sugar

Initial density of mixture in tank: 1.03178 g/cc

Run time: 10 minutes

Total number of racks and alignment: 2 racks at positions 2 and 6

Total number of vials: 38

Heating time in oven: 2 hours

Cooling time in sealed case: 15 hours

The first three tries failed for different reasons. In the first one, the faucet connection to the tube snapped off. In the second try, the magnetic stirrer lost optimal contact with the stirring bar and the sediment began to settle down in the bottom of the flask (lower reservoir). In the third try, the rubber stopper on top of the flask came off as the valves above it were switched too late. For all of these cases, the experiment was reset by draining the water from the

lower reservoir and adjusting the tube connecting it with the bottom of the tank.

The following was observed and/or done during the successful fourth run:

The plume came out quite strong and hit the water surface very quickly. The plume barely showed any asymmetry, only a small offset towards positions 7 or 8 (it was hard to determine). The flow was kept fairly constant between 40-42 mL/min (flowmeter reading; see flowmeter calibration for real flow). The outlet was started within 1 to 2 minutes into the experiment. The depth of the water before the outlet was started never exceeded 27 cm. The outlet was shut or stopped between 5 and 6 minutes into the experiment. The depth of the water had gone down to 26.3 cm.

Run 6

Data

Rack A

2.1000 5.0000 1.0159 1.2714 0.255500.20823

2.2000 6.3000 1.0148 1.2479 0.233100.18998

2.3000 7.4000 1.0106 1.1743 0.163700.13341

2.4000 8.8000 1.0100 1.1201 0.110100.089731

2.5000 10.000 1.0107 1.0836 0.072900 0.059413

2.6000 11.300 1.0210 1.0820 0.061000 0.049715

2.7000	12.600	1.0212	1.0535	0.032300	0.026324
2.8000	13.800	1.0108	1.0442	0.033400	0.027221
2.9000	15.000	1.0194	1.0416	0.022200	0.018093
2.1000	16.300	1.0119	1.0449	0.033000	0.026895
2.1100	17.600	1.0242	1.0522	0.028000	0.022820
2.1200	18.900	1.0105	1.0247	0.014200	0.011573
2.1300	20.100	1.0112	1.0329	0.021700	0.017685
2.1400	21.500	1.0134	1.0343	0.020900	0.017033
2.1500	22.700	1.0170	1.0294	0.012400	0.010106
2.1600	23.800	1.0160	1.0308	0.014800	0.012062
2.1700	25.200	0.999601	1.0250	0.025400	0.020701
2.1800	26.400	1.0223	1.0445	0.022200	0.018093
2.1900	27.800	1.0059	1.0311	0.025200	0.020538

Scientific Notebook
20-1402-461
Inititals: CC

Chuck Connor
Oct. 14,1999

Rack B and averages

6.1000 5.0000 1.0129 1.3619 0.349000.284435.0000 0.24633

6.2000 6.2000 1.0213 1.3301 0.308800.251676.2500 0.22082

6.3000 7.4000 1.0105 1.1916 0.181100.147607.4000 0.14051

6.4000 8.6000 1.0132 1.1912 0.178000.145078.7000 0.11740

6.5000 10.000 1.0195 1.1012 0.081700 0.066585 10.000 0.062999

6.6000 11.100 1.0016 1.0841 0.082500 0.067237 11.200 0.058476

6.7000 12.500 1.0248 1.0637 0.038900 0.031703 12.550 0.029014

6.8000 13.700 1.0127 1.0355 0.022800 0.018582 13.750 0.022901

6.9000 15.100 1.0060 1.0542 0.048200 0.039283 15.050 0.028688

6.1000 16.400 1.0043 1.0469 0.042600 0.034719 16.350 0.030807

6.1100 17.600 1.0047 1.0401 0.035400 0.028851 17.600 0.025835

6.1200 18.800 1.0136 1.0398 0.026200 0.021353 18.850 0.016463

6.1300 20.200 1.0137 1.0399 0.026200 0.021353 20.150 0.019519

6.1400 21.500 1.0108 1.0333 0.022500 0.018337 21.500 0.017685

6.1500 22.700 1.0174 1.0471 0.029700 0.024205 22.700 0.017156

6.1600 24.000 1.0083 1.0382 0.029900 0.024368 23.900 0.018215

6.1700 25.300 1.0112 1.0440 0.032800 0.026732 25.250 0.023716

6.1800 26.600 1.0156 1.0413 0.025700 0.020945 26.500 0.019519

6.1900 27.800 0.996301.0150 0.018700 0.015240 27.800 0.017889

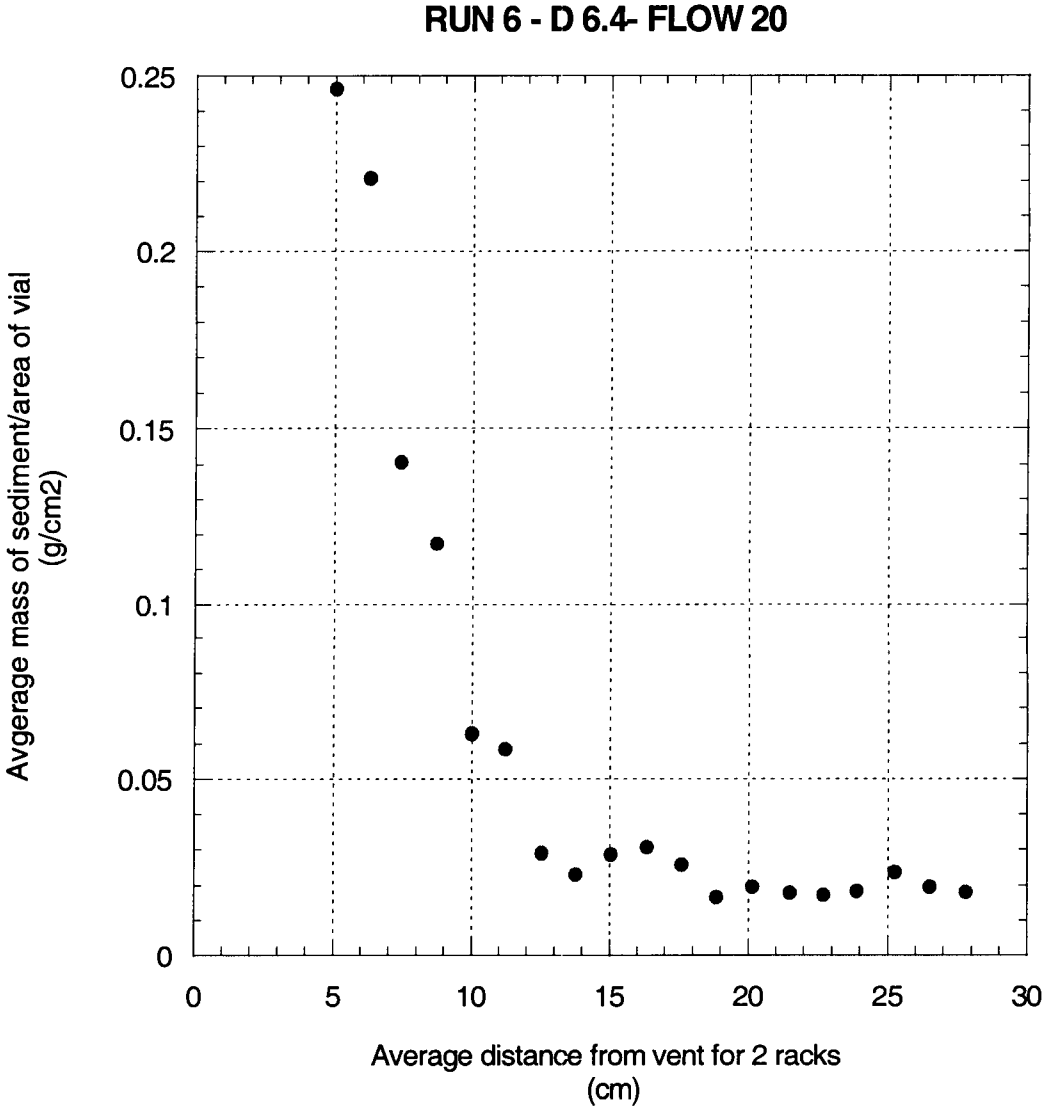
Graph

Scientific Notebook
20-1402-461
Inititals: CC

Chuck Connor
Oct. 14,1999

Notes

Run6 *8/17/99*



Scientific Notebook
20-1402-461
Inititals: CC

Chuck Connor
Oct. 14,1999

Sediment type: Silicon glass beads

Vent diameter: 6.4 mm

Flow rate: 20 mL/min (flowmeter reading) or 5 mL/min (real flow)

Initial depth of water: 26.5 cm

Solute type and amount: 5.43 Kg of sugar

Initial density of mixture in tank: 1.03178 g/cc

Run time: 11 minutes

Total number of racks and alignment: 2 racks at positions 2 and 6

Total number of vials: 38

Heating time in oven: 1.5 hours

Cooling time in sealed case: 15 hours

About 1 cm of mix (water and sugar) depth was lost before the experiment started. The water tank was filled with water to the original level, but no sugar was added. Thus, the initial density decreased slightly.

The tube (vent) came out above the teflon rod about 4 to 5 millimeters.

At the beginning of the experiment, many bubbles were observed coming out of the vent. At the same time, the plume would not reach higher than 10-12 cm and it collapsed. Most of the sedimentation occurred very close to the vent. The plume also showed some asymmetry

towards positions 7 or 8. The bubbles became less numerous but continued to appear during most of the experiment. Slowly, the plume began to increase its height, and about 7 or 8 minutes into the experiment, the plume reached the top and seemed to gain symmetry. The sedimentation during this period seemed to be more spread out since the plume was higher and it did not collapse like at the beginning.

The outlet was on since minutes 4 or 5 until the experiment was over. The water level never went over 27 cm, and it went down as low as 26 cm.

Vials 2 and 3 to 6 for rack 2 seemed a little tilted to a side during the experiment. The same was noted for vials 16 to 19 in rack 6.

Run 7

Data

Rack A

2.1000 5.0000 1.0547 1.4409 0.386200.31475

2.2000 6.3000 1.0436 1.2217 0.178100.14515

2.3000 7.5000 1.0463 1.2225 0.176200.14360

2.4000 8.8000 1.0374 1.1771 0.139700.11385

2.5000 10.000 1.0433 1.1645 0.121200.098777

2.6000 11.400 1.0459 1.1553 0.109400.089161

2.7000 12.700 1.0462 1.1467 0.100500.081907

2.8000	13.900	1.0462	1.1260	0.079800	0.065037
2.9000	15.100	1.0580	1.1437	0.085700	0.069845
2.1000	16.400	1.0388	1.1177	0.078900	0.064303
2.1100	17.800	1.0476	1.1265	0.078900	0.064303
2.1200	18.900	1.0502	1.1115	0.061300	0.049959
2.1300	20.300	1.0411	1.1035	0.062400	0.050856
2.1400	21.500	1.0541	1.1221	0.068000	0.055420
2.1500	22.800	1.0518	1.1127	0.060900	0.049633
2.1600	24.000	1.0375	1.1002	0.062700	0.051100
2.1700	25.300	1.0470	1.1081	0.061100	0.049796
2.1800	26.600	1.0418	1.0973	0.055500	0.045232
2.1900	27.900	1.0441	1.1094	0.065300	0.053219

Rack B

4.1000	5.0000	1.0399	1.2730	0.233100	0.18998
4.2000	6.3000	1.0533	1.2412	0.187900	0.15314
4.3000	7.5000	1.0453	1.1898	0.144490	0.11776
4.4000	8.8000	1.0562	1.2016	0.145400	0.11850
4.5000	10.000	1.0391	1.1552	0.116100	0.094621

4.6000	11.400	1.0471	1.1494	0.102300.083374	
4.7000	12.700	1.0555	1.1541	0.098600	0.080359
4.8000	13.900	1.0109	1.1005	0.089600	0.073024
4.9000	15.100	1.0406	1.1254	0.084800	0.069112
4.1000	16.400	1.0502	1.1280	0.077800	0.063407
4.1100	17.800	1.0448	1.1197	0.074900	0.061043
4.1200	18.900	1.0519	1.1253	0.073400	0.059821
4.1300	20.300	1.0420	1.1132	0.071200	0.058028
4.1400	21.500	1.0437	1.1140	0.070300	0.057294
4.1500	22.800	1.0544	1.1222	0.067800	0.055257
4.1600	24.000	1.0467	1.1191	0.072400	0.059006
4.1700	25.300	1.0408	1.1063	0.065500	0.053382
4.1800	26.600	1.0438	1.1089	0.065100	0.053056
4.1900	27.900	1.0438	1.1096	0.065800	0.053627
Rack C					
6.1000	5.0000	1.0508	1.2517	0.200900.16373	
6.2000	6.3000	1.0458	1.2137	0.167900.13684	
6.3000	7.5000	1.0568	1.1896	0.132800.10823	
6.4000	8.8000	1.0471	1.1570	0.109900.089568	
6.5000	10.000	1.0400	1.1399	0.099900	0.081418
6.6000	11.400	1.0552	1.1450	0.089800	0.073187

6.7000	12.700	1.0558	1.1335	0.077700	0.063325	
6.8000	13.900	1.0433	1.1240	0.080700	0.065770	
6.9000	15.100	1.0416	1.1106	0.069000	0.056235	
6.1000	16.400	1.0446	1.1201	0.075500	0.061532	
6.1100	17.800	1.0446	1.1172	0.072600	0.059169	
6.1200	18.900	1.0481	1.1123	0.064200	0.052323	
6.1300	20.300	1.0469	1.1151	0.068200	0.055583	
6.1400	21.500	1.0387	1.1169	0.078200	0.063733	
6.1500	22.800	1.0441	1.1130	0.068900	0.056153	
6.1600	24.000	1.0423	1.1050	0.062700	0.051100	
6.1700	25.300	1.0574	1.1140	0.056600	0.046129	
6.1800	26.600	1.0427	1.1035	0.060800	0.049552	
6.1900	27.900	1.0441	1.1050	0.060900	0.049633	

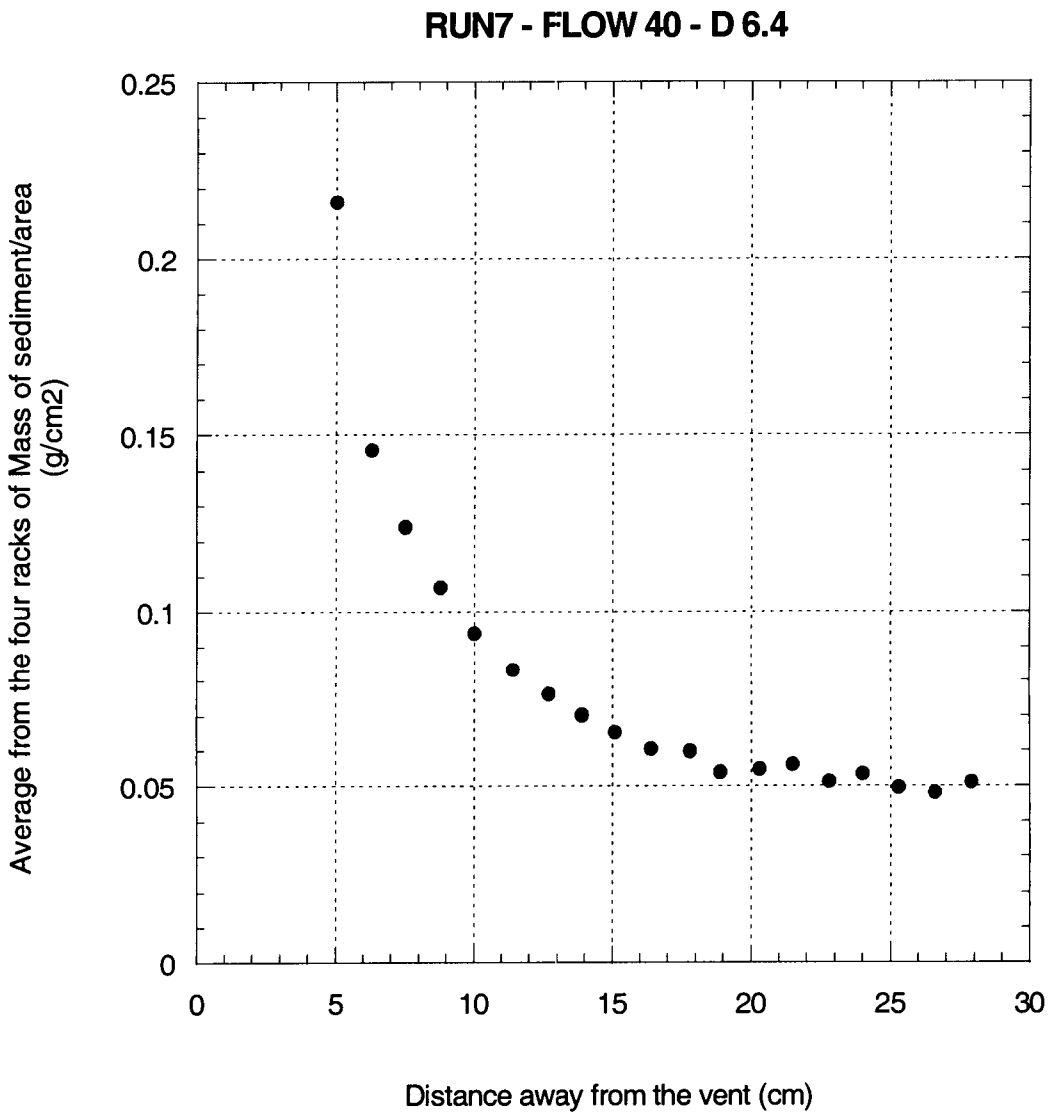
Rack D and mass Average

8.1000	5.0000	1.0114	1.2522	0.240800	0.196250	0.21618
8.2000	6.3000	1.0072	1.1891	0.181900	0.148250	0.14584
8.3000	7.5000	1.0118	1.1676	0.155800	0.126980	0.12414
8.4000	8.8000	1.0107	1.1411	0.130400	0.106280	0.10705
8.5000	10.000	1.0078	1.1318	0.124000	0.101060	0.093969
8.6000	11.400	1.0138	1.1210	0.107200	0.087368	0.083272
8.7000	12.700	1.0109	1.1073	0.096400	0.078566	0.076039

8.8000	13.900	1.0103	1.1051	0.094800	0.077262	0.070273
8.9000	15.100	1.0285	1.1095	0.081000	0.066015	0.065302
8.1000	16.400	1.0201	1.0844	0.064300	0.052404	0.060412
8.1100	17.800	1.0146	1.0825	0.067900	0.055338	0.059963
8.1200	18.900	1.0221	1.0876	0.065500	0.053382	0.053871
8.1300	20.300	1.0106	1.0778	0.067200	0.054768	0.054808
8.1400	21.500	1.0166	1.0758	0.059200	0.048248	0.056174
8.1500	22.800	1.0121	1.0669	0.054800	0.044662	0.051426
8.1600	24.000	1.0134	1.0783	0.064900	0.052893	0.053525
8.1700	25.300	1.0224	1.0832	0.060800	0.049552	0.049715
8.1800	26.600	1.0196	1.0747	0.055100	0.044906	0.048187
8.1900	27.900	1.0099	1.0694	0.059500	0.048492	0.051243

Graph

Notes


RUN7 - FLOW 40 - D 6.4
Average from the four racks of Mass of sediment/area (g/cm2)
0.25
0.2
0.15
0.1
0.05
0
0
5
10
15
20
25
30
Distance away from the vent (cm)

Scientific Notebook
20-1402-461
Inititals: CC

Chuck Connor
Oct. 14,1999

Run7 *8/18/99*

Sediment type: Silicon glass beads

Vent diameter: 6.4 mm

Flow rate: 40 mL/min (flowmeter reading) or 10 mL/min (real flow)

Initial depth of water: 26.5 cm

Solute type and amount: 5.43 Kg of sugar

Initial density of mixture in tank: 1.03178 g/cc

Run time: 10 minutes

Total number of racks and alignment: 4 standard short racks at positions 2, 4, 6 and 8

Total number of vials: 76

Heating time in oven: 4 hours

Cooling time in sealed case: 1 hour

The tube that mimics the vent was cut irregularly, causing it to have jagged edges on the rim and inside the tube itself.
Some bubbles were observed during most of the experiment, especially during the beginning. The flow was unsteady and jumpy, but it was kept between 37 and 43 mL/min (flowmeter reading; see flowmeter calibration).

At the beginning of the experiment, the plume came out fairly weak for such flow. A few seconds after the beginning, the plume

collapsed. After its collapse, the plume began a steady climb until it reached the surface, which happened at about 3 minutes after the beginning. The plume still behaved like it was heavy, with sediment falling out of the plume from the top to near the base of it. At 4 or 5 minutes into the experiment, the plume changed its shape and behavior towards a more powerful, fountain looking one. Most of the sediments now fell out of the plume only from its top. In this run, the plume showed great symmetry, unlike the ones before.

Most of the sediment contained in the lower reservoir was gone at about 7 minutes into the experiment.

The outlet was started 4 minutes into the experiment, when the level of the mix in the tank was about 27.3 cm. This outlet was left on for the rest of the experiment. The lowest level of the mix in the tank was 25.5 cm.

Vials 2.2 and 8.2 were toppled over during after the experiment.

Run 8

Data

Rack A

2.1000 5.0000 1.0176 1.2240 0.20640	0.16822
2.2000 6.3000 1.0193 1.1674 0.14810	0.12070
2.3000 7.5000 1.0230 1.1460 0.12300	0.10024

2.4000 8.8000 1.0142 1.1219 0.10770 0.087775

2.5000 10.000 1.0217 1.1173 0.095600 0.077914

2.6000 11.400 1.0185 1.1098 0.091300 0.074409

2.7000 12.700 1.0153 1.1004 0.085100 0.069356

2.8000 13.900 1.0139 1.0933 0.079400 0.064711

2.9000 15.100 1.0116 1.0880 0.076400 0.062266

2.1000 16.400 1.0195 1.0926 0.073100 0.059576

2.1100 17.800 1.0123 1.0635 0.051200 0.041728

2.1200 18.900 1.0095 1.0725 0.063000 0.051345

2.1300 20.300 1.0143 1.0715 0.057200 0.046618

2.1400 21.500 1.0144 1.0632 0.048800 0.039772

2.1500 22.800 1.0092 1.0562 0.047000 0.038305

2.1600 24.000 1.0058 1.0583 0.052500 0.042787

2.1700 25.300 1.0138 1.0641 0.050300 0.040994

2.1800 26.600 1.0193 1.0745 0.055200 0.044988

2.1900 27.900 1.0097 1.0707 0.061000 0.049715

Rack B

4.1000 5.0000 1.0134 1.2514 0.23800 0.19397

4.2000 6.3000 1.0177 1.1736 0.15590 0.12706

4.3000 7.5000 1.0199 1.1515 0.13160 0.10725

4.4000 8.8000 1.0126 1.1285 0.11590 0.094458

4.5000 10.000 1.0150 1.1040 0.089000 0.072535

4.6000 11.400 1.0235 1.1054 0.081900 0.066748

4.7000 12.700 1.0072 1.0981 0.090900 0.074083

4.8000 13.900 1.0150 1.0718 0.056800 0.046292

4.9000 15.100 1.0106 1.0798 0.069200 0.056398

4.1000 16.400 1.0141 1.0707 0.056600 0.046129

4.1100 17.800 1.0171 1.0870 0.069900 0.056968

4.1200 18.900 1.0215 1.0876 0.066100 0.053871

4.1300 20.300 1.0192 1.0870 0.067800 0.055257

4.1400 21.500 1.0120 1.0670 0.055000 0.044825

4.1500 22.800 1.0086 1.0717 0.063100 0.051426

4.1600 24.000 1.0086 1.0555 0.046900 0.038223

4.1700 25.300 1.0173 1.0828 0.065500 0.053382

4.1800 26.600 1.0165 1.0442 0.027700 0.022575

4.1900 27.900 1.0127 1.0697 0.057000 0.046455

Rack C

6.1000 5.0000 1.0089 1.5693 0.56040 0.45672

6.2000	6.3000	1.0138	1.3142	0.30040	0.24482
6.3000	7.5000	1.0232	1.1915	0.16830	0.13716
6.4000	8.8000	1.0093	1.1011	0.091800	0.074817
6.5000	10.000	1.0193	1.0886	0.069300	0.056479
6.6000	11.400	1.0104	1.0623	0.051900	0.042298
6.7000	12.700	1.0133	1.0615	0.048200	0.039283
6.8000	13.900	1.0136	1.0604	0.046800	0.038142
6.9000	15.100	1.0182	1.0630	0.044800	0.036512
6.1000	16.400	1.0120	1.0561	0.044100	0.035941
6.1100	17.800	1.0061	1.0529	0.046800	0.038142
6.1200	18.900	1.0119	1.0553	0.043400	0.035371
6.1300	20.300	1.0143	1.0510	0.036700	0.029910
6.1400	21.500	1.0113	1.0572	0.045900	0.037408
6.1500	22.800	1.0247	1.0608	0.036100	0.029421
6.1600	24.000	1.0255	1.0431	0.017600	0.014344
6.1700	25.300	1.0131	1.0349	0.021800	0.017767
6.1800	26.600	1.0173	1.0520	0.034700	0.028280
6.1900	27.900	1.0046	1.0336	0.029000	0.023635

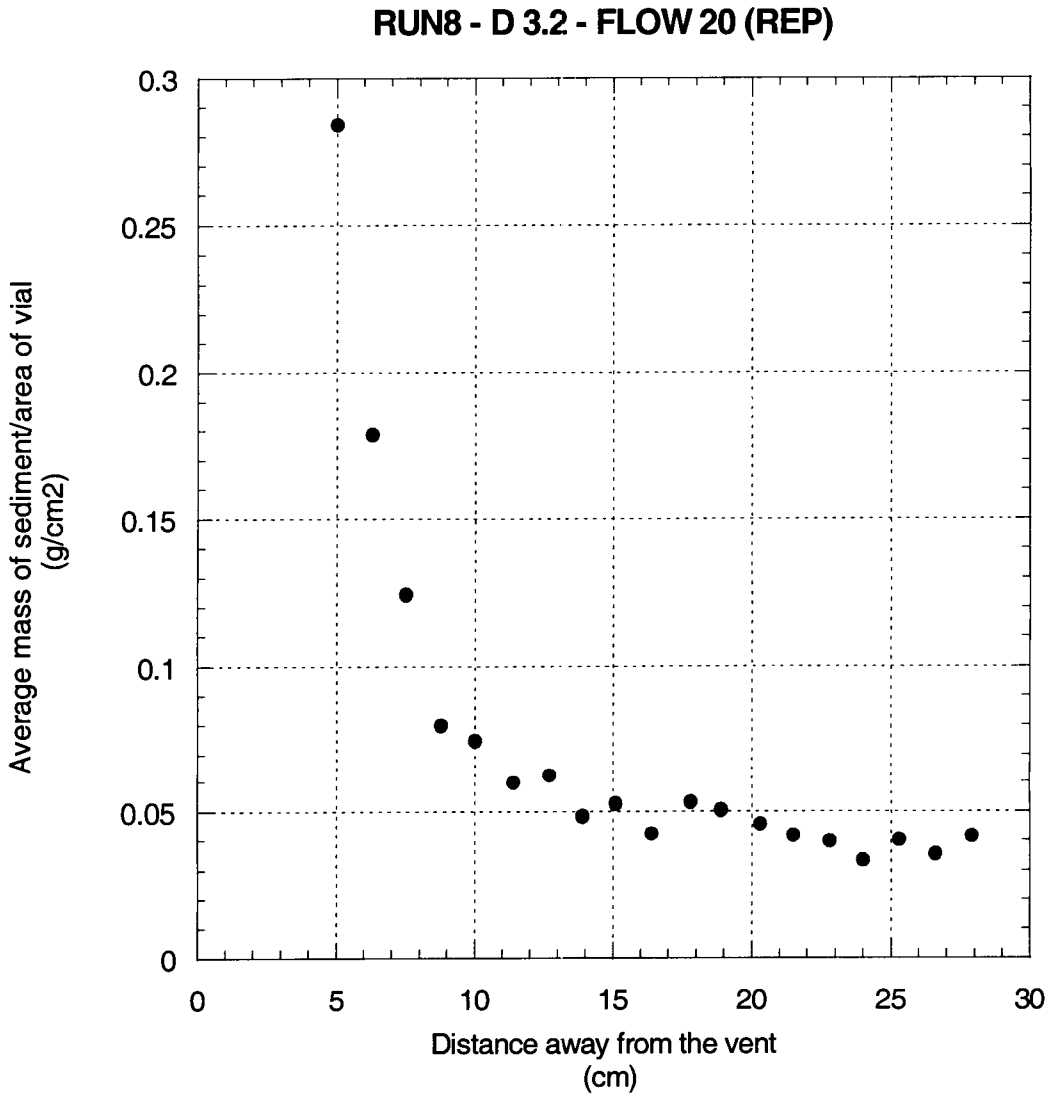
Rack D and mass Average

8.1000	5.0000	1.0167	1.4059	0.38920	0.31720	0.28403
8.2000	6.3000	1.0064	1.2800	0.27360	0.22298	0.17889
8.3000	7.5000	1.0136	1.2014	0.18780	0.15306	0.12443
8.4000	8.8000	1.0138	1.0910	0.077200	0.062918	0.079992
8.5000	10.000	1.0200	1.1330	0.11300	0.092095	0.074756
8.6000	11.400	1.0161	1.0841	0.068000	0.055420	0.059719
8.7000	12.700	1.0049	1.0864	0.081500	0.066422	0.062286
8.8000	13.900	1.0206	1.0752	0.054600	0.044499	0.048411
8.9000	15.100	1.0163	1.0842	0.067900	0.055338	0.052628
8.1000	16.400	1.0129	1.0469	0.034000	0.027710	0.042339
8.1100	17.800	1.0207	1.1129	0.092200	0.075143	0.052995
8.1200	18.900	1.0213	1.0960	0.074700	0.060880	0.050367
8.1300	20.300	1.0143	1.0773	0.063000	0.051345	0.045782
8.1400	21.500	1.0097	1.0657	0.056000	0.045640	0.041911
8.1500	22.800	1.0124	1.0618	0.049400	0.040261	0.039853
8.1600	24.000	1.0156	1.0613	0.045700	0.037245	0.033150
8.1700	25.300	1.0031	1.0628	0.059700	0.048655	0.040200
8.1800	26.600	1.0135	1.0692	0.055700	0.045395	0.035310
8.1900	27.900	1.0161	1.0728	0.056700	0.046210	0.041504

Scientific Notebook
20-1402-461
Inititals: CC

Chuck Connor
Oct. 14,1999

Graph


RUN8 - D 3.2 - FLOW 20 (REP)
Average mass of sediment/area of vial (g/cm2)
0.3
0.25
0.2
0.15
0.1
0.05
0
0
5
10
15
20
25
30
Distance away from the vent (cm)

Scientific Notebook
20-1402-461
Inititals: CC

Chuck Connor
Oct. 14,1999

Notes

Run8 *8/19/99*

Sediment type: Silicon glass beads

Vent diameter: 3.2 mm

Flow rate: 20 mL/min (flowmeter reading) or 5 mL/min (real flow)

Initial depth of water: 26.5 cm

Solute type and amount: 5.43 Kg of sugar

Initial density of mixture in tank: 1.03178 g/cc

Run time: 10 minutes

Total number of racks and alignment: 4 standard short racks at positions 2, 4, 6 and 8

Total number of vials: 76

Heating time in oven: 3.5 hours at 110 degrees C

Cooling time in sealed case: 1 hour

A few seconds after the eruption, the plume collapses symmetrically. This collapse was followed by a slow increase in height of the plume hindered by several smaller collapses. Finally, the plume grew steady unhindered until it reached the top about 7 minutes after the beginning. This plume, although reaching the top, seemed to be

quite dense, since particles fell out of the plume starting at a height of one third that of the plume itself.

Although the flow was kept quite stable, many bubbles came into the tank, visibly affecting the plume by pushing it upwards. These bubbles probably gave the push necessary to help the plume reach the top.

The deposition from the plume was quite if not very symmetrical. The outlet was never opened and the maximum water level was 27.3 cm.

The tube (vent) was stuck about 0.5 to 1 cm below the surface of the teflon rod.

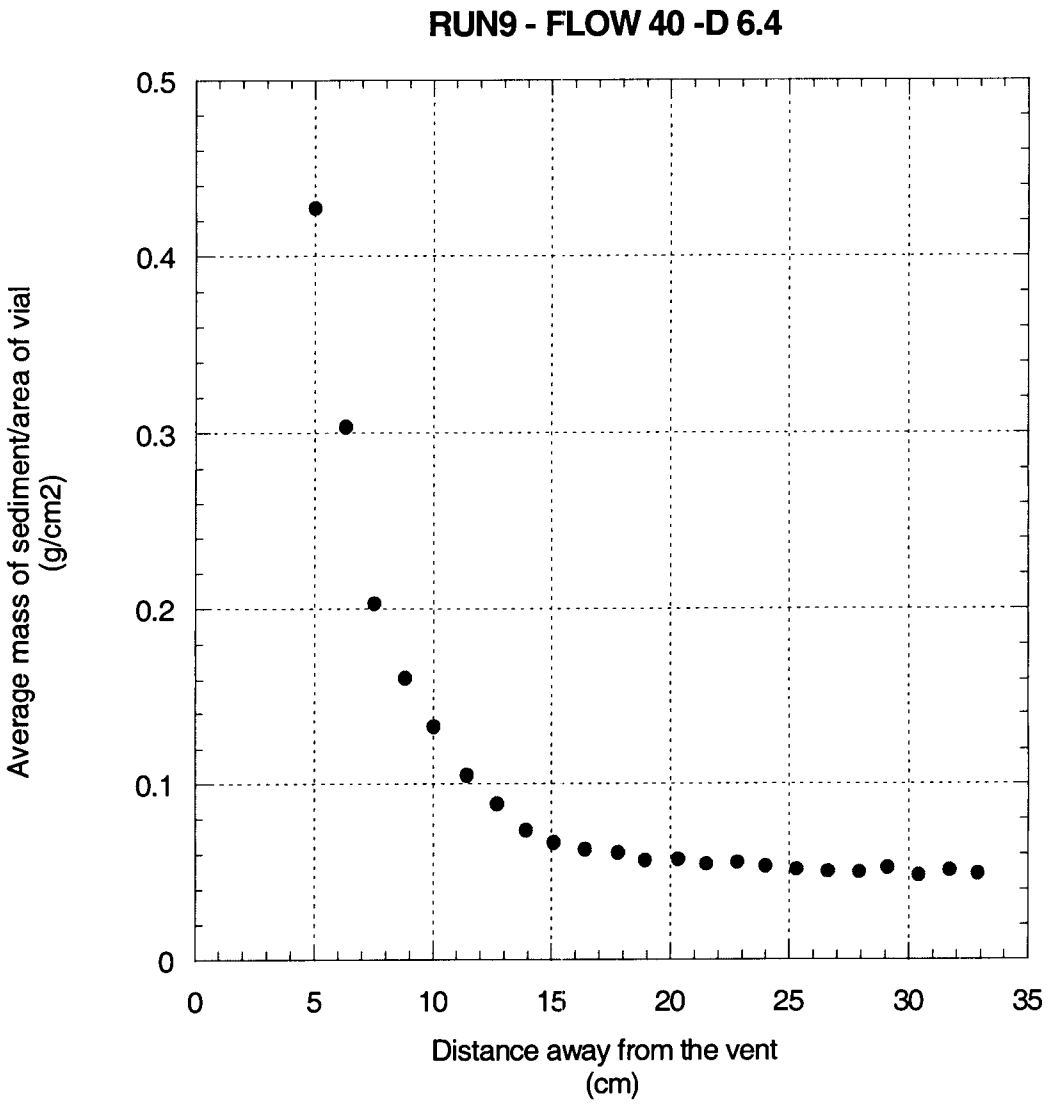
Vial 8.16 was toppled after the experiment.

Run 9

Data was lost. Graph is presented next:

Scientific Notebook
20-1402-461
Inititals: CC

Chuck Connor
Oct. 14,1999



Scientific Notebook
20-1402-461
Inititals: CC

Chuck Connor
Oct. 14,1999

Notes

Run9 *8/99*

Sediment type: Silicon glass beads

Vent diameter: 6.4 mm

Flow rate: 40 mL/min (flowmeter reading) or 10 mL/min (real flow)

Initial depth of water: 26.5 cm

Solute type and amount: 5.43 Kg of sugar

Initial density of mixture in tank: 1.03178 g/cc

Run time: 10 minutes

Total number of racks and alignment: 8 standard racks in all positions

Total number of vials: 168

Heating time in oven: 4 hours total

Cooling time in sealed case: 22 hours total

As soon as the plume appeared, it collapsed symmetrically. The plume then grew slowly, affected by the many bubbles that appeared in the plume, until it reached the surface of the water (about 5 minutes into the experiment). At this time, the shape and the depositional patterns of the plume are totally different from its shape at the beginning. In the beginning, a heavy collapsed plume seemed to deposit most of the sediment very close to the vent. The plume was very dense and quite compact. By the time the plume reaches the surface, it has an umbrella

shape, much wider and less dense than the plume at the beginning.
Also, the sediment seems to reach farther away when falling, causing a more spread deposition.
The plume showed (after the experiment) a fairly symmetric deposition, although it was a slightly slanted towards positions 4 or 5.
The flow was unstable, but it was kept between 38-42 mL/min (see flowmeter calibration).
The sediment lasted for 9 to 10 minutes.
The outlet was opened at about 8 minutes into the experiment, when the water level was at 27 cm. The original depth was a little lower than usual (26.25 cm).

NUMERICAL MODELLING OF MAGMA-REPOSITORY INTERACTIONS

Onno Bokhove

School of Mathematics, Bristol

The research presented in this report has been funded through a contract of the University of Bristol with the Center for Nuclear Waste Regulatory Analysis, South West Research Institute, San Antonio, Texas, U.S.A.

"Numerical modelling of magma-repository interactions", by O. Bokhove, version MagAir0.0.
September 1st, 1999.

Contents

1 Introduction 1

2 Flow-tube model for volatile-rich magma 4
- 2.1 Two fluid model: melt-volatiles and air . . . 4
- 2.2 Scaling . . . 5
- 2.3 One- and two-dimensional gas dynamics . . . 7

3 Numerical schemes for conservation laws 10
- 3.1 Spatial discretization . . . 10
- 3.2 Temporal discretization . . . 11
- 3.3 Characteristic decomposition . . . 11
- 3.4 Essentially Non-Oscillatory schemes . . . 13
- 3.5 Convex Essentially Non-Oscillatory schemes . . . 16
- 3.6 Forcing: gravity, geometry and dissipation . . . 16
- 3.7 Boundary conditions . . . 18
  - 3.7.1 Open or extrapolating . . . 18
  - 3.7.2 Characteristic lower boundary condition . . . 18
  - 3.7.3 Solid boundaries . . . 18
  - 3.7.4 Interface between melt-volatile fluid and air . . . 18
- 3.8 Structure of programs . . . 21
  - 3.8.1 Input file . . . 24
  - 3.8.2 Core program . . . 27

4 Numerical tests 29
- 4.1 Dynamics of magma . . . 29
  - 4.1.1 Stationary shock . . . 29
  - 4.1.2 Moving and reflected shock . . . 33
  - 4.1.3 Shock tube with and without tunnel end . . . 33
- 4.2 Dynamics in air . . . 41
  - 4.2.1 Stationary shock . . . 41
  - 4.2.2 Moving and reflected shock . . . 42
  - 4.2.3 Shock tube . . . 43
- 4.3 Dynamics of a melt-volatile fluid and air . . . 45
  - 4.3.1 Shock tube . . . 45

4.3.2 Resonating shocks 52

5 Magma-repository interactions **54**
5.1 Geometry and grid transformation 54
5.2 Basis simulation 56
5.3 Cannister movement and tracer advection 57

6 Conclusion **62**

A Retrieval of package **64**

B Makefile **65**

C Examples **69**
C.1 Example 1: stationary shock in magma 69
C.2 Example 2: shock tube in air 70
C.3 Example 3: basis simulation run 056 70

D Listings of enciso1.c and enoiso1.c **72**

Chapter 1

Introduction

This report explains the numerical programs behind a comprehensive modelling effort of magma-air interactions. Magma-air interactions occur when a dyke with high-volatile content magma ascends trough surrounding rock and encounters a drift filled with air at atmospheric pressure. While magma-air interactions do occur in natural systems such as lava tubes, the primary incentive has been to model volcanic flows in a magma dyke and a closed repository drift in which nuclear waste is stored. One significant question is whether magma flows that explode into a repository drift are able to destruct cannisters filled with nuclear waste and whether they are subsequently able to transport these cannisters or small pieces thereof. Explosive fluid dynamical events which develop after a volcanic dyke intrudes a series of repository drifts constitute one component in a hazard analysis for a nuclear repository site in a geological active area, such as the one proposed at Yucca Mountain (Connor *et al.* 1997). The present report describes in detail how these explosive volcanic flows have been modelled numerically.

Several simplifying assumptions will be introduced to provide a leading-order description of the fluid flow. Firstly, this study aims to understand the fluid dynamics of high-pressurized compressible magma ascending in a dyke and exploding into one side of a drift, but in section 5.3 we will also explicitly consider discrete cannister movement and passive tracer experiments. In particular, tracer experiments could model the advection of small waste pellets by the flowing magma. Secondly, the complicated three-dimensional fluid motion of magma and air is simplified into a spatially one-dimensional flow-tube model in an idealized geometry. Thirdly, the basaltic magma is a multiphase fluid of melt and liquid, and volatiles that will be parametrized as an isothermal, barotropic fluid with a monotonic relationship between pressure p and density ρ, according to Henry's law (Sparks 1978). This barotropic formulation of the equation of state is a leading-order approximation to the complicated multiphase fluid behaviour in which volatile gases exsolve from the melt into bubbles, and in which the magma becomes foam-like and the bubbles disrupt as a function of a continuing decompression (Wilson *et al.* 1980).

A reasonably realistic geometry emerges as follows. A magma dyke of characteristic width $1 - 2\,m$ ascends, forced by a pressure above lithostatic pressure with a speed of about $1 - 2\,m/s$ through the rock, from a magma chamber deep in the earth's crust. At a certain time, it encounters a series of drifts spaced about $80\,m$ apart at a depth of $300 - 400\,m$. This spacing defines the length of dyke that interacts with one tunnel, as is sketched in Fig. 1.1a. It is straightforward to modify the model to include a setting in which the dyke is not perpendicular to the drifts. The geometry is further straightened into a shock tube problem in which a vertical diaphragm between magmatic dyke and tunnel is instantaneously removed at time $t = 0$. This geometry is sketched in Fig. 1.1b and corresponds also to analogue experiments which

are complimentary to our overall investigations (Dr. A.M. LeJeune & Professor A.W. Woods, personal communication & observation). While the high-volatile content magma with it explosive potential is the focus of this work, low-volatile compressible magma have been considered in Woods and Sparks (1998) and in corresponding analogue experiments (Dr. A.M. LeJeune, personal communication and observation).

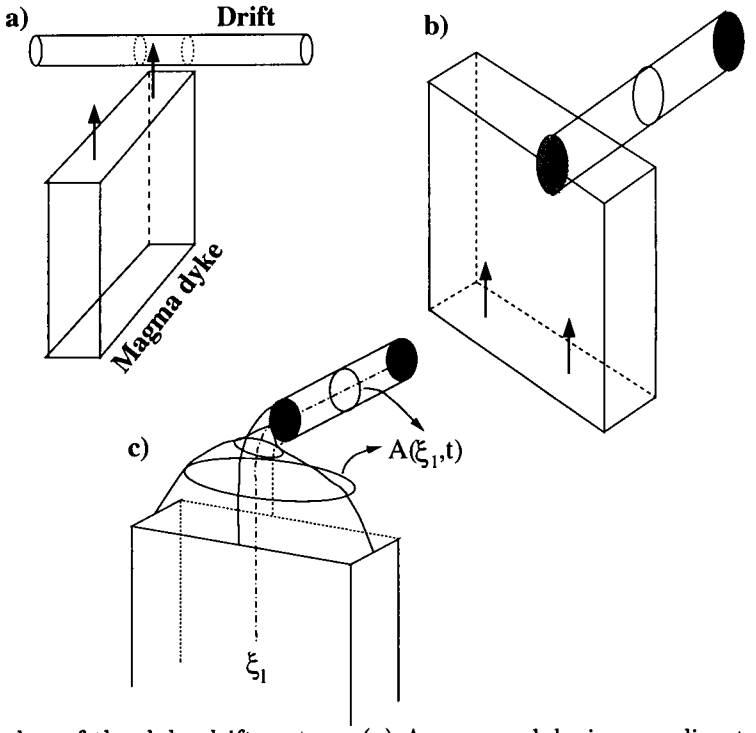


Figure 1.1: Sketches of the dyke-drift system. (a) A magma dyke is ascending towards a drift. (b) The idealized and experimental dyke-drift configuration in which flow starts after a vertical diaphragm (green/grey dashed patterned) between dyke and drift is removed instanteneously. In the experiment, a gum-rosin-acetone (GRA) mixture ascends in a vertical Hele-Shaw cell by imposing a vertical pressure gradient. GRA explodes into the under-pressured drift after removal of the diaphragm. (c) Sketch of a simplified configuration in a flow-tube model of the flow in dyke and tunnel. The flow in this model depends only on a smooth curvilinear coordinate ξ_1 which follows the dyke, and turns via an arc into the drift; variations of the cross-sectional area $A(\xi_1, t)$ of dyke, connecting arc, and drift are captured in the model.

The developed programs which will be described in this report consider transient compressible magma and air flows in separation, and in combination in one spatial dimension with a horizontal, vertical or curvilinear coordinate. Since shock waves and rarefaction waves are an essential feature of explosive volcanic dynamics, we have used established and modern Essentially Non-Oscillatory (ENO) shock capturing algorithms by Shu and Osher (1988, 1989) and Liu and Osher (1998). The employed algorithms have been tested and used before by the author for the shallow-water equations (Bokhove and Peregrine 1999). Shallow-water equations are identical to gas dynamics equations for an isothermal or isentropic gas in which pressure is proportional to density; depth in shallow-water dynamics plays a mathematical role that is similar to the role of density in gas dynamics. Shallow water dynamics could therefore be called pseudo compressible. The employed programs for compressible magma are thus adaptations of programs the author has used before in a shallow-water context; only the equation of state and ensuing interpretations have been changed. In

short, numerical methods for hyperbolic equations have validity in various branches of (pseudo) compressible geophysical and geological fluid dynamics, including shallow-water wave modelling and compressible magma flows. Several preliminary tests have been performed with codes that model volcanic and air flows, in separation and together in horizontal pipes and in dyke and drift systems. The dynamics of an interface is modelled when magma and air coexist and strongly influence one another. Varying cross-sections, gravity and parametrized frictional effects are ultimately all taken into account. The final code is validated by comparing idealized simulations in magma, air, and in magma and air seperated by an interface, respectively, with exact (implicit) solutions of stationary shocks, moving and reflected shocks, and shock tube solutions. Moreover, two different algorithms and associated programs have been used in the validation phase.

The organization of this reports is as follows: (i) The equations of motion for the compressible magma and air dynamics, and the geometry are presented in chapter 2. One-dimensional flow-tube equations and two-dimensional equations in a vertical plane are derived as simplifications of the three-dimensional equations of motion. These equations are rendered into dimensionless form to facilitate numerics. (ii) Numerical algorithms are explained in detail in chapter 3. Two shock-capturing numerical algorithms, a Local Lax Friedrichs and Essentially Non-Oscillatory scheme, are introduced for hyperbolic and/or conservative systems. The conservative terms in the equations of motion are discretized carefully to model shock propagation and seperately from the other, forcing, disspative, gravitational and geometric terms. Boundary conditions and interface conditions between magma and air require careful attention because numerical interference and instability can occur between conservative and the other —gravitational or geometrical— terms. (iii) The description of the structure of the actual programs (section 3.8) intends to be sufficiently detailed to learn how to use the programs. All programs have been written in the C-language, and have been used with minor adaptation of the input and output procedures on cc- and gcc-compilers on a Sun Ultrasparc 2200 with two CPU's running at $200\,MHz$, and on PC's with Pentium-II/III processors and one CPU running at $250 - 350\,MHz$, respectively. Matlab graphics programs are available to plot the produced data; the latter are stored separately to allow use of other graphics packages. (iv) A suite of numerical tests validate the final code and also form an introduction to learning how to use the programs (chapter 4). Numerical tests consist of comparing numerical exact (inviscid) solutions in a horizontal tube. Further comparison with stationary solutions of volcanic flows in hydaulically controlled horizontal nozzles, in vertical conduits (Woods 1995) or curved conduits forms an as yet unexploited possibility. Finally, several Appendices contain instructions on retrieval of the numerical package, worked examples for running and plotting some of the codes, the Makefile, and the program-listings of two codes.

It is unnecessary to read this report in chronological order when the goal is to use or modify the developed numerical programs. To get acquainted with the code, it is best to start with chapter 2; continue with sections 3.1, 3.2, and 3.8; and then follow the instructions in Appendices A and C while browsing through the Makefile in Appendix B and the remainder of this report.

Chapter 2

Flow-tube model for volatile-rich magma

The fluid equations of the compressible dynamics of magma and air will be introduced in turn. These equations form the basis for subsequent analytical and numerical investigations of explosive magma-air dynamics.

2.1 Two fluid model: melt-volatiles and air

The two-phase basaltic fluid of melt and volatiles will be modelled as an isothermal compressible fluid with a parametrized equation of state. The density ρ of this fluid equals the reciprocal of the volume occupied by a unit mass of the mixture of exsolved volatiles (gas), dissolved volatiles (liquid) and melt. The mass fraction of volatiles is $n(p)$. Dissolved volatiles and melt are lumped together as an incompressible mixture of mass fraction $1 - n(p)$. Division of these mass fractions by the volatile density, and lumped rock and liquid density, and summation gives an averaged density as function of pressure p

$$\rho(p) = \left(\frac{n(p)\, R_v\, T}{p} + \frac{1 - n(p)}{\sigma} \right)^{-1} \tag{1}$$

as a function of pressure p, which depends on several parameters: $R_v \approx 462\, J\, kg^{-1}\, K^{-1}$ the mixture's gas constant, fixed temperature $T = 1000 - 1200\, K$, and lumped melt density $\sigma \approx 2500\, kg\, m^{-3}$. Bubbles and melt are assumed to be in equilibrium such that Henry's law (Sparks 1978)

$$n(p) = n_0 - n_{s_H} \equiv n_0 - s_H\, p^{\beta} \tag{2}$$

applies with total volatile content $n_0 = 0.5 - 2\, wt\%$ and with Henry's constant $s_H \approx 3 * 10^{-6}\, Pa^{-1/2}$ and $\beta \approx 1/2$ for basaltic high-volatile content magmas. The barotropic fluid with equation of state (1) and Henry's law (2) is a leading-order approximation to the complicated physics of a multiphase basaltic melt-volatile mixture. In case of supersaturation the mass fraction $n(p)$ may be argued to become time dependent (Woods 1995). More complicated models would also examine the influence of relative motion between melt and bubbles. For typical volcanic parameters, the values $\rho(p)$ and sound speed $a(p) = \sqrt{\partial p / \partial \rho}$ have been drawn in Fig. 2.1. As $p \to p_i$ the sound speed $a(p) \to \infty$. In part because liquid and melt density have been lumped together, does the incompressibility pressure p_i not equal the pressure for which all volatiles become dissolved, i.e. $n(p = p_c) = 0$ and the equation of state requires correction for high pressures around

p_c. In the subsequent study we will only consider pressures $p < p_c$, although one could in principle couple the incompressible and compressible regions of the two-phase fluid modelled by (1) and (2) for $p < p_c$, and $\rho = \rho_c = \sigma$ for $p > p_c$.

In the analogue laboratory experiment the melt-volatile mixture is replaced by a gum-rosin-acetone (GRA) multiphase fluid. The gum-rosin has properties similar to rhiolite and the acetone is the volatile. Gum rosin is a brittle amphorphous solid at room temperature that changes to a liquid with a small amount of dissolved acetone (Phillips *et al.* 1995). The properties of GRA are incompletely understood (A.W. Woods and R.S.J. Sparks, personal communication) but to a certain extent reproduce the phase behaviour and volatile dependence of viscosity of hydrated magmas. While the volatile fractions in the gaseous and liquid phase in basaltic magma change as function of pressure, assuming that an instantenous equilibrium is established between the two phases, GRA appears to be a boiling mixture in which a large fraction of dissolved acetone changes almost instantenously into exsolved form when pressure decreases quickly, and vice versa (presumably with hysteresis effects). After this initial transition further exsolution appears to be negligible on the time scales of rapidly decompressing flows we are likely to encounter in a laboratory experiment. Pending more knowledge on the equation of state for the GRA multiphase fluid, GRA is modelled with fixed mass fraction $n(p) = n_0$ in which n_0 is a free parameter induced from the laboratory experiment, either by trial and error and comparison with the laboratory results, or by measuring the void fraction in the Hele Shaw cell before GRA enters into the tunnel but after boiling has occurred.

The three-dimensional compressible momentum and continuity equations for magma with equation of state (1) are

$$\begin{aligned} \frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u}\cdot\boldsymbol{\nabla}_3)\,\mathbf{u} + \frac{1}{\rho}\,\boldsymbol{\nabla}_3 p &= -g\,\hat{\mathbf{z}} - \mu\,\mathbf{F}, \\ \frac{\partial \rho}{\partial t} + \boldsymbol{\nabla}_3\cdot(\rho\,\mathbf{v}) &= 0, \end{aligned} \tag{3}$$

with three-dimensional velocity $\mathbf{u}$ and gradient $\boldsymbol{\nabla}_3$, gravity g, vertical unit vector $\hat{\mathbf{z}}$, frictional or forcing parameter μ and friction or forcing $\mathbf{F}$.

The compressible equations of motion for air (e.g. Batchelor 1967) are

$$\begin{aligned} \frac{\partial \mathbf{u}_a}{\partial t} + (\mathbf{u}_a\cdot\boldsymbol{\nabla}_3)\mathbf{u}_a + \frac{1}{\rho_a}\,\boldsymbol{\nabla}_3 p_a &= -g\,\hat{\mathbf{z}} - \mathbf{F}_a, \\ \frac{\partial \rho_a}{\partial t} + \boldsymbol{\nabla}_3\cdot(\rho_a\,\mathbf{u}_a) &= 0, \\ \frac{\partial s}{\partial t} + (\mathbf{u}_a\cdot\boldsymbol{\nabla}_3)\,s &= 0 \end{aligned} \tag{4}$$

with subscripts "a" distinguishing variables in air from those in the basaltic fluid, friction of forcing $\mathbf{F}_a$, entropy s and gas constant $R = 287.04\,J\,kg^{-1}\,K^{-1}$. To close the system, we need the first law of thermodynamics

$$\mathrm{d}U = T_a\,\mathrm{d}s - p\,\mathrm{d}(1/\rho) \tag{5}$$

with air temperature T_a and an equation of state, for which we choose the ideal gas law

$$p_a = \rho_a\,R\,T_a. \tag{6}$$

2.2 Scaling

Numerical integration is performed using dimensionless equations. Numerical values can then be chosen to lie around unity to improve numerical stability. By considering the equations of motion and state for magma,

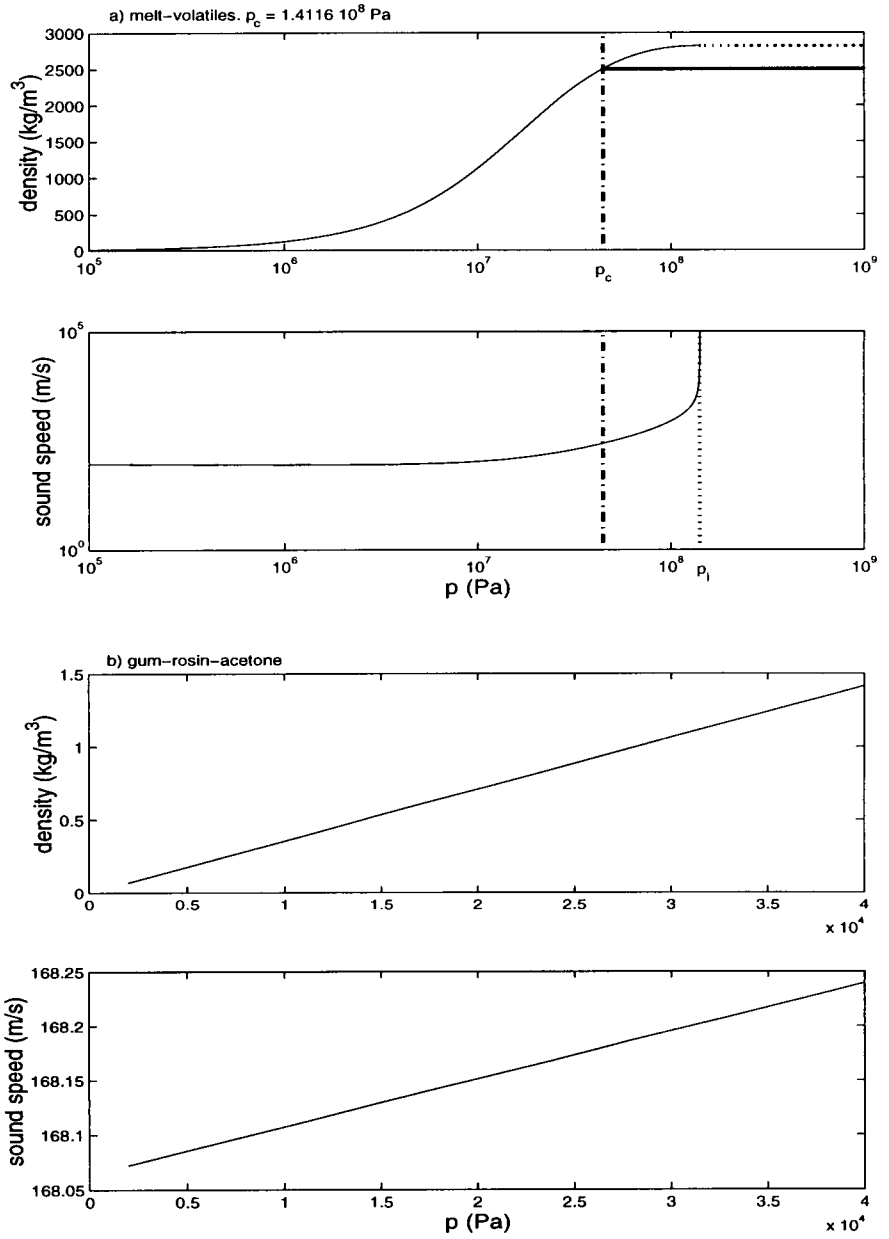


Figure 2.1: Graphs of idealized equations of state for: (a) a basaltic high-volatile content magma with $T = 1000\,K, \sigma = 2500\,kg\,m^{-3}, n_0 = 2\,wt\%, s_H = 3 * 10^{-6}\,Pa^{-1/2}, R_v = 462\,J\,kg^{-1}\,K^{-1}, \beta = 0.5$. (b) gum-rosin-acetone mixture with $T = 293\,K, \sigma = 1080\,kg\,m^{-3}, n_0 = 20\,wt\%, s_H = 6.749 * 10^{-5}\,Pa^{-1/2}, R_v = 482\,J\,kg^{-1}\,K^{-1}, \beta = 0.7782$. (i) Density and (ii) sound speed are drawn as function of pressure.

the following scalings for various parameters are suggested:

$$\begin{aligned} p &= P_0\, p', \quad \rho = \rho_0\, \rho', \quad t = (L/U)\, t', \quad \mathbf{x} = L\, \mathbf{x}', \\ \mathbf{u} &= U\, \mathbf{u}, \quad n(p) = n_0\, n'(p'), \quad \sigma = \rho_0\, \sigma' \end{aligned}$$

with $P_0 = n_0\, R_v\, T\, \rho_0$ and $U^2 = P_0/\rho_0$. Upon dropping the primes, the dimensionless (scaled) melt-volatile equations read

$$\begin{aligned} \frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \boldsymbol{\nabla}_3 \mathbf{u} + \frac{1}{\rho}\, \boldsymbol{\nabla}_3 p &= -\frac{1}{F_l}\, \hat{\mathbf{z}} - \nu\, \mathbf{F}, \\ \frac{\partial \rho}{\partial t} + \boldsymbol{\nabla}_3 \cdot (\rho\, \mathbf{v}) &= 0, \end{aligned} \tag{7}$$

with equation of state

$$\rho(p) = \left(\frac{n(p)}{p} + \frac{1 - n_0\, n(p)}{\sigma} \right)^{-1} \tag{8}$$

and Henry's law

$$n(p) = 1 - \epsilon\, p^{\beta} \tag{9}$$

with Froude number $F_l = P_0/(\rho_0\, g\, L)$, friction number $\nu = \mu_0/(L\, \sqrt{P_0\, \rho_0})$, and $\epsilon = s\, {P_0}^{\beta}/n_0$. This function is implemented in the core code, see section 3.8, as function Density(pp). For a given density the pressure is determined by a root finding routine which uses the function Dens(pp, d0) in the core program. A cursory examination of Fig. 2.1a) around pressure values of $10^6\, Pa$ and $3 * 10^4\, Pa$, respectively, suggests that values of $\rho_0 \approx 100\, kg\, m^{-3}$ and $\rho_0 \approx 1\, kg\, m^{-3}$, respectively, are reasonable for the volcanic fluid and GRA. Length scale L is chosen to be the radius d of the tunnel.

2.3 One- and two-dimensional gas dynamics

Consider a system of curvilinear coordinates ξ_i $(i = 1, 2, 3)$ with scale factors h_i (Batchelor 1967). An infinitesimal cube in such a coordinate system then has sides $h_i\, \mathrm{d}\xi_i$ and volume $d\tau$ given by $(\mathrm{d}\tau)^2 = \sum_{i=1}^{3} (h_i\, \mathrm{d}\xi_i)^2$.

In the dyke-drift configuration a representative $\xi_1(x, y, z)$-coordinate may be identified with corresponding cross-sectional area $A(\xi_1, t)$, normal to the ξ_1-isolines, as is illustrated in Fig. 1.1c by the dashed-dotted lines. The time-dependence of the cross-sectional area A corresponds to the reponse of the rock walls to pressures in the magma dyke or drift, or to movement of objects in the tunnel as a consequence of pressure gradients. A one-dimensional system, obtained by averaging over cross-sections and by neglecting flows in cross-sectional planes, is derived from (7) by considering mass and momentum density conservation in a control volume $h_1\, A(\xi_1, t)\, d\xi_1$. Momentum density is defined as $\rho\, u_1$ and $h_1 = h_1(\xi_1)$ is the scale factor associated with the curvilinear coordinate. Pressure in the ξ_1-direction acts both on the slices at ξ_1 and $\xi_1 + \Delta\xi_1$, giving a contribution $\sim -\partial(p\, A)/\partial\xi_1$, and on the flow tube walls between these slices, giving a contribution $\sim p\, \partial A/\partial\xi_1$. The gravitational force is a volume force. For the basaltic fluid one then finds the following equations of motion (e.g. Whitham 1974):

$$\begin{aligned} \frac{\partial(\rho\, A\, u)}{\partial t} + \frac{1}{h_1} \frac{\partial}{\partial \xi_1} \left(\rho\, A\, u^2 + p\, A \right) &= -A\, F_1 + \frac{p}{h_1} \frac{\partial A}{\partial \xi_1} - \frac{\rho\, A}{F_l\, h_1} \frac{\partial z(\xi_1)}{\partial \xi_1}, \\ \frac{\partial(\rho\, A)}{\partial t} + \frac{1}{h_1} \frac{\partial}{\partial \xi_1} \left(\rho\, A\, u \right) &= 0, \end{aligned} \tag{10}$$

where $u = u_1$ is the ξ_1-component of the velocity, and $-F_1$ the parametrized forcing and dissipation. In a vertical dyke or conduit, we have $\xi_1 = z$ and $h_1 = 1$, and in a horizontal tunnel $\xi_1 = x$ and $h_1 = 1$. Corner effects at the tunnel's entrance are simplified by considering geometries with smooth dyke-drift transitions (cf. Fig. 1.1b). In essence we follow the flow along the (dashed-dotted) centerline $\xi_1 = 0$ in Fig. 2.2a). The evolution of the energy density of magma is governed by

$$\begin{aligned} \frac{\partial}{\partial t}\left(\frac{1}{2}\,\rho\,A\,u^2 + \rho\,A\,[U_b(\rho) + \frac{1}{F_l}\,(z - z_0)]\right) \;+\; \frac{1}{h_1}\,\frac{\partial}{\partial \xi_1}\left[\left(\frac{1}{2}\,\rho\,A\,u^2 + \rho\,A\,[U_b(\rho) + \frac{1}{F_l}\,(z - z_0)] + p\,A\right) u\right] \\ = \; -\rho^2\,\frac{\partial U_b}{\partial \rho}\,\frac{\partial A}{\partial t} - A\,u\,F_1, \end{aligned} \tag{11}$$

where

$$U_b \;=\; \int_{p_0}^{p} \mathrm{d}\,\tilde{p}\,\frac{1}{\rho(\tilde{p})} \tag{12}$$

is the internal energy of the basaltic flow relative to some reference pressure p_0, and z_0 is a reference level. Total, kinetic, and potential plus internal energy are used to monitor some of the simulations. The measurement routine Measure() calculates and stores total, kinetic, and potential plus internal energy.

It is not well understood how to model the frictional forces of the bubbly magmatic liquid. An effective viscosity has been shown to increase with volatile exsolution and also with the pressure of bubbles (Jaupart and Allègre 1991), and an emperical parametrization of the high-viscosity magmatic foam was proposed, averaged over the cross-sectional area and proportional to the velocity

$$F_1 = \nu\,\frac{3\,e^{(5-100\,n_0)}\,(1-\phi)^{-5/2}}{2\,L_e^2}\,u, \tag{13}$$

with L_e a typical length scale, e.g. the width of the dyke or the radius of drift or conduit, and dimensionless parameter $\nu = \mu_0/(L\sqrt{P_0\,\rho_0})$ with $\mu_0 = (10 - 100)\,kg\,m^{-1}\,s^{-1}$. Relation (13) has validity when the vesicularity or void fraction

$$\phi = \frac{1}{1 + (1 - n_0\,n)\,p/(n\,\sigma)} \tag{14}$$

of the mixture remains below the point of fragmentation $\alpha = 70 - 90\%$ (Woods 1995). Beyond the fragmentation point, for $\phi \geq \alpha$, the gas becomes the continuous phase and frictional forces diminish. In this regime, a simple parametrization for turbulent flow in a pipe was proposed by Wilson *et al.* (1980):

$$F_1 = 0.0025\,\frac{\rho\,u^2}{L_e}. \tag{15}$$

Both parametrizations are implemented in the more involved core program in the function Forcing().

The scaled compressible equations of motion for air in a flow tube are:

$$\begin{aligned} \frac{\partial(\rho_a\,A\,u_a)}{\partial t} + \frac{1}{h_1}\,\frac{\partial}{\partial \xi_1}\left(\rho_a\,A\,u_a^2 + p_a\,A\right) &= -A\,F_a + \frac{p_a}{h_1}\,\frac{\partial A}{\partial \xi_1} - \frac{\rho_a\,A}{F_l, h_1}\,\frac{\partial z(\xi_1)}{\partial \xi_1}, \\ \frac{\partial(\rho_a\,A)}{\partial t} + \frac{1}{h_1}\,\frac{\partial}{\partial \xi_1}\left(\rho_a\,A\,u_a\right) &= 0, \\ \frac{\partial \Omega}{\partial t} + \frac{1}{h_1}\,\frac{\partial}{\partial \xi_1}\left((\Omega + p_a)\,u_a\right) &= -p\,\frac{\partial A}{\partial t} - u_a\,A\,F_a \end{aligned} \tag{16}$$

in which

$$\Omega = \rho_a\,A\left(\frac{1}{2}\,u_a^2 + e + \frac{1}{F_l}\,(z - z_0)\right) \tag{17}$$

is the energy density of air with internal energy e. Air is modelled as an ideal gas with $p_a = \kappa(s)\,\rho_a^\gamma = (\gamma - 1)\,\rho_a\,e$, where $\kappa = \kappa(s)$ is a function of entropy s and $\gamma = c_p/c_v = 1.4$ is the ratio of specific heats at constant pressure and volume, respectively. Viscous forces in air will be ignored, i.e. $F_a = 0$.

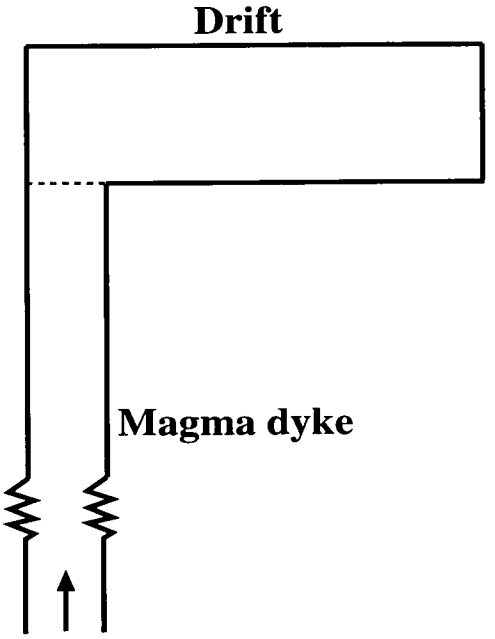


Figure 2.2: The effective two-dimensional domain in a vertical plane for flows averaged in the y-direction.

A similar approach may be followed to derive equations for representative flow in a vertical cross section with width $\Delta y = b(x, z, t)$, see Fig. 2.2. The corresponding two-dimensional equations of motion for basaltic magma are

$$\begin{aligned}
\frac{\partial(\rho\, b\, u)}{\partial t} + \frac{\partial(\rho\, b\, u^2 + p\, b)}{\partial x} + \frac{\partial(\rho\, b\, u\, w)}{\partial z} &= p\,\frac{\partial b}{\partial x} - F_1, \\
\frac{\partial(\rho\, b)}{\partial t} + \frac{\partial(\rho\, u\, b)}{\partial x} + \frac{\partial(\rho\, w\, b)}{\partial z} &= 0, \\
\frac{\partial(\rho\, b\, w)}{\partial t} + \frac{\partial(\rho\, b\, u\, w)}{\partial x} + \frac{\partial(\rho\, b\, w^2 + p\, b)}{\partial z} &= p\,\frac{\partial b}{\partial z} - \frac{1}{F_l}\,\rho - F_3,
\end{aligned} \tag{18}$$

and similar equations emerge for air.

Chapter 3

Numerical schemes for conservation laws

3.1 Spatial discretization

Essentially non-oscillatory (ENO) finite-difference numerical schemes have been used to discretise (10) and (16) on a uniform grid. The last decades have seen major improvements in shock-capturing numerical schemes. We primarily used a Local Lax Friedrichs (LLF2), which is second order in space and time, and have compared this scheme with a third-order Essentially Non-Oscillatory (ENO) scheme by Shu and Osher (1988, 1989) in all numerical tests. The speed of the LLF2 scheme favoured its use in the simulations of magma-repository interactions in a dyke-drift geometry. An extension of the LLF2 scheme to a third-order Convex ENO scheme, as developed by Liu and Osher (1998), is straightforward but has not yet been completed.

All the above-mentioned shock-capturing schemes integrate the equations in conservative form. Consider a system of conservation laws

$$\mathbf{q}_t + \sum_{k=1}^{d} \mathbf{f}_k(\mathbf{q})_{x_k} = \mathbf{q}_t + \mathcal{L}(\mathbf{q}) \quad = \quad \mathbf{F}^\dagger(q, x, t) \tag{1}$$

with initial condition $\mathbf{q}(\mathbf{x}, 0) = \mathbf{q}_0(\mathbf{x})$, $\mathbf{q} = (q_1, \ldots, q_n)^T, \mathbf{x} = (x_1, \ldots, x_d)^T$, and additional forcing, dissipative and/or geometric terms are denoted by $\mathbf{F}^\dagger$. On the computational grid $\mathbf{x}_j = j\,\Delta\mathbf{x}$ and $t_n = n\,\Delta t$, and $\mathbf{q}_j^n$ is the computed approximation of the exact solution $\mathbf{q}(\mathbf{x}_j, t_n)$. We continue our description of the algorithms for simplicity in one spatial dimension, where we have $\mathbf{q}_t + \mathbf{f}(\mathbf{q})_x = \mathbf{F}^\dagger$. Non-uniform grids may be transformed to uniform ones under a smooth transformation $\phi = \phi(x)$. This is of interest when we wish to model the flow far in the dyke on a coarse grid and the flow around the entrance of the tunnel and the flow in the tunnel on a fine grid in order to enhance the resolution of shock waves around the tunnel. All numerical methods presented here remain valid in this stretched curvilinear framework (Shu 1997), where we have $\mathbf{q}_t + \phi_x\, f(\mathbf{q})_\phi = \mathbf{F}^\dagger$.

The conservative schemes considered have a discretization of the form

$$\frac{dq}{dt} = q_j^n - \lambda\,(\hat{f}_{j+1/2} - \hat{f}_{j-1/2}) + F_j^\dagger, \tag{2}$$

with $\lambda = \Delta t/\Delta x$, for an appropriate higher-order flux $\hat{\mathbf{f}}$ such that $L(q) = \mathcal{L}(q) + O\big((\Delta x)^m\big)$. (When no confusion arises vector notation is dropped.)

The LLF2, ENO, or Convex ENO schemes use an adapative stencil to calculate the higher-order flux $\hat{f}_{j+1/2}$. An adapative stencil uses information on r grid points to the left and s grid points to the right of the central grid point j. The stencil is determined independently at each grid point j, which fixes r and s, based on information at surrounding grid points such that discontinuities are avoided. Information of at least one upwind gridpoint is taken into account. ENO and Convex ENO schemes use higher-order interpolating formulae, while a wave limiter is used in LLF2 to choose the smoothest stencil.

Discretization in two dimensions on a regular grid is a line-by-line extension of the one-dimensional ENO and convex ENO algorithms.

3.2 Temporal discretization

Either a second-order or third-order total variation diminishing (TVD) Runge-Kutta time discretization (Shu and Osher 1988, 1989) has been used:

$$\begin{aligned} q^{(1)} &= q^n + \Delta t\,[L(q^n) + F^{\dagger^n}] \\ q^{n+1} &= \frac{1}{2}\left(q^n + q^{(1)} + \Delta t\,[L(q^{(1)}) + F^{\dagger^{(1)}}]\right) \end{aligned} \tag{3}$$

and

$$\begin{aligned} q^{(1)} &= q^n + \Delta t\,[L(q^n) + F^{\dagger^n}] \\ q^{(2)} &= \frac{1}{4}\left(3\,q^n + q^{(1)} + \Delta t\,[L(q^{(1)} + F^{\dagger^{(1)}}]\right) \\ q^{n+1} &= \frac{1}{3}\left(q^n + 2\,q^{(2)} + 2\,\Delta t\,[L(q^{(2)}) + F^{\dagger^{(2)}}]\right). \end{aligned} \tag{4}$$

When the terms denoted by $F^\dagger$ contain no spatial derivative of the variables q, their descretization in time is straightforward. When (part of) the flux $\hat{f}$ balances $F^\dagger$ in stationary flows, which occurs for example in quiescent hydrostatic balanced flows, such a straightforward discretization may violate that balance at higher order and an adapted strategy may be required (see the discussion in section 3.6).

3.3 Characteristic decomposition

Systems of conservation laws can in general be written in the form

$$\mathbf{q}_t + \mathbf{A}\,\mathbf{q}_x = 0. \tag{5}$$

When this system is hyperbolic, it can be decomposed into characteristic form by using the left $\mathbf{R}^{-1}$ and right $\mathbf{R}$ eigenvector matrices of $\mathbf{A}$. Multiplication by $\mathbf{R}^{-1}$ brings (5) into characteristic form:

$$\mathbf{R}^{-1}\,\mathbf{q}_t + \mathbf{\Lambda}\,\mathbf{R}^{-1}\mathbf{q}_x = 0 \tag{6}$$

with diagonal eigenvalue matrix $\mathbf{\Lambda}$ and identity matrix $\mathbf{I} = \mathbf{R}^{-1}\,\mathbf{R}$. Matrix $\mathbf{\Lambda}$ is locally taken as constant and fluxes are evaluated at cell edges in ENO interpolations. The fluxes $\hat{\mathbf{f}}_{j+1/2}$ in the original formulation are found by projecting back from eigenvector space using the right eigenvector matrix $\mathbf{R}$. Roe-averaged

quantities (Shu and Osher 1989) are used to calculate variables at the cell edge $j+1/2$ from the variables at finite-difference points j and $j+1$:

$$\begin{aligned} u_{j+1/2} &= \frac{u_j\sqrt{\rho_j}+u_{j+1}\sqrt{\rho_{j+1}}}{\sqrt{\rho_j}+\sqrt{\rho_{j+1}}} \\ \rho_{j+1/2} &= \frac{1}{2}(\rho_j+\rho_{j+1}) \\ H_{j+1/2} &= \frac{H_j,\sqrt{\rho_j}+H_{j+1}\sqrt{\rho_{j+1}}}{\sqrt{\rho_j}+\sqrt{\rho_{j+1}}} \end{aligned} \tag{7}$$

with $H=(\tilde{\Omega}+p)/\rho$ and $\tilde{\Omega}=(1/2)\,\rho\,u^2+\rho\,e$ used only for air dynamics. Projection of the fluxes and variables onto the space of eigenvectors and back is computationally expensive.

With base variables $q=\{m=\rho\,u,\rho\}$ for magma, taking $A=h_1=1$ momentarily, matrices $\mathbf{A},\mathbf{R}^{-1}$ and $\mathbf{R}$ have the form:

$$\mathbf{A}=\begin{pmatrix} 2\,u & a^2-u^2 \\ 1 & 0 \end{pmatrix},\quad \mathbf{R}=\begin{pmatrix} u-a & u+a \\ 1 & 1 \end{pmatrix},\quad \mathbf{R}^{-1}=\frac{1}{2\,a}\begin{pmatrix} -1 & u+a \\ 1 & a-u \end{pmatrix} \tag{8}$$

with sound speed $a(\rho)=a(p)=\sqrt{\partial p/\partial\rho}$. The eigenvalues $\mathbf{\Lambda}$ are $u\pm a$ and the characteristic equations are:

$$\frac{\partial(u\pm c)}{\partial t}+(u\pm a)\,\frac{\partial u\pm c}{\partial x}=0 \tag{9}$$

with $c(\rho)=c(p)=\int^p \mathrm{d}\tilde{p}\,[1/\rho(\tilde{p})]\,\sqrt{\partial\rho/\partial\tilde{p}}$.

With base variables $q=\{m_a=\rho_a\,u_a,\rho_a,\tilde{\Omega}\}$ for air, taking $A=h_1=1$, matrices $\mathbf{A},\mathbf{R}^{-1}$ and $\mathbf{R}$ have the form (subscripts "a" have been dropped momentarily):

$$\mathbf{A}=\begin{pmatrix} (2-\gamma)\,u & \frac{1}{2}(\gamma-3)\,u^2 & (\gamma-1) \\ 1 & -u & 0 \\ [\frac{\gamma\tilde{\Omega}}{\rho}-\frac{3}{2}(\gamma-1)\,u^2] & [(\gamma-1)\,u^3-\frac{\gamma\tilde{\Omega}\,u}{\rho}] & (\gamma-1)\,u \end{pmatrix}, \tag{10}$$

$$\mathbf{R}=\begin{pmatrix} u-c_a & u+c_a & u \\ 1 & 1 & 1 \\ (H-u\,c_a) & (H+u\,c_a) & \frac{1}{2}u^2 \end{pmatrix},\quad \mathbf{R}^{-1}=\begin{pmatrix} -\frac{1}{2}(\frac{1}{c_a}+u\,b_1) & (\frac{1}{2}b_2+\frac{u}{c}) & \frac{1}{2}b_1 \\ \frac{1}{2}(\frac{1}{c_a}-u\,b_1) & (\frac{1}{2}b_2-\frac{u}{c}) & \frac{1}{2}b_1 \\ u\,b_1 & (1-b_2) & -b_1 \end{pmatrix} \tag{11}$$

with sound speed $c_a=\sqrt{\gamma p/\rho}$, $b_1=(\gamma-1)/c_a^2$, $b_2=\frac{1}{2}b_1\,u^2$ and $H=(\tilde{\Omega}+p)/\rho$. The eigenvalues $\mathbf{\Lambda}$ are $\{u\pm c_a,u\}$ and the characteristic equations are:

$$\begin{aligned} \frac{\partial p}{\partial t}+(u\pm c_a)\,\frac{\partial p}{\partial x}\pm\rho\,c_a\left(\frac{\partial u}{\partial t}+(u\pm c_a)\,\frac{\partial u}{\partial x}\right) &= 0, \\ \frac{\partial s}{\partial t}+u\,\frac{\partial s}{\partial x} &= 0. \end{aligned} \tag{12}$$

In the actual code, we use $q=\{m_1=\rho\,A\,u;\,\rho\,A;\,m_2=\rho\,A\,v;\,\hat{\Omega}=A\,\tilde{\Omega}\}$ where v is the y-velocity. For magma v and $\hat{\Omega}$ are initialized to zero and remain zero, while for air v remains zero. For uniform pipes one takes $A=1$. Lateral momentum m_2 is necessary in a two-dimensional version of the code in which A is replaced b. The involved matrices have been updated accordingly.

3.4 Essentially Non-Oscillatory schemes

The ENO algorithm for finite difference grids was developed by Shu and Osher (1988, 1989) (see also Shu 1997) [1]. The ENO calculation of higher-order flux $\hat{f}_{i+1/2}$ in one dimension is based on the use of adaptive "stencils", in which variables on r grid points left and s grid points to the right of grid point i are used to find the flux at cell edge $i+1/2$. A higher-order interpolating formula is constructed at each time step by using information from surrounding grid points such that discontinuities are avoided. At least one upwind grid point occupies the stencil.

ENO schemes use the concept of divided differences. Divided differences $V[\cdot]$ of a function $V(x)$ are defined recursively. Starting with

$$V[x_{i-1/2}] = V(x_{i-1/2}), \tag{13}$$

the kth-degree divided differences for $i >= 1$ are defined by

$$V[x_{i-1/2}, \ldots, x_{i+k-1/2}] = \frac{V[x_{i+1/2}, \ldots, x_{i+k-1/2}] - V[x_{i-1/2}, \ldots, x_{i+k-3/2}]}{x_{i+k-1/2} - x_{i-1/2}}. \tag{14}$$

An important property of divided differences is that

$$V[x_{i-1/2}, \ldots, x_{i+k-1/2}] = \frac{V^k(\xi)}{k!}, \tag{15}$$

where V^k is the kth-derivative of $V(x)$, for some ξ inside the stencil $[x_{i-1/2}, x_{i+k-1/2}]$ provided function $V(x)$ is smooth in this stencil. When $V(x)$ is discontinuous in the above stencil, then

$$V[x_{i-1/2}, \ldots, x_{i+k-1/2}] = O(\frac{1}{\Delta x}); \tag{16}$$

divided differences can thus be used to detect discontinuities and to choose the smoothest stencil around or at a certain grid point.

When function $h(x)$ satisfies

$$f(q(x)) = \frac{1}{\Delta x} \int_{x-\Delta x/2}^{x+\Delta x/2} \mathrm{d}\tilde{x}\, h(\tilde{x}) \tag{17}$$

then we have

$$\frac{\partial f(q(x))}{\partial x} = \Big(h(x + \Delta x/2) - h(x - \Delta x/2) \Big) / \Delta x. \tag{18}$$

Flux $\partial f/\partial x$ is approximated by a numerical flux

$$\left.\frac{\partial f}{\partial x}\right|_i = \frac{\hat{f}_{i+1/2} - \hat{f}_{i-1/2}}{x_{i+1/2} - x_{i-1/2}} \tag{19}$$

and so $\hat{f}_{i+1/2} \approx h(x_{i+1/2})$. Consider the primitive function $H(x)$ of $h(x)$

$$H(x) = \int_{-\infty}^{x} \mathrm{d}\tilde{x}\, h(\tilde{x}) \Rightarrow \partial H/\partial x = h \tag{20}$$

at $x_{i+1/2}$; we find

$$H(x_{i+1/2}) = \int_{\infty}^{x_{i+1/2}} \mathrm{d}\tilde{x}\, h(\tilde{x}) = \sum_{k=-\infty}^{i} \int_{x_{k-1/2}}^{x_{k+1/2}} \mathrm{d}\tilde{x}\, h(\tilde{x}) = \Delta x_k \sum_{k=-\infty}^{i} f(q_k) \tag{21}$$

[1] The presented description of the ENO algorithm is a compilation of material in these references and unpublished notes by Dr. A. Rogerson.

with $\Delta x_k = x_{k+1/2} - x_{k-1/2}$. The lower limit of the integral is irrelevant and can be replaced by $x_{j_0+1/2}$ for some integer j_0. Since $H(x_{i+1/2}) - H(x_{i-1/2}) = (x_{i+1/2} - x_{i-1/2})\, f(q_i)$, we find that the first divided difference is

$$H^1_{i-1/2} = H[x_{i-1/2}, x_{i+1/2}] \equiv f[q_i] \tag{22}$$

and in general

$$H^k_{i-1/2} = H[x_{i-1/2}, \ldots, x_{i+k-1/2}] = \frac{1}{k} f[q_i, \ldots, q_{i+k-1}], \quad f^k_i = f[q_i, \ldots, q_{i+k}]. \tag{23}$$

Using (14), the first couple of divided differences of f are

$$\begin{aligned} H^2_{i-1/2} = H[x_{i-1/2}, \ldots, x_{i+3/2}] &= \frac{H[x_{i+1/2}, x_{i+3/2}] - H[x_{i-1/2}, x_{i+1/2}]}{x_{i+3/2} - x_{i-1/2}} \\ &= \frac{f[q_{i+1}] - f[q_i]}{x_{i+3/2} - x_{i-1/2}} \qquad (24) \\ &= \frac{1}{2} f[q_i, q_{i+1}] \quad \text{(on an equally spaced grid)} \qquad (25) \\ H^3_{i-1/2} = H[x_{i-1/2}, \ldots, x_{i-5/2}] &= \frac{H[x_{i+1/2}, \ldots, x_{i+5/2}] - H[x_{i-1/2}, \ldots, x_{i+3/2}]}{x_{i-5/2} - x_{i-1/2}} \\ &= \frac{1}{3} f[q_i, q_{i+2}] \quad \text{(on an equally spaced grid)}. \qquad (26) \end{aligned}$$

The utilized portion of

$$H^k_{i-1/2} = H[x_{i-1/2}, \ldots, x_{i-1/2+k}] = \frac{H^{k-1}_{i+1/2} - H^{k-1}_{i-1/2}}{x_{i+k-1/2} - x_{i-1/2}}$$

consists of the stencil of grid points used in calculating a divided difference. The sketch in Fig. 3.1 helps to clarify the stencil involved in each divided difference.

So for every i we need to calculate

$$H^k_{i-\upsilon-1/2}, H^k_{i-\upsilon+1-1/2}, \ldots, H^k_{i+1/2}, \quad \text{for } k = 1, \ldots, Q_1 + 1 \text{ and } \upsilon = k - 1, \tag{27}$$

since for positive Roe speed/eigenvalues we need

$$H^k_{i-\upsilon-1/2}, H^k_{i-\upsilon+1-1/2}, \ldots, H^k_{i-1/2}, \tag{28}$$

and for negative Roe speed/eigenvalues we also need $H^k_{i+1/2}$, that is, at least one upwind point is always involved.

The next step is to obtain $H(x)$ at order Q_1 by a reconstruction via a primitive function technique, for which a Q_1-point stencil is required. Consequently, to obtain $h(x)$ at order Q_1, a $(Q_1 + 1)$-point stencil is needed for its primitive, $H(x)$.

The first point in the stencil is chosen according to the local sign of $f'(q)$ at $x_{i+1/2}$. We may use the Roe speed

$$\bar{a}_{i+1/2} = \frac{f(q_{i+1}) - f(q_i)}{q_{i+1} - q_i} \tag{29}$$

or eigenvalues determined from these Roe speeds for each component of the characteristic equations for a system of partial differential equations, e.g. (9) and (12). In algorithmic form, this so-called ENO-Roe scheme (Shu and Osher 1989) has the form:

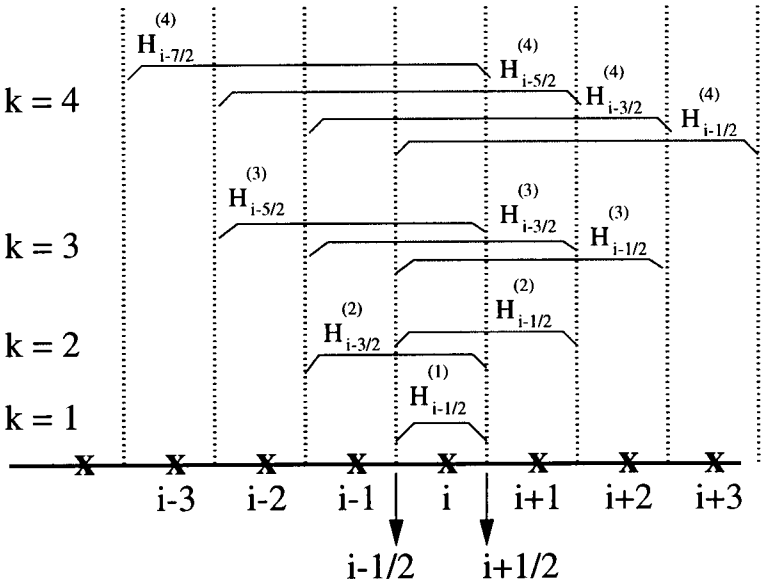


Figure 3.1: The utilized portion of divided differences is defined by their reach, which is sketched up to third order in a space diagram of the grid. Crosses indicate grid points at integer values of i_o and cell edges of grid point i are denoted by dotted lines at half values $i_o \pm 1/2$.

*** Choose first point upwind. ***
if $a_{i+1/2} >= 0$ then $k_{min}^{(1)} = i$;
else $k_{min}^{(1)} = i + 1$;

$H^1(x) = H^{(1)}_{k_{min}^{(1)}-1/2}\,(x - x_{k_{min}^{(1)}-1/2})$

for $k = 2, Q_1 + 1$
*** Smoothest choice gives smallest divided difference. ***
if $|H^k_{k_{min}^{(k-1)}-1/2}| >= |H^k_{k_{min}^{(k-1)}-3/2}|$
then $k_{min}^{(k)} = k_{min}^{(k-1)} - 1$;
else $k_{min}^{(k)} = k_{min}^{(k-1)}$;
$H^{(k)}(x) = H^{(k-1)}(x) + H^k_{k_m^{(k)}in-1/2}\,\Pi_{k=k_{min}^{(k-1)}}^{k_{min}^{(k-1)}+k-1}\,(x - x_{k-1/2})$

It follows that

$$H(x) \approx H^{(Q_1+1)}(x) = H^1_{k_{min}^{(1)}-1/2}\,(x - x_{k_{min}^{(1)}-1/2}) + \sum_{k=2}^{Q_1+1} H^k_{k_{min}^{(k)}-1/2}\,\Pi_{\alpha=0}^{k-1}\,(x - x_{k_{min}^{(k-1)}+\alpha-1/2}), \quad (30)$$

$$h(x) \approx h^{Q_1}(x) = \frac{\partial H^{(Q_1+1)}}{\partial x}$$

$$= H^1_{k_{min}^{(1)}-1/2} + \sum_{k=2}^{Q_1+1} H^k_{k_{min}^{(k)}-1/2} \sum_{\alpha=0}^{k-1} \Pi_{\beta=0(\beta\neq\alpha)}^{k-1}\,(x - x_{k_{min}^{(k-1)}+\beta-1/2}). \quad (31)$$

Shu and Osher (1989) also define such an algorithm for Local Lax Friedrichs (ENO-LLF) and a mixture of ENO-Roe and ENO-LLF. Our ENO codes use the mixed scheme. ENO-LLF is slower but more dissipative and is able to handle entropy-violating expansion shocks. ENO-LLF is a successor of the LLF2 scheme presented next. The mixed scheme takes the best of both worlds. In the ENO programs, the ENO-Roe and ENO-LLF schames are found in the core program in subroutines EnoRoe() and EnoLLF(). Both routines have been simplified for use on a regular grid with the aid of another routine Setcoeff() in which the factors under the multiplication symbol in (31) are calculated.

3.5 Convex Essentially Non-Oscillatory schemes

So far, only a standard second-order local lax-Friedrichs flux (LLF2) has been implemented. Strictly speaking, this is not the convex ENO implmentation by Liu and Osher (1998) but only one of its building blocks. LLF2 consists of the following (Liu and Osher 1998):

$$\begin{aligned} \hat{f}^{LLF2}_{j+1/2} &= \frac{1}{2}\Big(f(q_{j+1}) + f(q_j) - \alpha_{j+1/2}\,(q_{j+1} - q_j)\Big) + \\ &\quad \frac{1}{4}\,m\Big[\Delta_+ f(q_j) + \alpha_{j+1/2}\Delta_+ q_j, \Delta_- f(q_j) + \alpha_{j+1/2}\Delta_- q_j\Big] - \\ &\quad \frac{1}{4}\,m\Big[\Delta_+ f(q_{j+1}) - \alpha_{j+1/2}\Delta_+ q_{j+1}, \Delta_- f(q_{j+1}) - \alpha_{j+1/2}\Delta_- q_{j+1}\Big] \end{aligned} \tag{32}$$

with

$$\Delta_\pm p_j = \pm\,(p_{j\pm1} - p_j) \tag{33}$$

and a canonical minmod wave limiter $m[\cdot,\cdot]$

$$m(x,y) = \left\{ \begin{array}{cc} (\mathrm{sign}\,x)\min(|x|,|y|) & \text{if } x\,y > 0, \\ 0 & \text{otherwise.} \end{array} \right\} \tag{34}$$

For a scalar equation, dissipation $a_{j+1/2}$ is the maximum of $|f'(q)|$ in cell j and $j+1$. For a system the largest eigenvalue (as an upper bound of the largest local eigenvalue) fixes $a_{j+1/2}$ for all components at the cell edge $j+1/2$ between adjacent cells j and $j+1$. Higher-order extensions of convex ENO are found in Liu and Osher (1998). A small modification of the ENO-LLF algorithm in Shu and Osher (1989), as is used in the ENO codes, will yield a faster but more dissipative convex algorithm.

The simplicity of LLF2 and its higher-order convex ENO extensions and the associated computational speed, explains its appeal. None of the time-consuming characteristic decompositions of Shu and Osher's ENO-schemes and other schemes in the literature (e.g. Toro 1990, Leveque 1997) are required. The numerical simulations with the LLF2 scheme will testify that numerical diffusion and dissipation causes shocks to be smoothed more than in the ENO methods.

3.6 Forcing: gravity, geometry and dissipation

The conservative flux terms in the equations of motion have been preserved in the discretization. The remaining terms are either geometric, gravitational or dissipative and are parametrized as functions of the integrated variables $\{m = \rho\,u,\, \rho\}$ and $\{m_a = \rho_a\,u_a,\, \rho_a,\, \tilde{\Omega}\}$ without further spatial discretization. Once the flux term are discretized, we are left with a system of ordinary differential equations (ODE's) of $O(Npxts)$, which

include forcing and discretized conservative terms, and the proposed TVD Runge-Kutta time discretization suffices to integrate these ODE's forward in time (see section 3.2).

For a quiescent fluid, the momentum balance

$$\frac{1}{h_1}\frac{\partial(p\,A)}{\partial\xi_1} - \frac{p}{h_1}\frac{\partial A}{\partial\xi_1} + \frac{\rho\,A}{F_l}\frac{\partial z}{\partial\xi_1} = 0 \tag{35}$$

is not automatically guaranteed to be preserved under the discretization and the fluid may start artificial flow while it should remain at rest. For solid lower boundaries, hydrostatic balance seems to be preserved in the simulations, but open boundaries require more care. At the lower boundary, the characteristic equations for the melt are used. The eigenvalues $u \pm a$ and left and right eigenmatrices are calculated from variables at the last interior grid point. Depending on the sign of the eigenvalue, matter either flows out of the domain, in which case exterior ghost points obtain the values of the last interior grid point, or flows into the domain, in which case the ghost points obtain the initial values of the last interior grid point. (One could place this last interior grid point an infinitesimal distance into the domain.) The above method to preserve hydrostatic balance is reasonably satisfactory, but does create a spurious flow which is fortunately small relative to the rarefaction and shock waves of interest. In a horizontal pipe with varying cross-sectional are A artificial flow of a similar kind can arise. At present, the resolution is chosen high enough to minimize artificial flows to an acceptable level. To the best of my knowledge, no general method has been proposed to avoid these artificial flows, although several options exist. One option is to adjust the last two terms in (35) with higher-order corrections that eliminate the discrepancy, another method of Leveque (1998) is specially designed for quasi-steady flows.

In the LLF2 numerical scheme, a division of the pressure in a hydrostatic one p_h and a perturbation p' with $p = p_h + p'$ allows one to absorb the gravitational term by defining

$$\frac{1}{h_1}\frac{\partial p_h}{\partial\xi_1} = -\frac{\rho}{h_1}\frac{\partial z}{\partial\xi_1}. \tag{36}$$

Rearanging (36) with (1) reveals that pressure p_h can be determined from

$$\int_p^{p_M} \mathrm{d}\tilde{p}\,\frac{1}{\rho(p_h+p'} = -\int_z^{z_M}\mathrm{d}\,\tilde{z} \qquad \Rightarrow \tag{37}$$

$$\ln(p_M/p) - 2\,\epsilon\left(\sqrt{p_M+p-p_h}-\sqrt{p}\right) + \frac{(1-n_0)}{\sigma}\,(p_M-p_h) + \frac{2}{3}\frac{n_0\,\epsilon}{\sigma}\left((p_M+p-p_h)^{3/2}-p^{3/2}\right) + \frac{1}{F_l}\left(z_M-z\right) = 0, \tag{38}$$

in which pressure p is assumed to be known (at grid points) and in which P_M and z_M are judiciously chosen as reference pressure and level. Relation (38) is a cubic in $\sqrt{p_M+p-p_h}$ and only has one or two positive root(s) for certain reference pressures. With $A = 1$ the flux term in the momentum equations simplifies to $m^2/\rho + p'$. This division and absorption of the gravitational term takes care of spurious flows but I have not been able to deal properly with the coupling between melt-volatiles and air at the interface.

The numerical treatment of the parametrized viscous forces appears to be good. It would be desirable to test the numerical model and improve our understanding of discretized boundary conditions by considering the hydraulically-controlled stationary flows in horizontal, vertical and curved conduits. Stationary volcanic flows in vertical conduits obey an ODE and have been solved in (Woods 1995).

Geometric, gravitational and frictional terms are implemented in the core program in function Forcing().

3.7 Boundary conditions

Boundary conditions are implemented in a function called BounCond(). For the moment boundary conditions are set (#define) in the core numerical program which bears the name of the code. A search for #define will quickly locate these definitions near the top of the core program.

3.7.1 Open or extrapolating

Extrapolating boundaries are open boundaries in which interior values are extrapolated in first or fourth order fashion. They are named EXTRAP, EASTEXTRAP and WESTEXTRAP.

3.7.2 Characteristic lower boundary condition

A decomposition of the magma equations into characteristic form reveals which wave components are leaving and entering the domain. The eigenvalues $u \pm a$ in magma at the lower boundary determine whether the corresponding characteristic exits the domain or not. For outgoing waves, exterior ghost points are defined by extrapolating values of the last interior grid point. For incoming waves, exterior grid points are defined by specifying a wave, for example based on the initial value of the interior grid point. In the precense of forcing, gravity and geometric terms, characteristics stay no longer constant on characteristic curves and an adjustment is required. I have not made these adjustments and the current inviscid option works reasonably well. The characteristic boundary routine is implemented under the name WESTCHAR.

3.7.3 Solid boundaries

Solid boundaries are implemented by extrapolating or reflecting the values of density and energy density around the boundary, while taking the opposite value for the momentum density. The last grid point in the fluid has one cell edge coinciding with the boundary. In the function BounCond() solid boundaries are defined as SOLID (domain closed), and EASTSOLID and WESTSOLID.

3.7.4 Interface between melt-volatile fluid and air

Ghost fluid method

Numerical schemes for gas dynamics on Eulerian grids that propagate interfaces between two or more fluids are difficult to construct. Recently, Fedkiw *et al.* (1999) have used a level set approach and a ghost fluid method to model interface dynamics in one and two dimensions. Fedkiw *et al.* (1999) used a level set equation

$$\frac{\partial \Phi}{\partial t} + u\frac{\partial \Phi}{\partial x} + v\frac{\partial \Phi}{\partial y} = 0 \tag{39}$$

to demarcate the interface as the zero level of level set function Φ. Initially, Φ is the signed distance function, which is advected around by (39) using methods in Fedkiw, Merriman and Osher (1999b) and then reinitialized using

$$\frac{\partial \Phi}{\partial t} + S(\Phi)\left(\sqrt{\Phi_x^2 + \Phi_y^2}\right) = 0 \tag{40}$$

to keep Φ approximately equal to the distance function near the interface. $S(\Phi_0)$ is a smoothed sign function of Φ_0. A wave limiter approach Fedkiw *et al.* (1999) prohibits Φ from developing variability at subgrid scales. The level set approach appears to be more complicated than necessary and does not exactly follow

the interface movement. Instead of tracking the level set function, I decided to track the interface contour by particles. In two dimensions, particle advection will have to be filtered to limit subgrid scale motion. Contour advection simulations show that fine-scale Lagrangian structures can be recovered by advection with a coarse-grained Eulerian velocity, so particle advection will be reasonably accurate.

One dimension

In one dimension the interface between melt-volatiles on one and air on the other side is given by a fluid particle at $\xi = \Xi_I(t)$. The Lagrangian dynamics of the particle is governed by

$$\begin{aligned} \frac{d\Xi_I}{dt} &= \frac{u_i}{h_1}, \\ \frac{dU_{in}}{dt} &= -\frac{1}{\rho\, h_1}\frac{\partial p}{\partial \xi}\bigg|_{\xi=\Xi_I(t)} = -\frac{1}{\rho_a\, h_1}\frac{\partial p_a}{\partial \xi}\bigg|_{\xi=\Xi_I(t)}. \end{aligned} \tag{41}$$

Both velocity $U_{in}(t) = u\big(\xi = \Xi_I(t), t\big)$ and pressure are continuous across the interface, while density is generally not.

The dynamics of the compressible fluid on each side the interface is modelled with any of the schemes proposed sofar. ENO fluxes, as we have seen, are calculated by using the smoothest higher-order polynomial in a stencil $[x_{i-r}, x_{i+s}]$ around grid point x_i with $r, s > 0$. Velocity $U_{in}(t)$ is constructed from the smoothest polynomial of the velocity $u(\xi, t)$ around $\xi = \xi_i(t)$ in ENO-Roe fashion. The detailed algorithm is a trivial modification of Shu and Osher's (1989) general algorithm.

*** Choose first grid point closest to interface $\xi_{i_0} < \Xi_I < \xi_{i_0+1}$. ***
*** Construct divided differences $H^{(k)}_{i-1/2}$ for velocity $u(\xi)$. ***
if $|\Xi_I - \xi_{i_0}| <= |\Xi_I - \xi_{i_0+1}|$ then $k^{(1)}_{min} = i_0$
else $k^{(1)}_{min} = i_0 + 1$
$H^{(1)}(x) = H^{(1)}_{k^{(1)}_{min}-1/2}\,(x - x_{k^{(1)}_{min}-1/2})$
for $k = 2, Q_1 + 1$
    if $|H^k_{k^{(k-1)}_{min}-1/2}| >= |H^k_{k^{(k-1)}_{min}-3/2}|$
    then $k^{(k)}_{min} = k^{(k-1)}_{min} - 1$;
    else $k^{(k)}_{min} = k^{(k-1)}_{min}$;
    $H^{(k)}(x) = H^{(k-1)}(x) + H^k_{k^{(k)}_{min}-1/2}\,\Pi_{k=k^{(k-1)}_{min}}^{k^{(k-1)}_{min}+k-1}\,(x - x_{k-1/2})$

It follows that

$$H(x) \approx H^{(Q_1+1)}(x) = H^1_{k^{(1)}_{min}-1/2}\,(x - x_{k^{(1)}_{min}-1/2}) + \sum_{k=2}^{Q_1+1} H^k_{k^{(k)}_{min}-1/2}\,\Pi_{\alpha=0}^{k-1}\,(x - x_{k^{(k-1)}_{min}+\alpha-1/2}), \tag{42}$$

$$\begin{aligned} u(x) \approx u^{Q_1}(x) &= \frac{\partial H^{Q_1+1}}{\partial x} \\ &= H^1_{k^{(1)}_{min}-1/2} + \sum_{k=2}^{Q_1+1} H^k_{k^{(k)}_{min}-1/2} \sum_{\alpha=0}^{k-1} \Pi^{k-1}_{\beta=0(\beta\neq\alpha)}\,(x - x_{k^{(k-1)}_{min}+\beta-1/2}). \end{aligned} \tag{43}$$

Only a limited number of divided differences are required around the interface location, so the storage requirements are small.

The interface dynamics is thus determined by particle advection, but to calculate the fluxes at the cell edges in the fluid on either side of the interface the ghost fluid method of Fedkiw *et al.* (1999a) is used. We will denote the magmatic fluid left of the interface by FL1 and the air to the right by FL2. Since pressure and velocity are generally continuous across the interface, ghost values of the ghost fluid are obtained by equating values of pressure and velocity of the ghost fluid FL1 right of the interface to those of real fluid FL2, and vice versa left of the interface. While the melt ghost fluid FL1 is thus fully determined, the air ghost fluid FL2 is not. The entropy s of the ghost fluid FL2 left of the interface is determined by extrapolating the entropy of FL2 at grid point $i_I + 2$, the second grid point to the right of the interface, to grid points i_I, i_{I-1}, i_{I-2} (see Fig. 3.2). Since $p_a = \kappa(s)\,\rho_a^\gamma$, $\kappa_{I+2} = \kappa(s_{I+2})$ is readily obtained from pressure and density values. Ghost values for energy and air density can thus be found via κ_{I+2}. Fedkiw *et al.* (1999a) call this procedure an isobaric fix, which does prevent local "overheating" and corresponding numerical instabilities. The isobaric fix clearly lacks mathematical rigour —it does not conserve momentum or mass— but works reasonably well. The obvious advantage is simplicity and with local grid refinement the method becomes accurate and remains efficient.

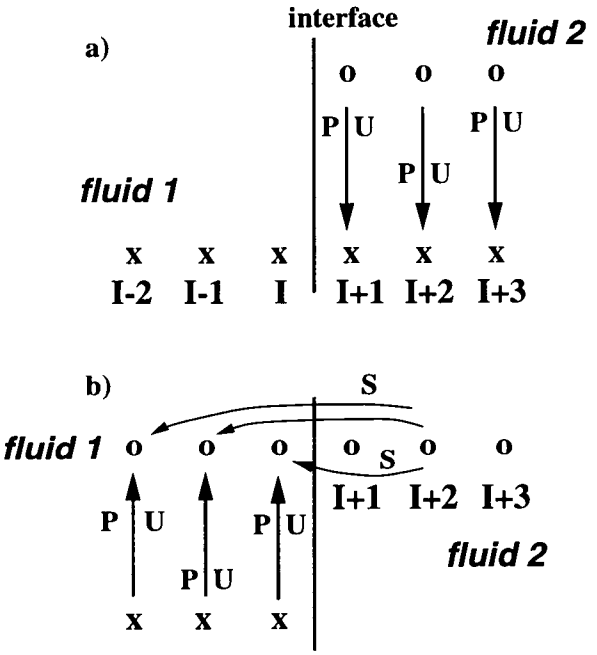


Figure 3.2: (a) By requiring continuity across the interface, pressure P and velocity U of real fluid 2 determine P and U of ghost fluid 1 right of the interface, and vice versa left of the interface. (b) Ghost entropy of fluid 2 (air) is determined by extrapolation of the entropy S of the second grid point $i_I + 2$ in air to grid points left of the interface. This is an isobaric fix. (After Fedkiw *et al.* 1999.)

I have tried to match the LLF2 and ENO schemes with a time-dependent finite-volume treatment near the interface, which conserves mass and momentum, but sofar I have not been able to overcome numerical instabilities. Since the interface is a fluid particle, discretization in a Lagrangian framework appears to be most efficient. An attempt to descretize the equations in a Lagrangian framework was promising, but the large density difference across the interface lead to undesirable large Eulerian grid spacings in air after the regular Lagrangian grid was mapped back into Eulerian space. Lagrangian grid adjustment posed a new set of presently unresolved problems.

ENC	ENO	Characteristics	Exact solutions
enciso1.c	enoiso1.c	melt-volatiles, Eulerian, x-grid	stationary/moving/reflected shock, shock tube
encair1.c	enoair1.c	air, Eulerian, x-grid	stationary/moving/reflected shock, shock tube
encliso1.c	-	melt-volatiles, Lagrangian, a-grid	stationary/moving/reflected shock, shock tube
enclair1.c	-	air, Lagrangian, a-grid	stationary/moving/reflected shocks, shock tube
encmva1.c	enomva1.c	melt-volatiles & air, Eulerian, x-grid	shock tube, resonating/reflected shock
encmax1.c	-	melt-volatiles & air, Eulerian, ξ_1-grid varying cross section & arc	-
encmvs1.c	-	melt-volatiles & air, Eulerian stretched ϕ-grid	reduction to ODE for hydraulic flow
encmvt1.c	-	melt-volatiles & air, Eulerian stretched ϕ-grid	time dependent $A(\xi, t)$

Table 3.1: The various available programs and their characteristics. ENO is Essentially Non-Oscillatory (Shu and Osher 1988, 1989; Shu 1997). ENC is second-order Local Lax-Friedrichs (Liu and Osher 1998).

The interface dynamics is set by #define INTERFACE and is implemented in the core program in a couple of places. The velocity of the interface is determined by ENO-interpolation in the main loops as b[nrk] = Uinterpol(nrk); and the interface position with notation xb[nk] advances in time in the function Runge-Kutta().

3.8 Structure of programs

A series of test programs and final target programs have been developed. Table 3.1 gives an overview of the developed C-programs and their main achievements, and Tables 3.2 and 3.3 list the accompanying Matlab plotting programs and their options for comparison with a few exact solutions. The matlab programs are largely self-explanatory. Test programs generally have two to four versions: there are two Eulerian codes, with LLF2 (e.g. encxxx1) and/or ENO (e.g. enoxxx1) versions, and one or two Lagrangian codes, with LLF2 (e.g. enclxxx1) and/or ENO (enolxxx1) versions.

Each code consists of several files which are compiled and linked together into one single executable. That executable is named after the core program which contains the numerical algorithm. The header file for both LLF2 and ENO codes is iso1.h. For example, the test program with executable enciso1 for a magmatic fluid in one horionzontal dimension consists of five programs:

Plot program	Fluid	Comments
mval.m	magma &air	stack plots of u, ρ, p in dyke-drift system
isot1bfilm.m	magma & air	animations of u, p
isoair1bfilm.m	air, magma & air	animations of u, p
diagn.m	magma & air	plots of largest shocks, particles & cannisters
mval1stend.m	magma & air	initial and final profiles of several runs
diadp.m	magma & air	plots of largest shocks, particles & cannisters of several runs

Table 3.2: Summary of matlab programs for making animations, stack plots for melt-volatiles in curved geometry, and diagnosing the positions in space and time of the largest shocks in air, tracer particles and cannisters.

- enciso1.c is the central core program with function TimeStep() which provides the name of the executable,

- enciso1cpu.c contains the Main() C-function which directs to functions dealing with reading the input file enciso1.in, creating general output files e1ceXXX, e1ctXXX, e1cqXXX, e1ciXXX, e1cdXXX with run number XXX, and the core function TimeStep(),

- enciso1in.c creates the output file given a run number XXX, which is specified in input file enciso1.in,

- enciso1inpu.c reads parameter from the input file enciso1.in, and

- enoiso1ext.c contains external functions, e.g. functions from Numerical Recipes in C which allocate variable arrays, convey error messages and find roots of provided functions.

The central line in the Makefile for code enciso1 reads (see Appendix B)
ENCISO1 = enciso1.o enciso1cpu.o enciso1in.o enciso1inpu.o enoiso1ext.o
and indicates that programs enciso1.c enciso1cpu.c enciso1in.c enciso1inpu.c enoiso1ext.c are used in compiling and linking. Except for the core program, the remaining programs are used in several codes based on the same algorithm, that is LLF2 or ENO. For example, for enoiso1 we find:
ENOISO1 = enoiso1.o enoiso1cpu.o enoiso1in.o enoiso1inp.o enoiso1ext.o.
ENO versions of the LLF2 programs are largely similar:

- enoiso1.c is the central core program which provides the name of the executable,

- enoiso1cpu.c contains the Main() c-function which directs to functions dealing with reading the input file enoiso1.in, creating general output files e1seXXX, e1stXXX, e1sqXXX with run number XXX, and the core function TimeStep(),

- enciso1in.c creates the output file given a run number XXX, which is specified in input file enoiso1.in,

Plot program	Fluid	Exact solution program	Function	Formula
isoair1.m				
	air	airschok.m shocktube	airshocktu.m	$F_{sair}(M)$
	air	reflsair.m moving & reflected shock	-	-
	magma & air	shtumeltair.m shocktube	fndcp.m preshtmeltair.m	$F_{ndcp}(p)$ $F_{pmva}(M)$
isot1.m				
	magma	shtuiso.m shocktube	preshtiso.m	$F_{piso}(p)$
	magma	rflshiso.m moving & reflected shock	rflisov3.m	$F_{mriso}(p)$
	magma		densiso.m	$\rho(p)$
	magma		soundiso.m	$a(\rho) = a(p)$

Table 3.3: Programs isot1.m and isoair.m are matlab plotting programs for magma only, air only, and magma and air together, respectively. Various test simulations have been used in which numerical results are compared with exact solutions. Most of these exact solutions are implicit and require some numerical evaluation; the miriad of Matlab-programs that perform this task are presented in this table.

- enciso1inpu.c reads parameter from the input file enoiso1.in, and

- enoiso1ext.c see above.

Observe that the ENO input file and output files have different names. After this introduction, the Makefile should be self-explanatory. The Makefile is included in Appendix B.

The functions referred to in Table 3.3 are:

$$F_{sair} = \sqrt{\gamma}\,\kappa^{1/(2\gamma)}\,p_1^{(\gamma-1)/(2\gamma)}\left(1+\frac{2\gamma\,(M^2-1)}{\gamma+1}\right)^{(\gamma-1)/(2\gamma)} + \frac{a_1\,(M^2-1)}{(\gamma+1)\,M} - \frac{a_4}{\gamma-1}, \quad (44)$$

$$F_{ndcp} = \frac{1}{\rho^2}\left(\frac{\partial p}{\partial \rho}\right)^2 = \sqrt{\frac{1-\epsilon\sqrt{p}}{p^2} + \frac{\epsilon}{2\,p\sqrt{p}} - \frac{\epsilon\,n_0}{2\sqrt{p}}}, \quad (45)$$

$$F_{pmva}(M) = u_2 + \int_{p_1}^{p_2} d\tilde{p}\,F_{ndcp}(\tilde{p}) - \int_{p_1}^{p_4} d\tilde{p}\,F_{ndcp}(\tilde{p}), \quad (46)$$

$$p_2 = p_1\left(1+2\,\gamma\,\frac{(M^2-1)}{(\gamma+1)}\right), \quad (47)$$

$$u_2 = 2\,a_1\,\frac{(M^2-1)}{(\gamma+1)\,M}, \quad (48)$$

$$F_{piso}(p) = \sqrt{\frac{(p-p_1)\,(\rho(p)-\rho(p_1))}{\rho(p)\,\rho(p_1)}} + \int_{p_1}^{p} d\tilde{p}\,F_{ndcp}(\tilde{p}) - \int_{p_1}^{p_4} d\tilde{p}\,F_{ndcp}(\tilde{p}), \quad (49)$$

$$F_{mriso}(p) = p - p_2 - \left(1+\frac{\rho(p_2)}{\rho(p)-\rho(p_2)}\right)\rho(p_2)\,u_2^2, \quad (50)$$

$$u_2 = \left(1-\frac{\rho(p_1)}{\rho(p_2)}\right)U_i, \quad (51)$$

$$U_i = \sqrt{\frac{\rho(p_2)\,(p_2-p_1)}{\rho(p_1)\,(\rho(p_2)-\rho(p_1))}}. \quad (52)$$

There are two Matlab-programs that create animations from given input data: isot1bfilm.m deals with data from magmatic fluids generated in the enc/eno/encl/enoliso1 programs; and isoairbfilm deals with data from simulations with air only and magma and air together generated by programs enc/encl/eno/enolair1 and enc/enomval, encmax1, encmvs1 and encmvt1, respectively.

3.8.1 Input file

The structure of the inputfile enciso1.in or enoiso1.in is partly fixed and partly flexible. After the run number runno is specified, see the input files below, a coefficient Nrcoef specifies how many coefficients will be defined. The first eight 0–7 are necessary to run a basic doe, but the remaining coefficients, if any, can be used to specify application specific coefficients. Program encmvs1.c has 29 of these coefficients, while program encmvt1.c has 37 coefficients. It is straighforward to add more, but coefficients can only be taken away once they are cleared in the program.

For example, the input file for the program enciso1.c with executable enciso1 for the stationary shock in magma presented in section 4.1 is:

Variable	Value	Comment
Nor:	3	only for ENO: spatial order of scheme
Nrk:	3	only for ENO: time order of scheme
Nt:	500	number of time steps
Nmeas:	50	cycle for measurements
Npxts:	100	number of grid points
Nsys:	3	number of variables
dt:	0.05	time step
cfl:	0.1	cfl condition
Lx:	0.5	scaled length of domain
c0:	1.0	(not in use)
icno:	0	initial condition choice number
runno:	000	three-digit run number to label output files
Nrcoef:	7	number of coefficients to read
n0:	0.02	volatile content of magma
Rv:	462	gas constant of magma
T:	1000	isothermal temperature magma
sigma:	2500	melt density
s:	3.e06	Henry's constant
Ra:	287.04	gas constant of air
dampn1:	0.e0	damping (not in use)
rho0:	100.0	scaling factor for density

* stationary shock simulation enciso1.c 000 *

Together with the timestep $\Delta t = cfl\,\Delta x$, the number of timesteps determines the final simulation time in case a variable timestep is used. Codes encmvs1.c & encmvt1.c uses a variable time step based on the maximum speed present.

The input file for the program encmvt1.c with executable encmvt1 for cannister movement in a dyke-drift system with magma and air (section 5.3) differs from the previous example:

Variable	Value	Comment
⋮	⋮	
Lx:	20	scaled length domain
⋮	⋮	
Nrcoef:	36	number of coefficients to read
⋮	⋮	
7-rho0:	100.0	scaling factor for density ($kg\,m^{-3}$)
8-RADIUSARC:	1.0e0	scaled radius of arc/void
9-ATUNNEL:	0.7854e0	scaled area of drift
10-ADIKE:	3.2e0	scaled area of dyke
11-Nudamp:	0.e0	damping factor (void)
12-1/Fl:	0.0054e0	inverse Froude number
13-pmagma:	2.e07	initial dyle tip pressure (Pa)
14-Ldike:	215.e0	scaled length dyke
15-ptun:	1.e05	pressure in drift (Pa)
16-gamma:	1.4	γ
17-mu0/L:	2.e0	μ_o/L
18-lambda:	0.0025	turbulent friction factor
19-alpha:	0.7e0	critical vesicularity (%)
20-diametertunnel:	1.0e0	scaled diameter d of drift
21-Amin:	0.2e0	not in use
22-widthdike:	0.2e0	scaled width w dyke
23-Ltunnel:	40.e0	scaled L_{drift}
24-alphae:	0.25e0	geometric factor for coarse grid in dyke (5.1)
25-dlr:	40.e0	scaled factor ΔL_r for coarse grid
26-Lr-dlr:	40.e0	scaled factor L_r for coarse grid
27-pshock:	3.e05	not in use
28-noparticles:	33	number of released tracer particles
29-timepastart:	9.6154e0	scaled time particles start advection
30-ramptimet0:	19.2308e0	scaled ramp time for time-dependent nozzle
31-Lcan/L:	1.4e0	scaled, L_{can}/L
32-rcan/L:	0.2e0	scaled, r_{can}/L
33-rhocan/rho0:	110e0	scaled, ρ_{can}/ρ_0
34-staticfr:	1.0	static friction λ_s
35-dynamicfr:	0.5	dynamics friction λ_d
36-CD:	1.0	drag coefficient C_D

* stationary shock simulation enciso1.c 085 *

Note that $c0$ has been changed to $N0$. $N0$ is the amount of time steps between special measurements, just like $Nmeas$; for variable timestep it defines the time between special measurements.

The input files enciso1.in and enoiso1.in contain all the runs for testing the various programs in this report as well as most of the runs in Bokhove (1999). Only the top lines of these inut files will be read. With a simple cut and paste action, we can choose the input file we like or modify an existing one. Appendices A and C contain instructions on installing and using the programs.

3.8.2 Core program

The core program is the most important program. It contains the coded version of the numerical scheme, while all the other programs take care of initial IO or contain support functions. The boundary conditions are # defined in the top lines of the program. One has to set these boundary conditions in the program and then compile.

Several of the most important functions of the core program are:

- Function TimeStep(Tskeuze) is the main routine with the time and Runge-Kutta loop.

- Function Initial-Condition(Tk, Nii, tc) defines the initial conditions. The domain length and grid points are calculated, and flexible array and matrix lengths are defined. Different choices of initial condition are specified by the input parameter icno. One can either use the initial conditions available, just browse through and read the comments in the program, or create one.

- Function Measure(nnt, time, Ni) measures profiles of the key variables after Nmeas timesteps or a time equivalent —in case variable time steps are used.

- Boundary conditions are dealt with in function BounCond(nrk).

- Functions Fluxiso(nrk) and Fluxeul(nrk)) calculate the conservative terms in the equations. In the older programs for magma simulations this is sometimes done within the time and Runge-Kutta loop.

- Function Runge-Kutta(nk) takes care of time stepping and of cannister movements in program encmvt1.c.

- All terms other than fluxes are considered in function Forcing(nrk, Niz).

- Functions Uinterpol(nrk), Updateface(0), and Diagnos(nnt, time) only appear in magma-air programs. Function Uinterpol(nrk) calculates the velocity at the interface using ENO interpolations; Updateface(0) updates the interface, magma, and air indices after each time step; and function Diagnos(nnt, time) takes care of specialized measurements at (shorter) cycles after N0 timesteps (or a time interval equivalent).

- Particle advection in encmvt1.c is done in function Particles-Forward(nrk).

A flow chart of the algorithm is given in Fig. 3.3.

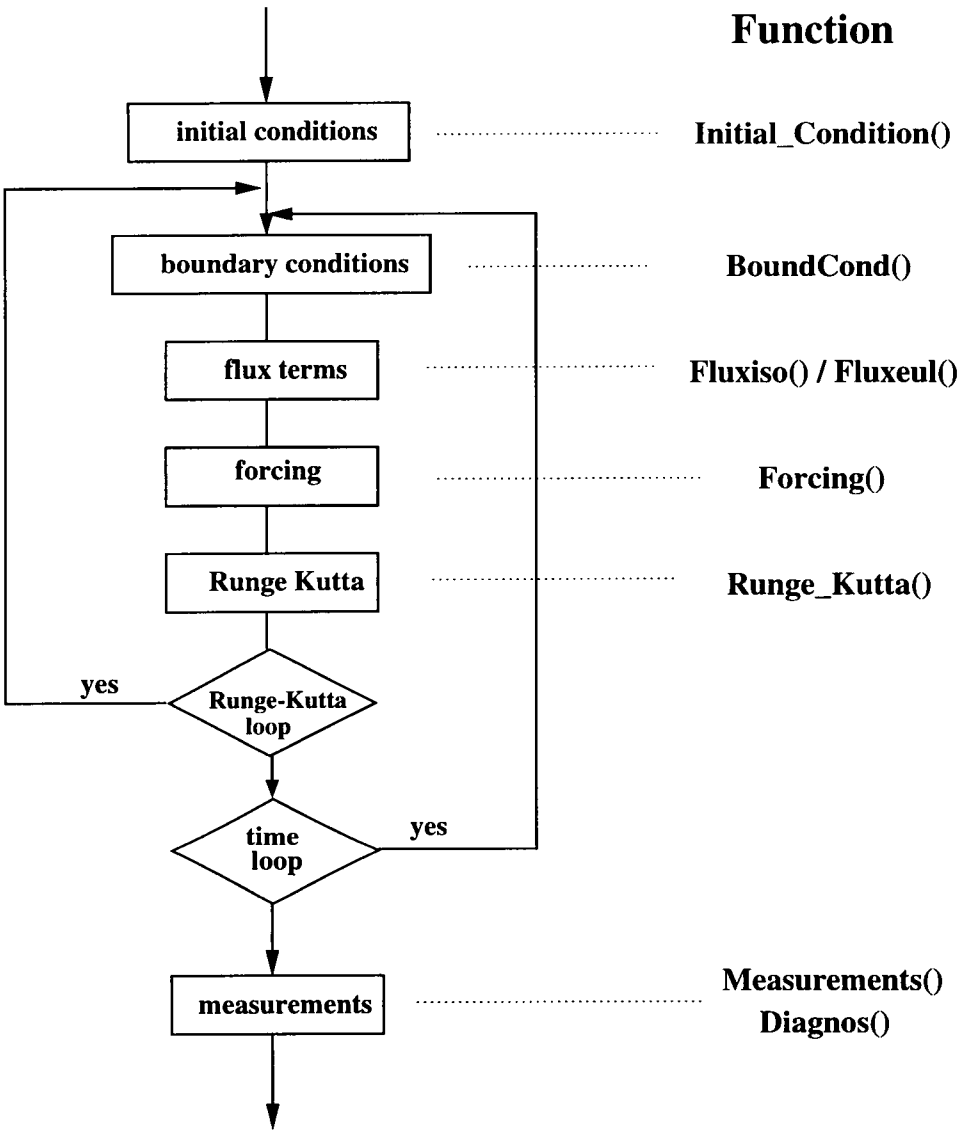


Figure 3.3: Flow chart and the associated functions in core function TimeStep().

Chapter 4

Numerical tests

To develop and test the numerical codes, the basaltic fluid flows (7) and air flows (16) have been considered in separation before the interface dynamics was added. In all these three cases, for the two separate fluids and the two fluids together, specific simulations in one-dimension are compared with exact (implicit) solutions of stationary shocks, moving and reflected shocks, and shock tube solutions. A summary of the numerical tests in this section is found in Table 4.1.

4.1 Dynamics of magma

4.1.1 Stationary shock

The stationary shock solution of (10) (with scale factor $h_1 = 1, \xi_1 = x, g = \nu = 0$) follows directly from momentum density and mass conservation across the discontinuity:

$$[\rho v] = 0, \qquad [\rho v^2 + p] = 0, \tag{1}$$

where $v \equiv u - U$ is the speed relative to the moving shock with speed U, and where $[m]$ denotes the jump in quantity m across the discontinuity. If the pressures left and right of a stationary shock are given by p_2 and p_1 with left state 2 and right state 1, then the velocities are $u_2 = \rho_1 u_1/\rho_2$ and

$$u_1 = \sqrt{\frac{\rho_2}{\rho_1}\frac{(p_2 - p_1)}{(\rho_2 - \rho_1)}},$$

respectively. We start with values $p_2 = 5 * 10^6\, Pa$ and $p_1 = 10^5\, Pa$ and a discontinuity at $x = 1.25\, m$.

The stack plots in Fig. 4.2 and hereafter should be interpreted as follows. The initial profile of density or velocity is plotted versus space. Subsequent profiles of density and velocity, arising after subsequent time-intervals, are plotted in the same graph but the value on the vertical axis is shifted upward each time by a specified amount; in other words it is "stacked". Stack plots thus become density or velocity plots versus space and time. Sometimes the vertical axis is denoted as time axis but I prefer to display the amplitude of the variable.

The LLF2 simulation is seen, in the stack plots of Fig. 4.2a), to shed an acoustic wave before it adjusts to a stationary numerical shock solution, while ENO appears to be almost perfect (see Fig. 4.2b)) because the steepness of the wave is determined by one grid spacing only and because no adjustment occurs. For shocks moving into still fluid ahead of the shock both ENO and CENO schemes show an acoustic wave adjustment.

Program	Description & icno	Run #	CFL	Δx	$Npxts$	T_{end}	T_{sim}
enciso1.c	stationary shock 0	e1c*000	0.1	0.05	100	0.25	s
enciso1.c	shock tube 1	e1c*001	0.1	0.03	400	3.3	s
enciso1.c	shock tube & wall 2	e1c*002	0.1	0.03	400	6	4 : 44
enciso1.c	moving & reflected shock 3	e1c*003	0.1	0.5/400	400	0.5	s
enoiso1.c	stationary shock 0	e1s*000	0.1	0.05	100	0.25	s
enoiso1.c	shock tube 1	e1s*001	0.1	0.03	400	3.3	2 : 37
enoiso1.c	shock tube & wall 2	e1s*002	0.1	0.03	400	6	4 : 44
enoiso1.c	moving & reflected shock 3	e1s*003	0.1	0.5/400	400	0.5	9 : 17
encair1.c	stationary shock 0	e1c*010	0.1	0.05	100	0.2	s
encair1.c	shock tube 1	e1c*011	0.1	0.015	800	1.2	s
encair1.c	moving & reflected shock 2	e1c*012	0.1	0.5/400	400	0.1875	s
enoair1.c	stationary shock 0	e1s*010	0.1	0.05	100	0.2	s
enoair1.c	shock tube 1	e1s*011	0.1	0.015	800	1.2	4 : 15
enoair1.c	moving & reflected shock 2	e1s*012	0.1	0.5/400	400	0.1875	3 : 57
encmva1.c	shock tube 1	e1c*020	0.05	0.03	400	1.68	-
encmva1.c	shock tube 1	e1c*021	0.05	0.012	1000	1.68	5 : 58
enomva1.c	shock tube 1	e1s*020	0.05	0.03	400	1.68	7 : 23
enomva1.c	shock tube 1	e1s*021	0.05	0.012	1000	1.68	50 : 28
encmva1.c	resonating shocks 3	e1c*030	0.05	0.012	1000	15	41 : 44
encmvs1.c	magma-repository 3	e1c*056	0.05	-	3000	21.91	113 : 7.8
encmvs1.c	magma-repository 3	e1c*060	0.05	-	3000	219.03	842 : 71.64
encmvt1.c	magma-repository 3 (cannister)	e1c*091	0.05	-	6000	39.43	208 : 25.6

Table 4.1: Summary of numerical tests. Most of the basaltic runs, air runs, and melt-volatile-air runs have been scaled with $P0 = 924000\,Pa$, $L = 5\,m$, $\rho_0 = 100\,kg\,m^{-3}$, $T_0 = L/U_0 = 0.052\,s$. T_{end} refers to dimensionless time units. Computational times concern computation on a Sun Ultrasparc 2200 with two CPU's running at $200\,MHz$ for all but the encmvs1 and encmvt1 programs, which were run on a PC with Pentium-III processor. Files e1c*XXX refer to LLF2 output and files e1s*XXX refer to ENO output.

The acoustic adjustments leads to a decrease anc increase in energy for the LLF2 simulation, while energy is constant for ENO (Fig. 4.1).

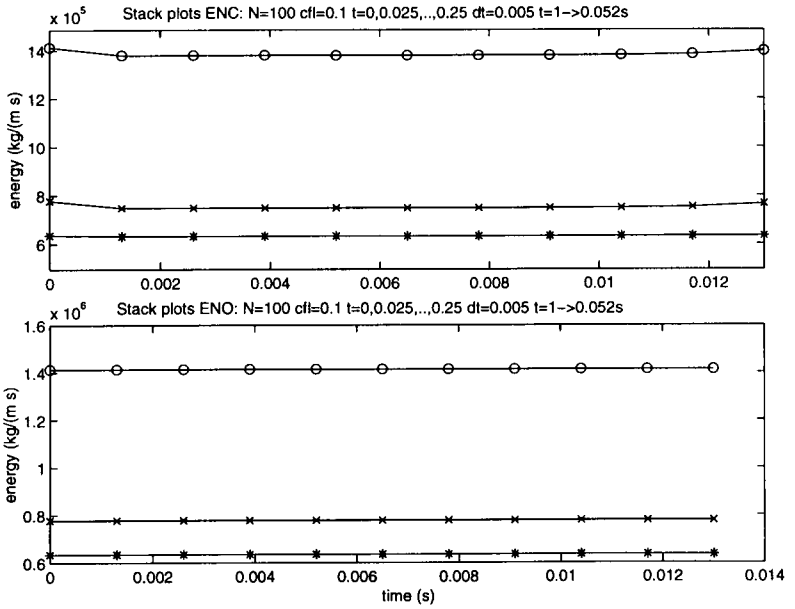


Figure 4.1: Kinetic (*), potential (x), and total (o) energy in a simulation of a stationary shock in melt-volatile fluid: (a) LLF2 code, and (b) ENO code.

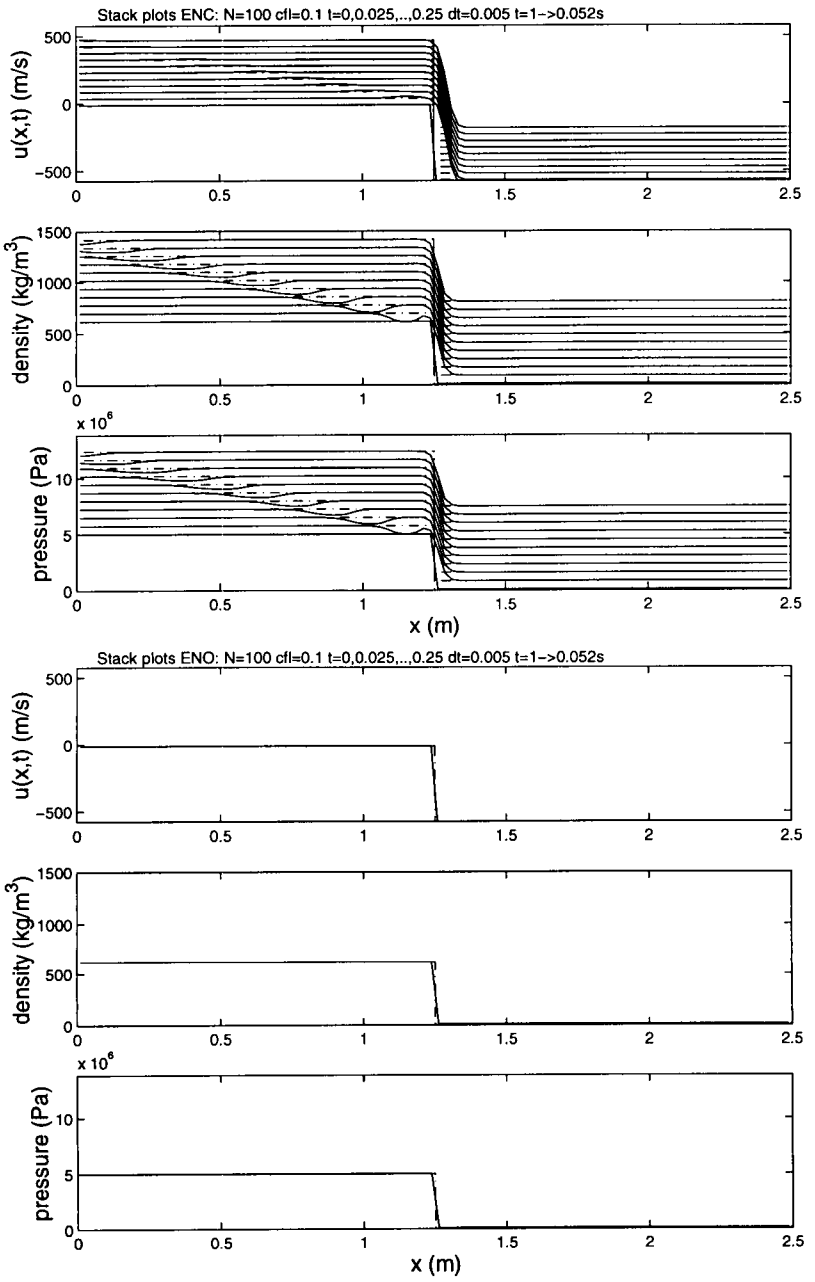


Figure 4.2: Velocity, density and pressure profiles against space are shown for a stationary shock of a melt-volatile fluid: (a) LLF2 code, stack plots with offset $0.8\,P_0 = 7.4\,10^5\,Pa$ in p, $0.8\,\rho_0 = 80\,kg\,m^{-3}$ in ρ and $0.5\,U = 48.06\,m/s$ in u, lenght scale $L = 5\,m$ and time spacing is $0.025\,s$, and (b) ENO code. Dashed-dotted lines indicate the exact and solid lines the numerical solution.

4.1.2 Moving and reflected shock

For a shock moving into quiescent fluid on the right in state 1, shock relations (1) give

$$v_1 = U_i = \sqrt{\frac{\rho_2}{\rho_1}\frac{p_2 - p_1)}{(\rho_2 - \rho_1)}}, \qquad u_2 = (1 - \rho_1/\rho_2)\, U_i \tag{2}$$

for $p_2 > p_1$. After reflection, the region 3 between the reflected shock and the walls is at rest. Evaluation of the shock relations (1) with left state 2 and right state 3 gives the implicit relation

$$p_3 - p_2 - \frac{(p_2 - p_1)\,(\rho_2 - \rho_1)}{\rho_1}\left(1 + \frac{\rho_2}{(\rho(p_3) - \rho_2)}\right), \tag{3}$$

in terms of the initially known pressures p_1 and p_2 which determine —note that $\rho_{1,2} \equiv \rho(p_{1,2})$. Once p_3 is known, density ρ_3 follows as $\rho(p_3)$ and the reflected shock speed is negative, i.e.

$$U_r = -\frac{\rho_2\, u_2}{(p_3 - \rho_2)}. \tag{4}$$

The strength of the shock reflection is given by the ratio of pressure jumps of reflected over incident shock $S_{rm} = (p_3 - p_2)/(p_2 - p_1)$ which is a function of p_1 and p_2. S_{rm} is plotted versus p_1 and p_2 in Fig. 4.3 and depends on p_1 and p_2 in a nonlinear fashion. Moreover, the magnitude of the reflected shock is always higher, sometimes significantly so, than that of the incoming shock.

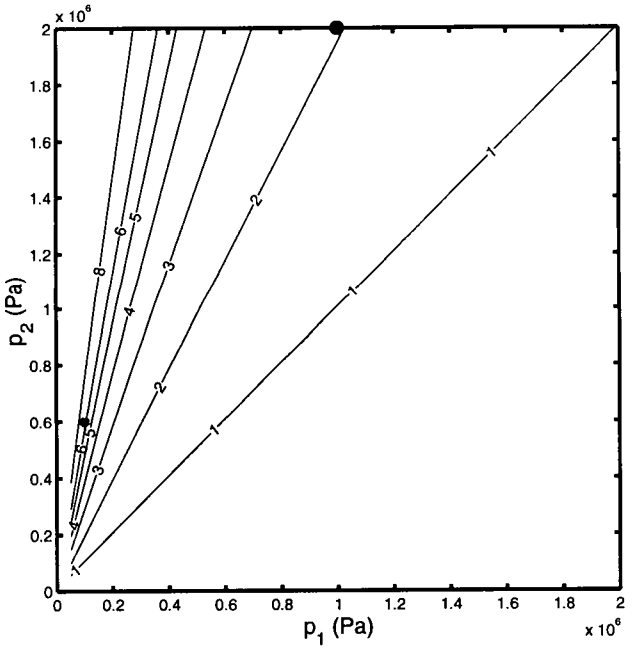


Figure 4.3: Contourplot of S_{rm} versus p_1, p_2 for $p_2 > p_1$.

The simulations shown in Fig. 4.4 and Fig. 4.5 compare well with the exact solutions. The ENO simulation capturs the moving and reflected shock better than the LLF2 simulation.

4.1.3 Shock tube with and without tunnel end

The shock tube solution of (10) (with scale factor $h_1 = 1, \xi_1 = x, g = \nu = 0$ and) with pressure $5 * 10^6\, Pa$ on the left and $10^5\, Pa$ on the right is obtained by matching a shock to a backward propagating rarefaction

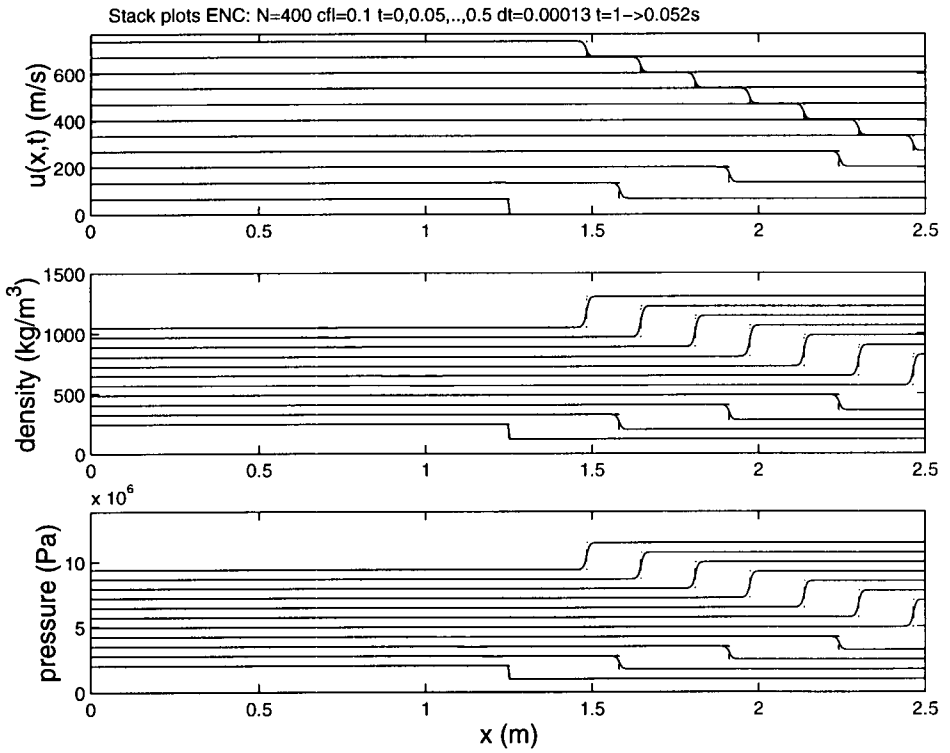


Figure 4.4: Velocity, density and pressure profiles are shown for a moving and reflected shock of a melt-volatile fluid: (a) LLF code, and (b) ENO code. Dashed-dotted lines indicate the exact and solid lines the numerical solution. (Stack plots with offset $0.8\,P_0 = 7.4\,10^5\,Pa$ in p, $0.8\,\rho_0 = 80\,kg\,m^{-3}$ in ρ and $0.7\,U = 67.29\,m/s$ in u. Lenght scale $L = 5\,m$. Time spacing is $0.5\,s$.)

wave (similar calculations are found in Whitham 1974).

The initial condition for a shock tube consists of a quiescent state with pressure p_4 (on the left) and p_1 on the right ($p_4 > p_1$). The space-time diagram Fig. 4.6 explains the evolution: the left state 4, for $x < a_4\,t$, is connected via a rarefaction, for $-a_4\,t < x < (u_3 - a_3)\,t$ to a constant region 3, for $(u_3 - a_3)\,t < x < U\,t$, behind the shock. After the shock $x > U\,t$ and the speed of sound is $a(p)$, so $a_3 = a(p_3)$. From the shock relations (1) we find

$$u_3 = \sqrt{\frac{(p_3 - p_1)\,(\rho_3 - \rho_1)}{\rho_1\,\rho_3}}, \tag{5}$$

while the rarefaction wave is a simple wave. The C_+-characteristic is constant on $dx/dt = u + a$: $u + c(p) = c(p_4)$ and

$$c(p) = \int^p \mathrm{d}\tilde{p}\,\frac{1}{\rho(\tilde{p})}\left(\frac{\partial\rho(\tilde{p})}{\partial\tilde{p}}\right)^{1/2}.$$

Matching this constant characteristic to the constant region 3 gives an implicit relation:

$$u_3 + c(p_3) - c(p_4) = \sqrt{\frac{(p_3 - p_1)\,(\rho_3 - \rho_1)}{\rho_1\,\rho_3}} + c(p_3) - c(p_4) = 0 \tag{6}$$

from which p_3 can be determined, and hence u_3 follows from (5). For the simple expansion wave we have

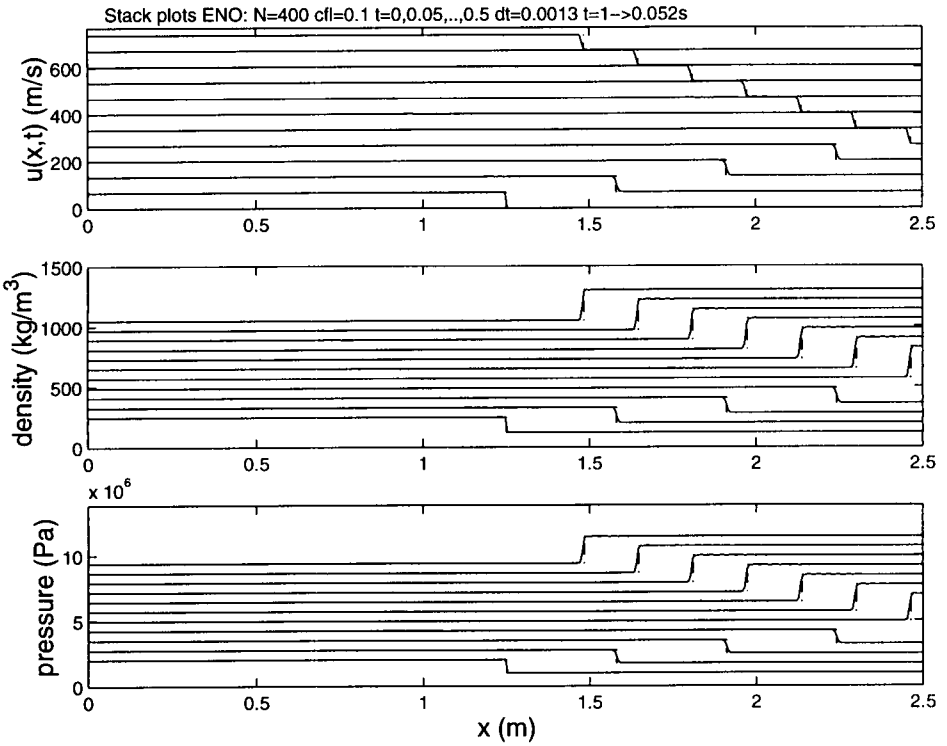


Figure 4.4: Continued

$dx/dt = x/t = u - a(p)$ and with $u + c = c(p_4)$ we find $x = (c(p_4) - c(p) - a(p))\,t$ and $u = c(p_4) - c(p)$ for $p_3 < p < p_4$.

In the following simulations $p_1 = 10^5\,Pa$ and $p_4 = 5 * 10^6\,Pa$. The evolution of velocity, density and pressure are given in Fig. 4.7 for the exact solution (dashed-dotted lines) and the numerical solution (solid lines). The rarefaction wave is travelling to the left, moving away from the discontinuous initial condition. This smooth rarefaction wave connects to a forward propagating shock wave, i.e. to the right, which is apparent in the numerical solution as a steep pressure change of about $5 * 10^5\,Pa$. The steepness of the shock wave is limited by the grid size and the numerical scheme. In particular, the ENO simulation resolves the shock better than the LLF2 simulation which has smoothed the shock stronger. Energy is dissipated in the shock and more energy is lost after the shock leaves the domain around $t = 0.14\,s$ (see Fig. 4.8).

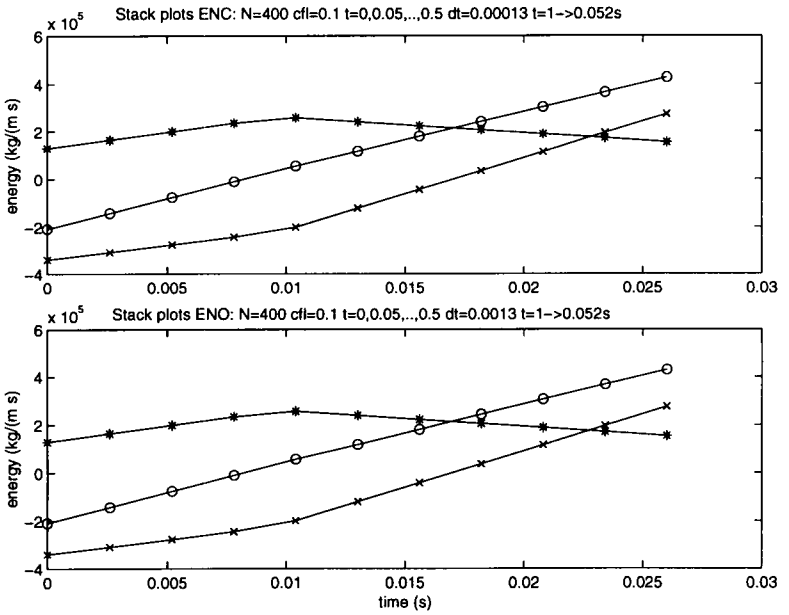


Figure 4.5: Kinetic (*), potential (x), and total (o) energy in simulation of moving and reflected shock in a melt-volatile fluid: (a) LLF2 code and (b) ENO code.

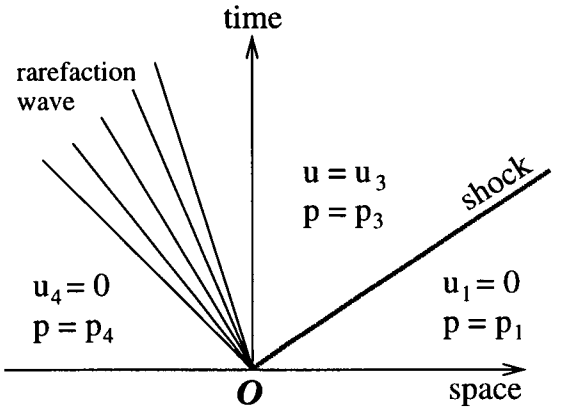


Figure 4.6: Space-time diagram of shock tube flow evolution for magma.

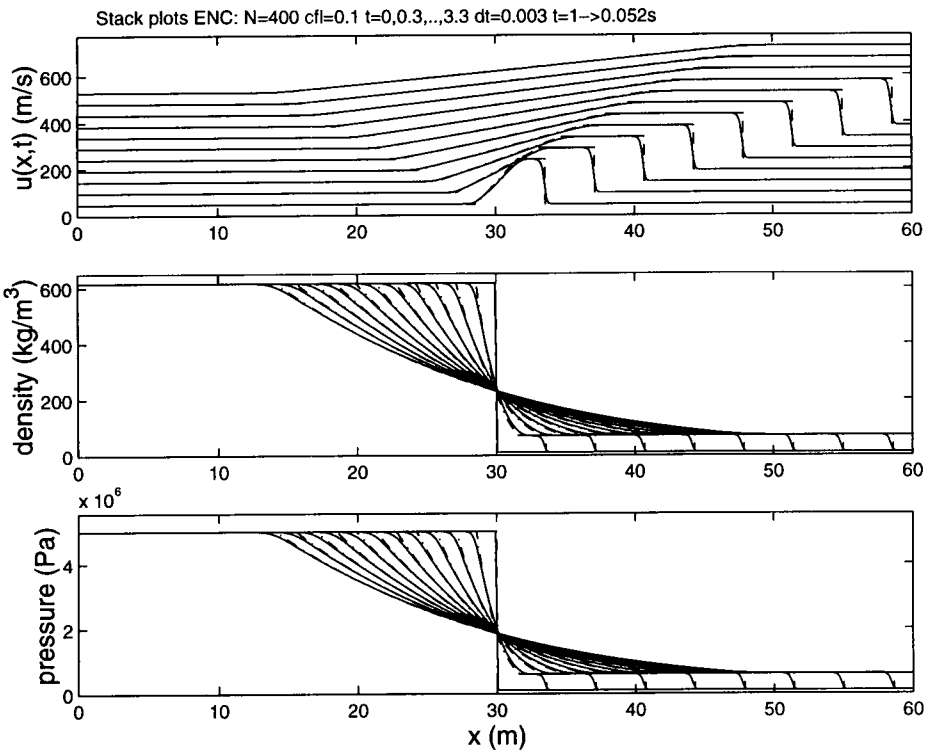


Figure 4.7: Shock tube tests for melt-volatile flow with open boundary conditions. Velocity, density, and pressure stack plots are shown for (a) a LLF2 code and (b) an ENO code. Dashed-dotted lines indicate exact and solid lines numerical solutions. Stack plots have offset $0.5\,U = 48.0625\,m/s$ in u. $L = 5\,m$. Time spacing 0.3.

When a wall is added at $x = 60\,m$ the incoming shock in the shock tube reflects and leads to a reflected shock which is much larger (Fig. 4.9) than one might expect from the moving and reflected shock calculations in Fig. 4.4. The difference between the two shock reflection problems is that the rarefaction wave feeds into the reflected shock and, more importantly, that the pressures before (p_2) and after (p_1) the incoming shock are an order of magnitude lower. According to (3) or the lower left cross in Fig. 4.3 the reflected shock has a pressure $p_3 = 3.74 * 10^6\,Pa$ for $p_1 \approx 10^5\,Pa$ and $p_2 \approx 6 * 10^5\,Pa$, giving $S_{rm} \approx 6$. This is in agreement with the simulations and should be contrasted with the calculations in Fig. 4.4 where for $p_1 = 10^6\,Pa$ and $p_2 = 2 * 10^6\,Pa$ the reflected shock has $p_3 = 4.056 * 10^6\,Pa$, giving a shock reflection amplitude $S_{rm} \approx 2$ that is in agreement with Fig. 4.3 —marked by the upper solid circle. The energy dissipation in the reflected shock is much stronger than the dissipation in the incoming one, as is evident from the change in slope at $t \approx 0.14\,s$ in Fig. 4.10. The ENO code gives sharper shock resolution but is in the reflection process not devoid of small-scale oscillations. The large shock amplification in reflection is expected to be generic and largely independent of the detailed equation of state.

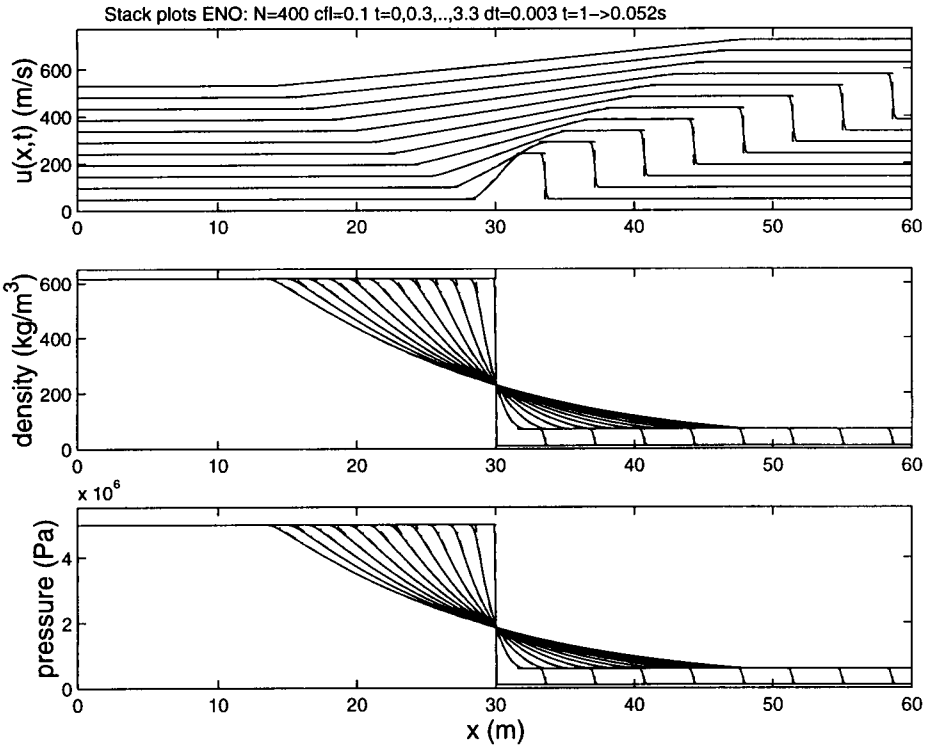


Figure 4.7: Continued.

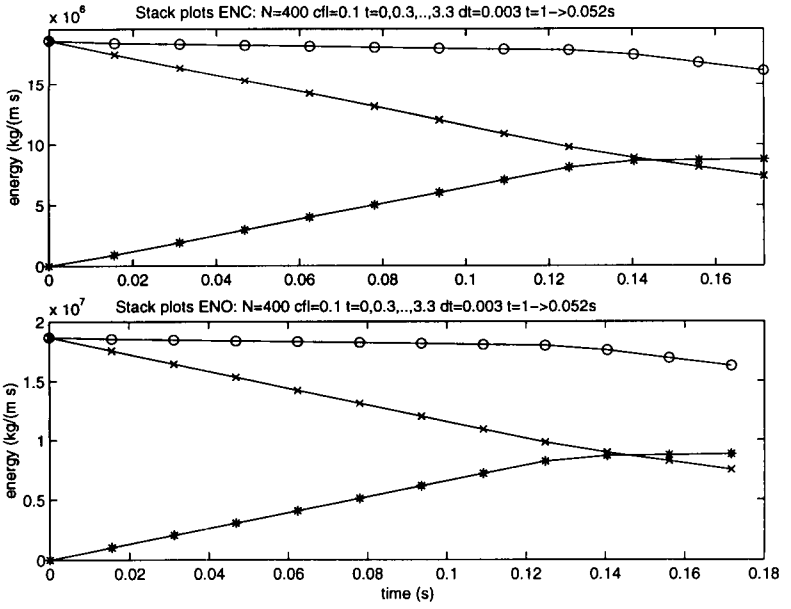


Figure 4.8: Shock tube tests for melt-volatile flow with open boundary conditions. Kinetic (*), potential (x) and total energy (o) are shown for a) a LLF2 code and (b) an ENO code.

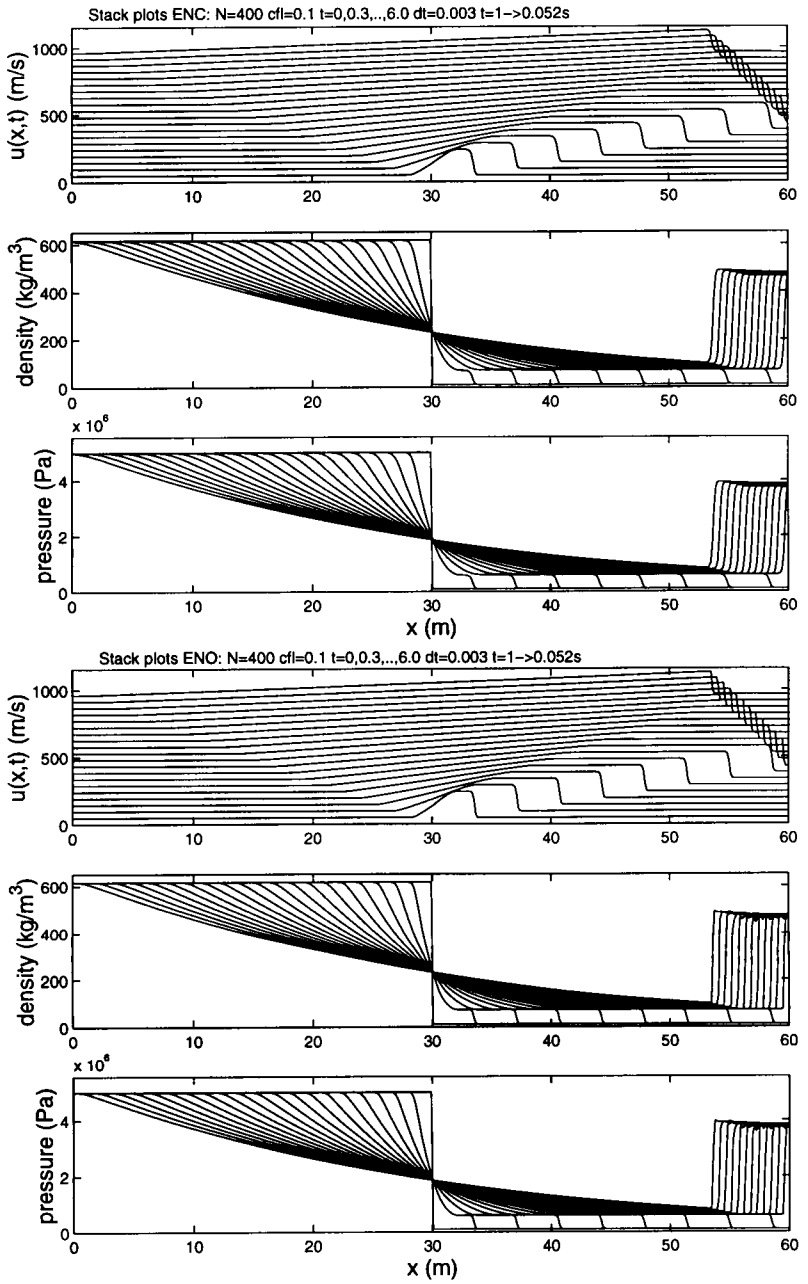


Figure 4.9: Shock tube tests for melt-volatile flow with a right wall. Velocity, density, and pressure stack plots are shown for (a) a LLF2 code and (b) an ENO code. Stack plots have offset $0.5\,U = 48.0625\,m/s$ in u. $L = 5\,m$. Time spacing 0.3.

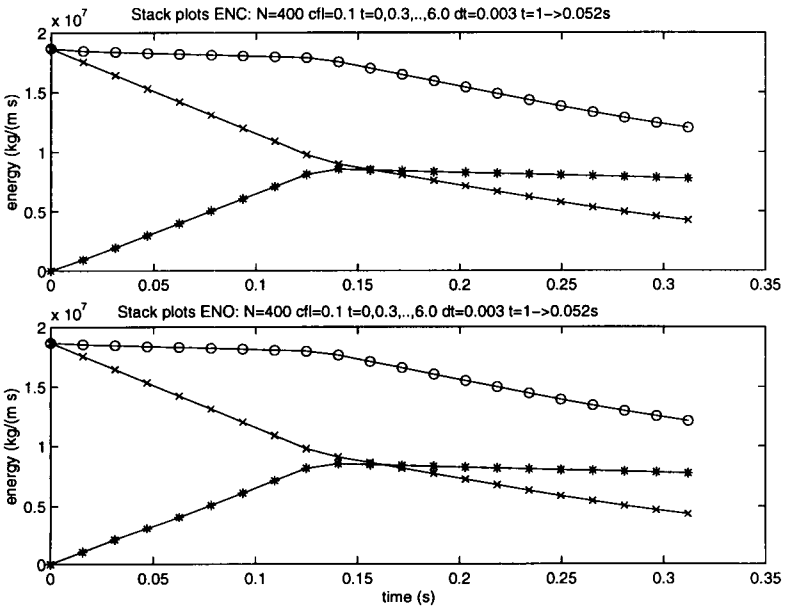


Figure 4.10: Shock tube tests for melt-volatile flow with a right wall. Kinetic (*), potential (x) and total energy (o) are shown for a) a LLF2 code and (b) an ENO code.

4.2 Dynamics in air

The shock relations for air are (Whitham 1974)

$$[\rho_a \, v_a] \;=\; 0, \qquad [p_a + \rho_a \, v_a^2] = 0, \qquad [\rho_a \, v_a \, (h + \frac{1}{2} v^2)] = 0 \tag{7}$$

with $h = e + p_a/\rho_a = (\gamma/(\gamma-1))\,(p_a/\rho_a)$ and $v_a \equiv u_a - U$. We will drop the subscript "a" as long as no confusion arises.

4.2.1 Stationary shock

Given constant pressure, density and velocity in left state 2 and stationarity of the shock, the right state 1 is found from (7):

$$u_1 = \frac{2\gamma p_2 + (\gamma-1)\,\rho_2 \, u_2^2}{(\gamma+1)\,\rho_2 \, u_2}, \qquad p_1 = p_2 + \rho_2 \, u_2^2 - \rho_2 \, u_2 \, u_1, \qquad \rho_1 = \frac{\rho_2 \, u_2}{u_1}. \tag{8}$$

For an initial condition with $p_2 = 1.8\,10^5\,Pa$, $\rho_2 = p_2/(\rho_0 \, T_{ia})$, $T_{ia} = 293\,K$, $u_2 = -0.8\,c_{air}$ and sound speed $c_{air} = \sqrt{\gamma p/\rho}$, the LLF2 and ENO simulations correspond well with the exact solution, as is evident from Fig. 4.11. The ENO simulation is again perfect, while the shock in the LLF2 simulation is less well resolved and the energy grows slightly (not visible in Fig. 4.12).

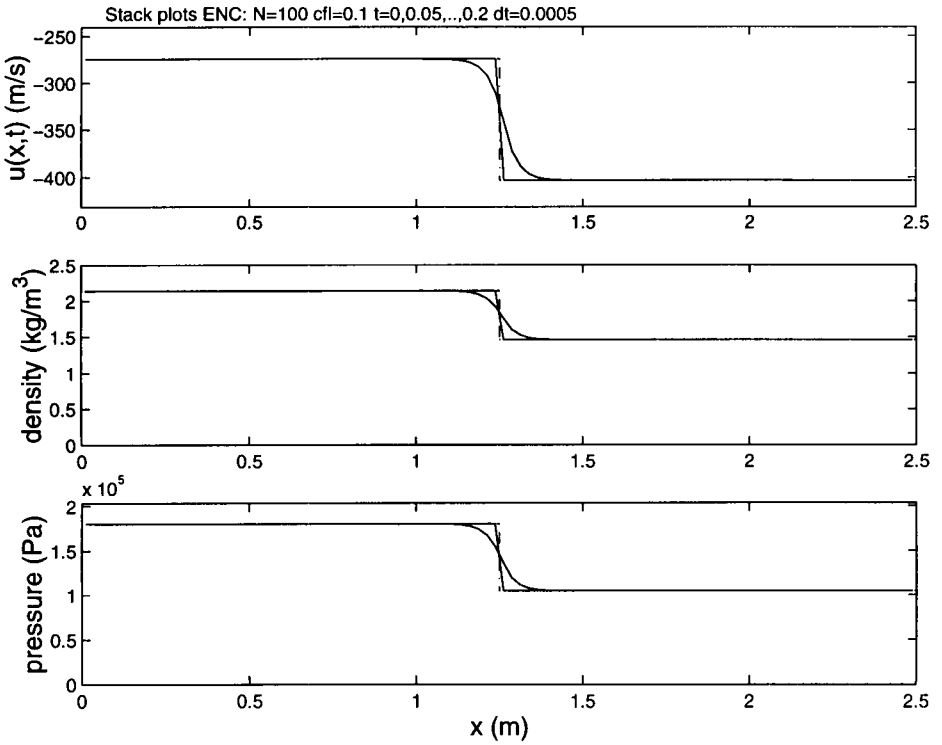


Figure 4.11: For a stationary shock in air, velocity, density, and pressure profiles are shown for (a) a LLF2 code and (b) an ENO code. Dashed-dotted lines represent exact and solid lines numerical solutions.

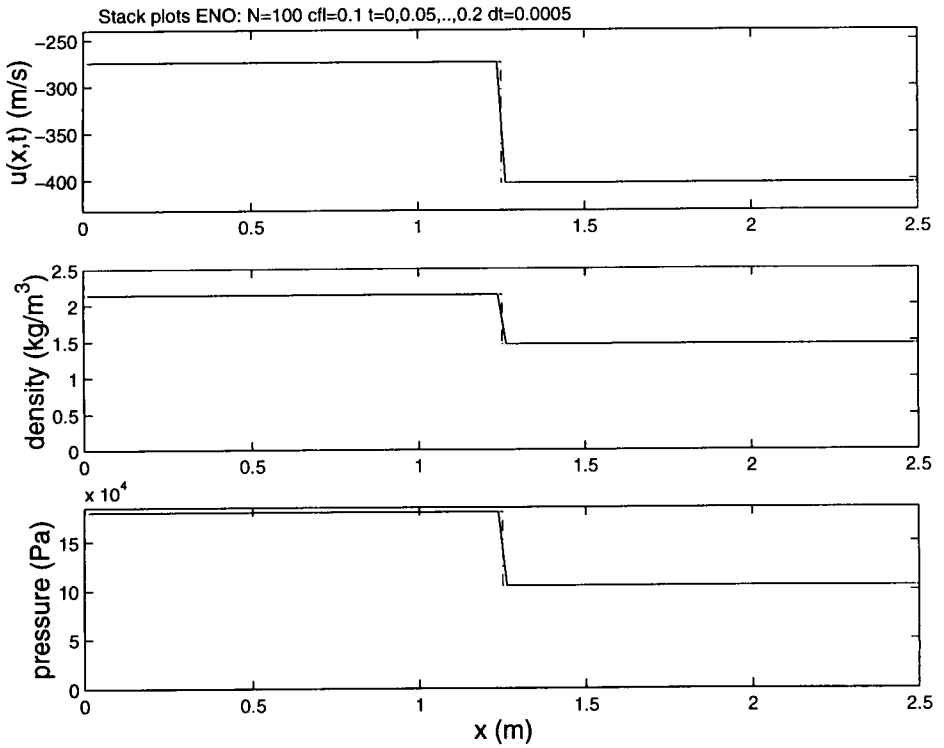


Figure 4.11: Continued.

4.2.2 Moving and reflected shock

Consider the space-time plot Fig. 4.13 for a moving and reflected shock in air. Given p_1, ρ_1, and p_2, several relations follow from the shock condition for the incoming shock (Whitham 1974):

$$
\begin{aligned}
z_i &\equiv \frac{(p_2 - p_1)}{p_1} = \frac{2\gamma(M_i^2 - 1)}{(\gamma + 1)}, \\
M_i &\equiv \frac{U_i - u_1}{a_1} = \sqrt{1 + \frac{(\gamma + 1)\, z_i}{2\gamma}}, \\
\frac{u_2 - u_1}{a_1} &= \frac{2(M_i^2 - 1)}{(\gamma + 1)\, M_i} = \frac{z_i}{\gamma\sqrt{(1 + (\gamma + 1)\, z_i/(2\gamma))}}, \\
\frac{\rho_2}{\rho_1} &= \frac{(\gamma + 1)\, M_i^2}{2 + (\gamma - 1)\, M_i^2} = \frac{1 + (\gamma + 1)\, z_i/(2\gamma)}{1 + (\gamma - 1)\, z_i/(2\gamma)}, \\
\frac{a_2}{a_1} &= \sqrt{\frac{(1 + z_i)\left(1 + (\gamma - 1)\, z_i/(2\gamma)\right)}{1 + (\gamma - 1)\, z_i/(2\gamma)}}
\end{aligned}
\tag{9}
$$

with $a_{1,2}$ the speed of sound in state $1, 2$, respectively.

By a change of sign the relations above can be altered for the reflected shock. Since $u_1 = u_3 = 0$ the expressions for u_2/a_1, $-u_2/a_2$ and a_2/a_1 determine z_r and the values for M_r, ρ_3, p_3 follow:

$$
z_r \equiv \frac{(p_3 - p_2)}{p_2} = \frac{z_i}{1 + (\gamma - 1)\, z_i/(2\gamma)},
$$

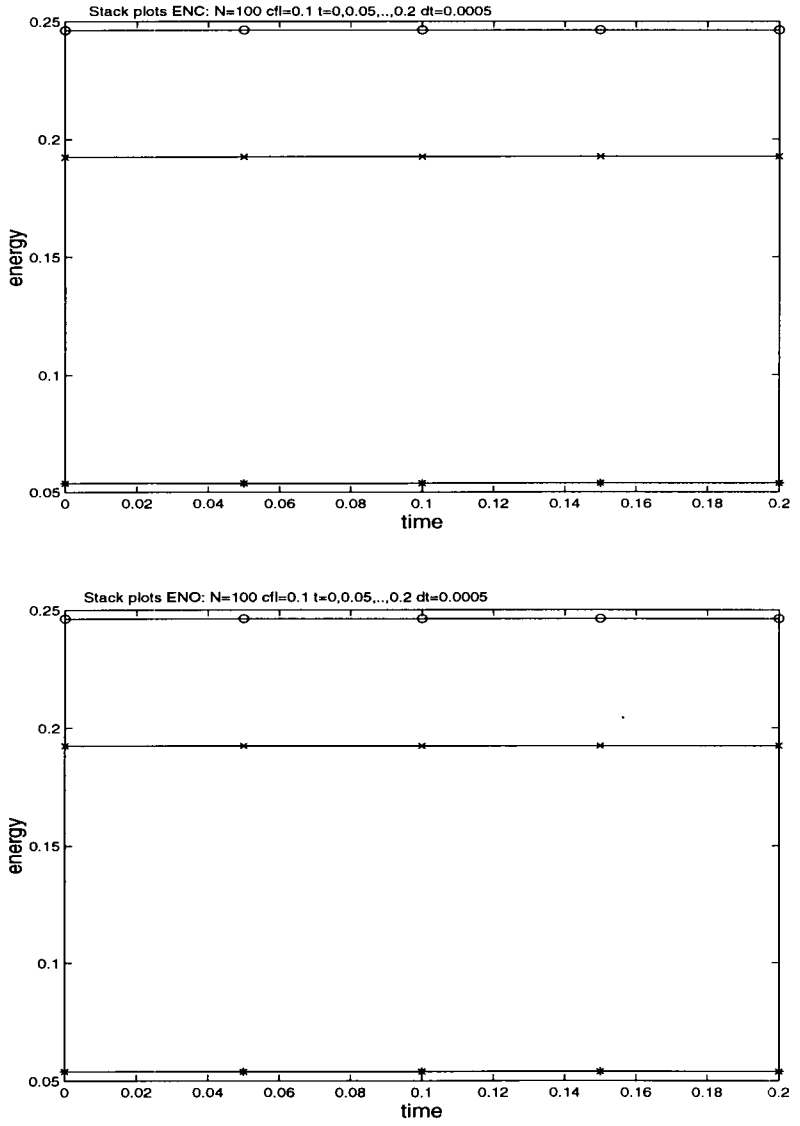


Figure 4.12: Kinetic (*), potential (x) and total energy (o) are shown for a) a LLF2 code and (b) an ENO code for a stationary shock in air.

$$
\begin{aligned}
M_r &\equiv \frac{U_r - u_2}{a_2} = -\sqrt{1 + \frac{(\gamma+1)\, z_r}{2\gamma}}, \\
\frac{u_3 - u_2}{a_2} &= \frac{2\,(M_r^2 - 1)}{(\gamma+1)\, M_r} = \frac{-z_r}{\gamma\sqrt{(1 + (\gamma+1)\, z_r/(2\gamma))}}, \\
\frac{\rho_3}{\rho_2} &= \frac{(\gamma+1)\, M_r^2}{2 + (\gamma-1)\, M_r^2}.
\end{aligned} \tag{10}
$$

The comparison between analytical and numerical solutions is good (Fig. 4.14 and Fig. 4.15), although the ENO simulation resolves the shock better —as usual.

4.2.3 Shock tube

The shock tube problem in air adds a new feature to the solution. The space time sketch Fig. 4.16 for a shock tube in air requires an extra region 2 between an interface and the shock.

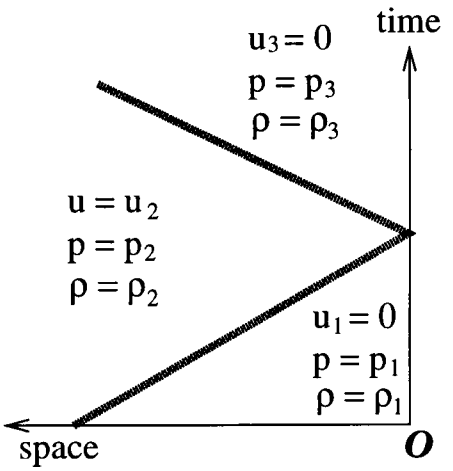


Figure 4.13: Space time sketch for shock reflection against a wall, at the right, in air.

The characteristics in air are (Whitham 1974)

$$\frac{1}{\rho\, a}\,\frac{\partial p}{\partial \rho}\,\frac{dp}{dt} \pm \frac{du}{dt} = 0 \tag{11}$$

on $C_{\pm}: dx/dt = u \pm a$. For a γ-law gas $p = \kappa(s)\,\rho^{\gamma}$ with entropy s and $a^2 = \gamma p/\rho$, the characteristics are

$$\frac{2\,a}{\gamma - 1} \pm u = constant \tag{12}$$

on $C_{\pm}$. Given u_1, p_1, ρ_1, relations (9) yield u_2, p_2, ρ_2 in terms of M_i. Across the interface pressures and velocities are continuous. A simple wave connects the given state 4 to region 3 because entropy is conserved. We thus find that

$$\kappa = p_4/(\rho_4)^{\gamma} = p_3/(\rho_3)^{\gamma}. \tag{13}$$

The positive characteristic $2\,a/(\gamma-1) + u = 2\,a_4/(\gamma-1)$ is constant in the simple-wave region and matching it to region 3 gives

$$2\,a_3/(\gamma - 1) + u_3 = 2\,a_4/(\gamma - 1). \tag{14}$$

Combining (13), (14) with $u_3 = u_2(M_i), p_3 = p_2(M_i)$ from (9), an implicit relation determining $M = M_i$ arises

$$\frac{\sqrt{\gamma}\,\kappa^{1/(2\,\gamma)}\,p_1^{(\gamma-1)/(2\,\gamma)}}{(\gamma-1)}\left(1 + \frac{2\gamma\,(M^2-1)}{(\gamma+1)}\right)^{(\gamma-1)/(2\,\gamma)} p_1{}^{(\gamma-1)/(2\,\gamma)} + \frac{a_1\,(M^2-1)}{(\gamma+1)\,M} - \frac{a_4}{(\gamma-1)} \;=\; 0. \tag{15}$$

Once M is know, the remaining variables can be determined. In the simple wave region $-a_4\,t < x < (u_3 - a_3)\,t$, one has

$$u - a = \frac{x}{t}, \qquad u + \frac{2\,a}{(\gamma-1)} = \frac{2\,a_4}{(\gamma-1)}. \tag{16}$$

After rearrangement of (16) and by using (13), we find

$$a \;=\; \frac{2\,a_4}{(\gamma+1)} - \frac{(\gamma-1)}{(\gamma+1)}\,\frac{x}{t},$$

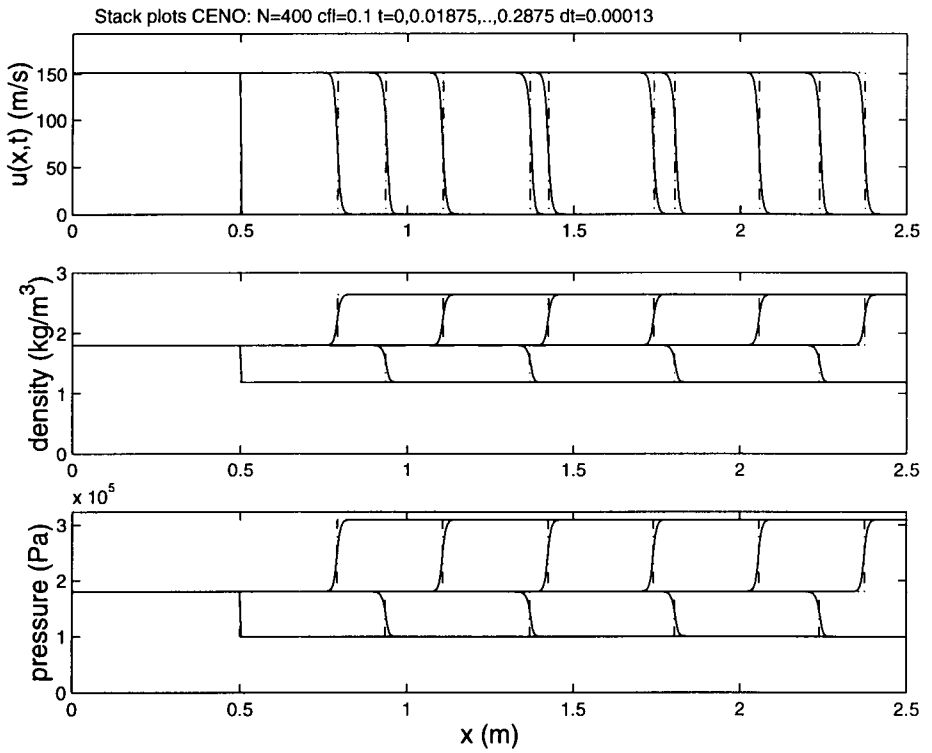


Figure 4.14: For a moving and reflected shock in air, velocity, density, and pressure profiles are shown for (a) a LLF2 code and (b) an ENO code.

$$\begin{aligned} \rho &= \left(\frac{a^2}{\gamma\kappa}\right)^{1/(\gamma-1)}, \\ p &= \kappa\left(\frac{1}{\gamma\kappa}\frac{2a_4}{(\gamma+1)} - \frac{(\gamma-1)}{(\gamma+1)}\frac{x}{t}\right)^{\gamma/(\gamma-1)}, \\ u &= \frac{2}{(\gamma+1)}\left(\frac{x}{t} + a_4\right). \end{aligned} \tag{17}$$

Exact and numerical solutions compare well (see Fig. 4.17 and Fig. 4.18); the ENO simulations clearly give sharper interface and shock resolution; the position of the interface is indicated by open circles in Fig. 4.17.

4.3 Dynamics of a melt-volatile fluid and air

4.3.1 Shock tube

The same space time sketch Fig. 4.16 as in air applies with the modification that the fluid left of the interface, in regions 3 and 4, consists of a basaltic fluid, and the fluid to the right of the interface, in regions 1 and 2, is air. While the shock relations (9) in air again determine state 2, given state 1, the left states 3 and 4 follow directly from the shock-tube solution for melt only, and continuity of pressure and velocity across the interface. The implicit relations

$$u_2 + c(p_2) - c_4 = 0, \tag{18}$$

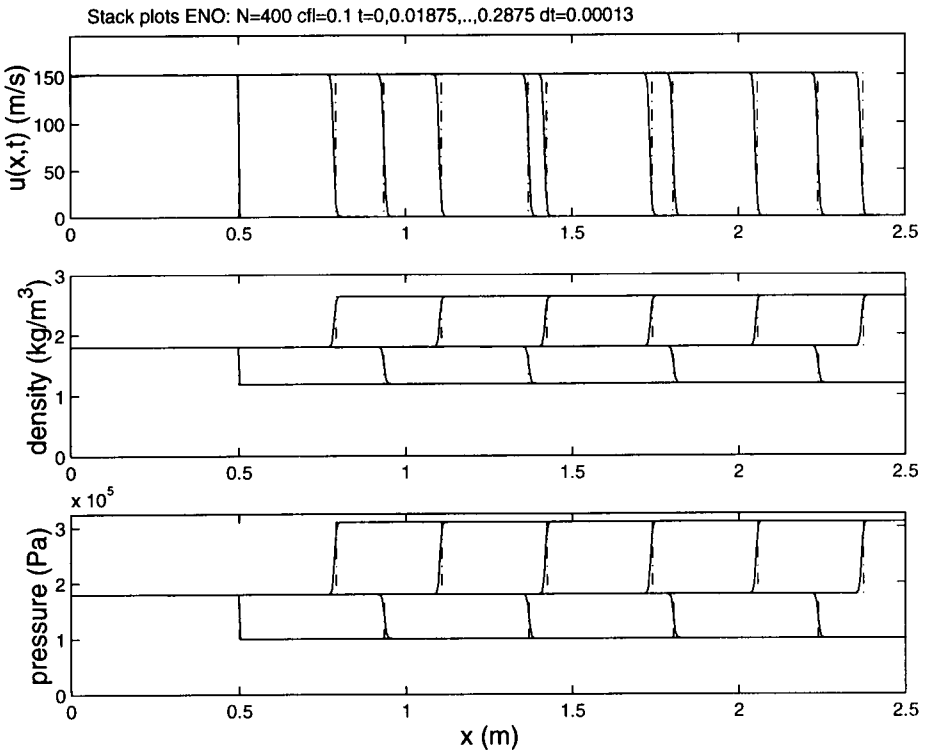


Figure 4.14: Continued.

$$u_2 = \frac{2\,a_1\,(M^2-1)}{(\gamma+1)\,M}, \tag{19}$$

$$p_2 = \left(1+\frac{2\,\gamma\,(M^2-1)}{(\gamma+1)}\right)p_1 \tag{20}$$

determine $M = U/a_1$, and the remaining variables follow. Note that a is defined differently in the melt ($a_{3,4}$) and air ($a_{1,2}$) regions. In the simple wave region $-a_4\,t < x < (u_3 - a_3)\,t$ velocity and pressure can be given in terms of space and time, but is more convenient to specify

$$x = (c_4 - c(p) - a(p))\,t, \qquad u = c_4 - c(p) \tag{21}$$

as function of pressure in the range $p_3 < p < p_4$ and time.

The comparison between exact and numerical shock tube solutions is problematic for initial conditions with the large density and pressure differences that are relevant for the study of magma-repository interactions. The comparison in Fig. 4.19 reveals that the interface is modelled reasonably well, but that the numerical shock moves ahead of the exact shock, which happens in the first few timesteps and over the first few grid cells. A reduction of only the time step yields no improvement. A ghost fluid method (see section 3.7.4) is used to match the two fluids across the interface. The simple extrapolation of entropy values of air into the ghost air/fluid region where the melt resides is a nonrigorous fix but other methods I have tried turned out to be unstable. The ghost fluid method is useful because it is easy to implement with little extra computational cost. Given these circumstances, the ghost fluid method is applied with brute force: increasing resolution clearly improves the performance, as may be seen in Fig. 4.20 where the resolution

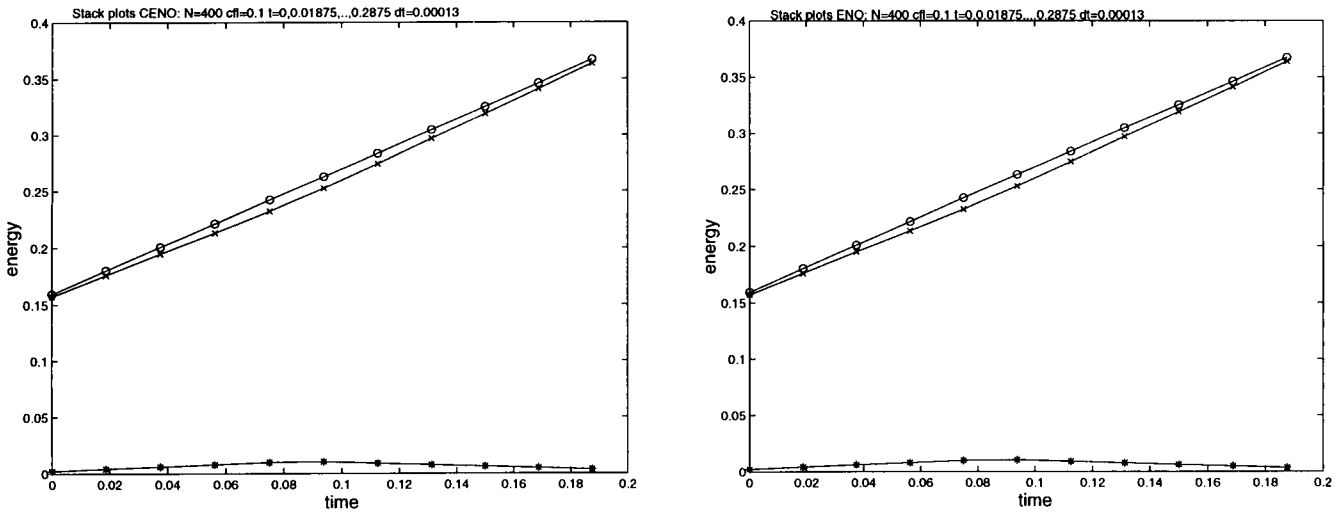


Figure 4.15: Kinetic (*), potential (x) and total energy (o) are shown for a) a LLF2 code and (b) an ENO code for a moving and reflected shock in air.

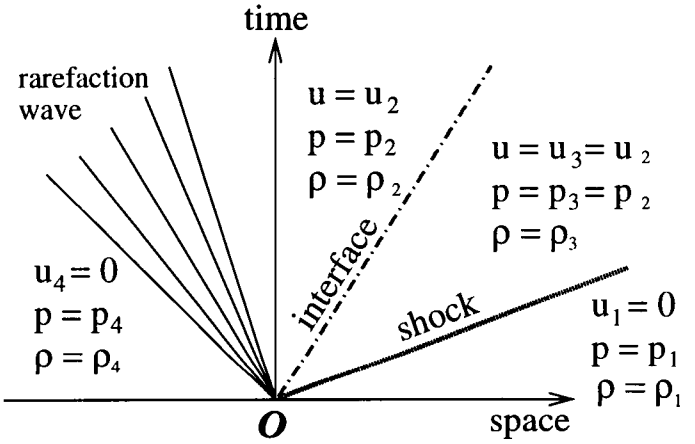


Figure 4.16: Four regions appear in this space time sketch for a shock tube in air.

is increased by a factor of 2.5. The ENO simulation is worse than the LLF2 simulation due to oscillations around the interface. The ENO algorithm could be modified to switch to a convex ENO (or LLF2) algorithm near the interface, but this has not yet been implemented. Based on this performance, further calculations employ the faster LLF2 scheme.

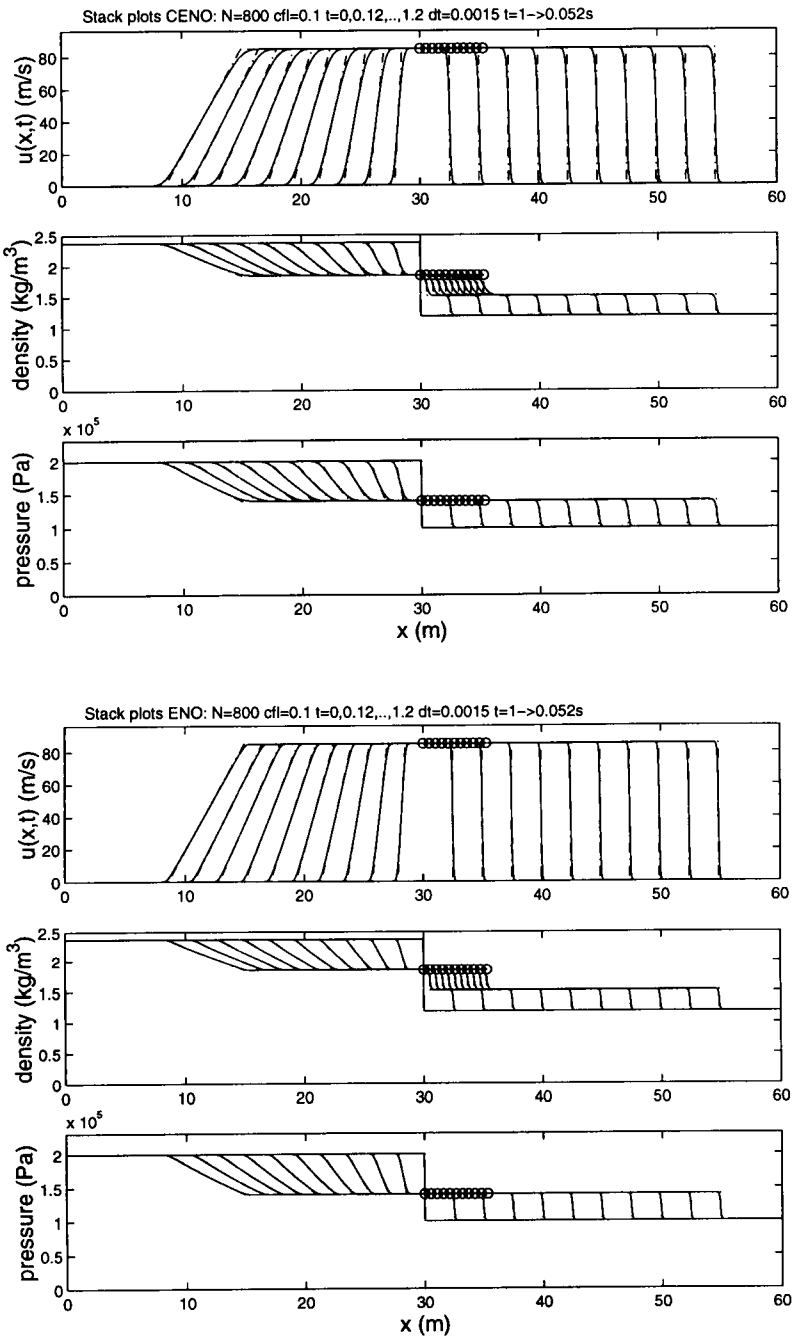


Figure 4.17: For a shock tube in air, velocity, density, and pressure profiles are shown for (a) a LLF2 code and (b) an ENO code. Open circles denote the exact positions of the interface.

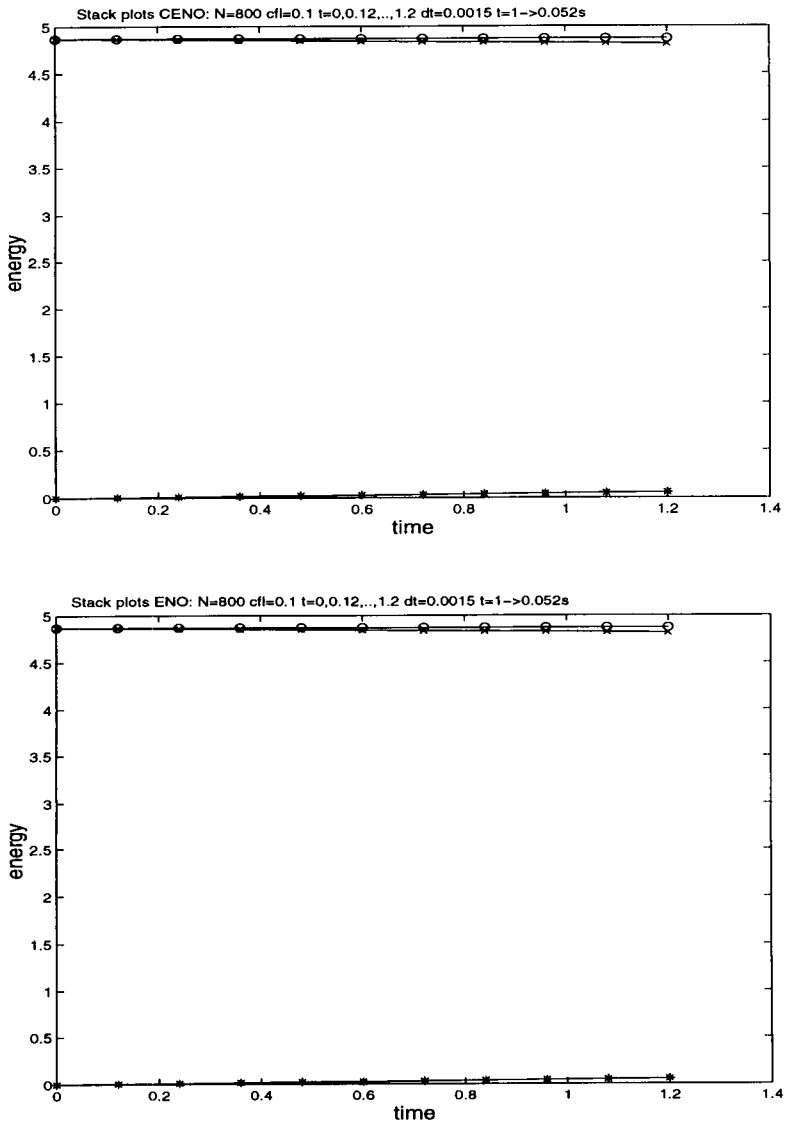


Figure 4.18: Kinetic (*), potential (x) and total energy (o) are shown for a) a LLF2 code and (b) an ENO code for a shock tube in air.

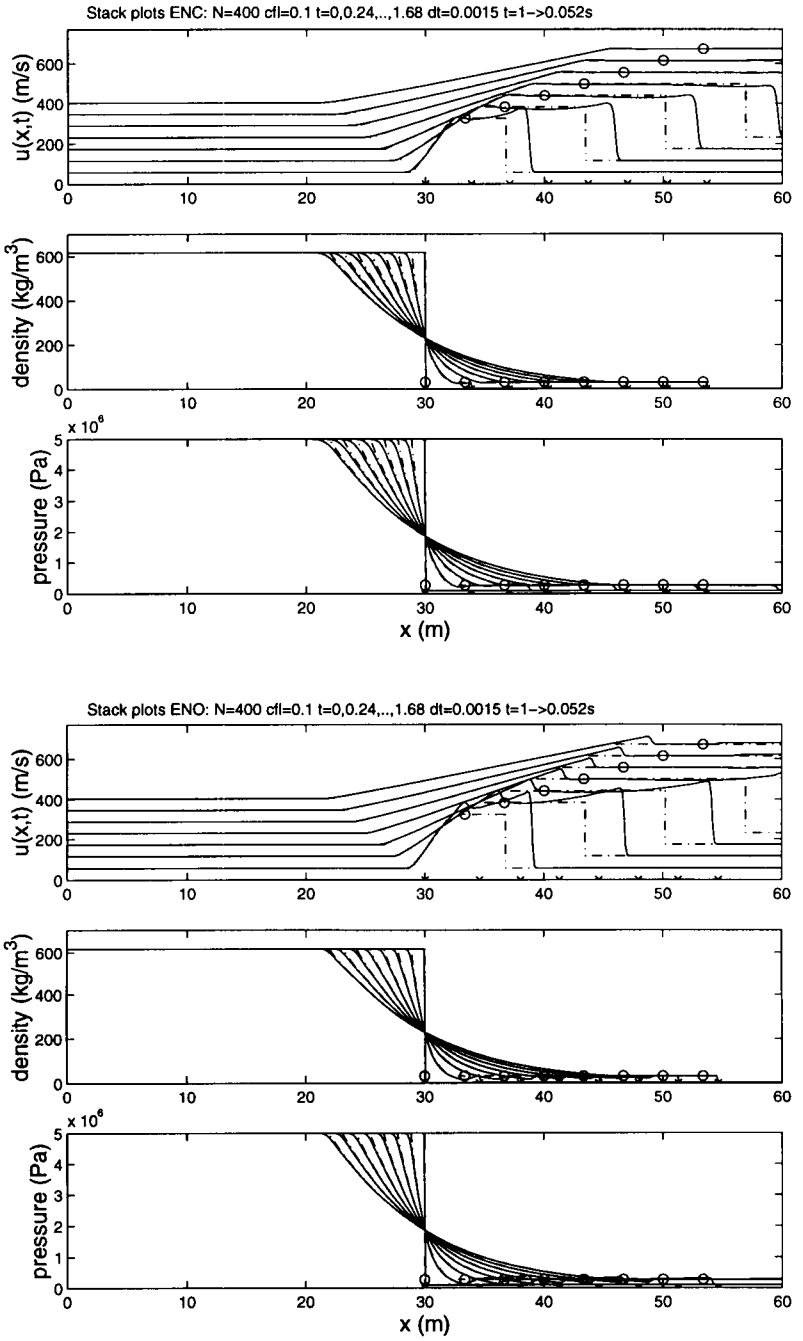


Figure 4.19: For a shock tube with a melt-volatile fluid left of the interface and air to the right, velocity, density, and pressure profiles are shown for (a) a LLF2 code and (b) an ENO code. Open circles denote the exact and crosses the simulated positions of the interface.

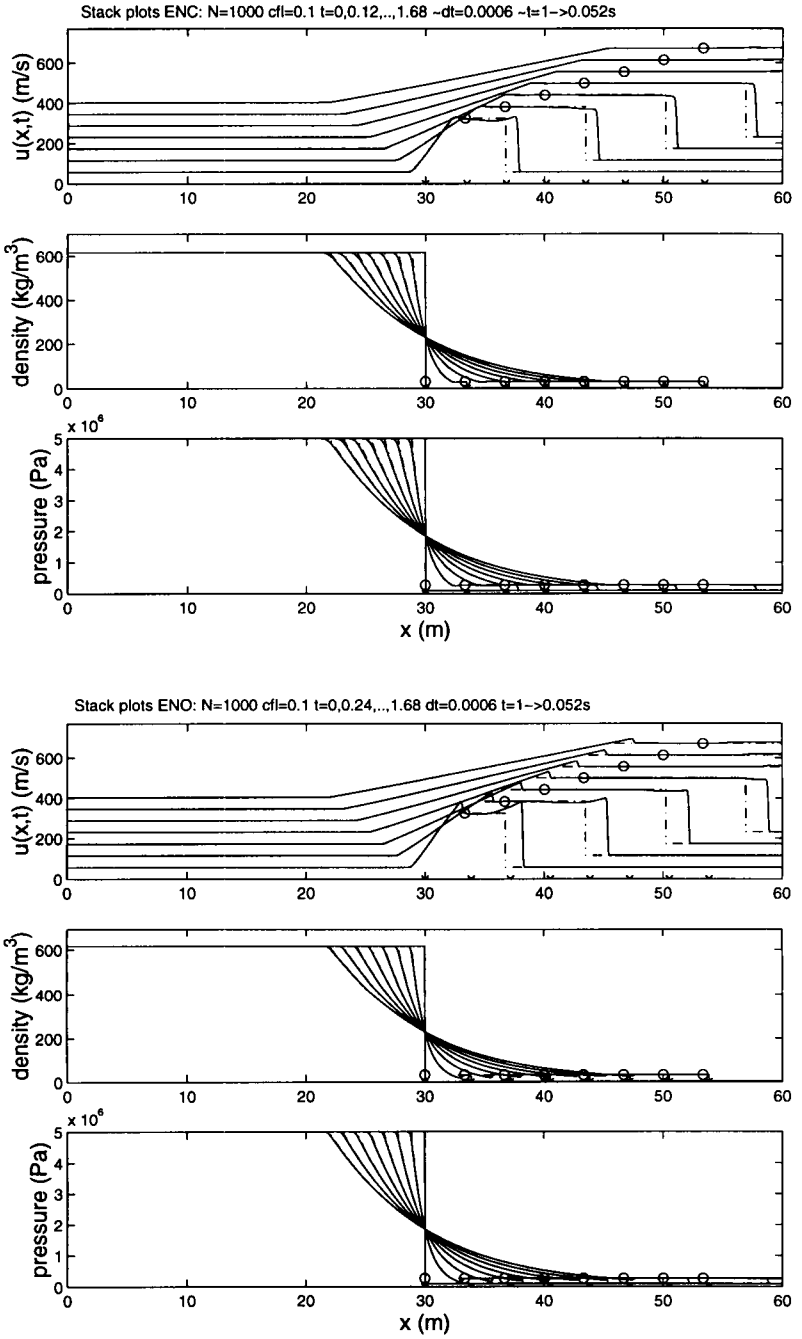


Figure 4.20: Same as in Fig. 4.19 with a resolution increase of 2.5.

4.3.2 Resonating shocks

For states with constant values of density, pressure, velocity and energy, values which may change across shocks and interface, the dynamics can be solved exactly by tracking each shock-shock, shock-wall and shock-interface interaction in space and time. The space-time diagram in Fig. 4.21 explains the evolution.

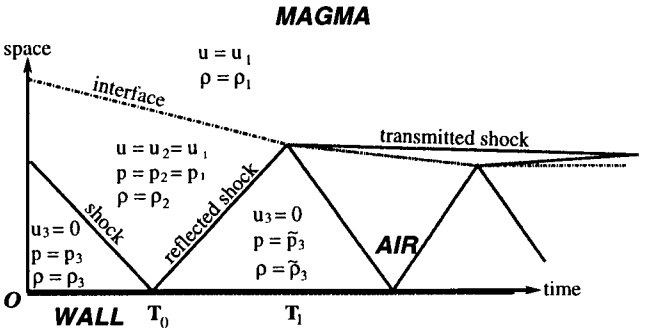


Figure 4.21: Space-time diagram of the resonating shock interactions between tunnel wall and melt-air interface. The vertical axis is space, the horizontal axis is time. The interface is denoted by a dashed-dotted line and the shocks by a solid line. Initially, there is an upper state 1 in the basaltic fluid, an intermediate air state 2, and a lower air state 3 at rest. (There is no gravity.)

Initially, basaltic fluid in constant state 1 flows towards the wall; across the interface in state 2, velocity and pressure in air are continuous and hence the same as in state 1, but air density has a lower value; the shock in air approaches the wall faster then the interface, and demarcates air state 2 from air state 3 while the latter is at rest. As time advances to T_0, the shock in air reflects and amplifies against the wall, the reflected shock reflects against the interface at time T_1, which has moved forward, and so on. The interface-shock reflection yields a reflected shock and a transmitted shock, the latter propagates in the basaltic fluid. The first couple of transmitted shocks are still swept towards the wall because they are unable to overcome the incoming fluid speed u_1 in state 1. They are, however, travelling away from the interface. Several transmitted shocks are thus generated in the basaltic fluid, the later ones overtake and annihilate the earlier ones and increase the speed of the shock front in the magma. Finally, a strongly-amplified shock in the magma propagates away from the wall with great speed. Numerical evidence in the stack plots of Fig. 4.22 and the energetics of Fig. 4.23 supports this picture. In particular, pressure and velocity profiles indicate how the pressure rises in the air pocket following each shock reflection, while the movement of the interface emerges in the density profile; the interface position is marked on all horizontal axes. (The small density jump of the shock in air has almost disappeared in the line thickness, but a large jump across the transmitted shock in magma appears after the first shock-interface encounter in profile 6.)

As the magma-air interface slows down and finally arrests, the dynamics is no longer presented accurately in our flow-tube model. Upon slowing down, a gravity current must form, fill the floor at the end of the drift, and leave an air pocket in the upper corner, which will later start to spread across the ceiling of the drift. Gravity currents are clearly not captured by the flow-tube model. Based on a long-wave assumption, one could extend the flow-tube model to include a free-surface $h(\xi_1, t)$ as function of a horizontal spatial coordinate and time only, in analogy to shallow-water and gravity-current modelling in geophysical and geological fluid dynamics (e.g. Gill 1982). Alternatively, one could simulate the dynamics of a two-dimensional flow model, averaged over the lateral y-direction, in a vertical plane through the dyke-drift system before resorting to computationally expensive three-dimensional simulations. While these more

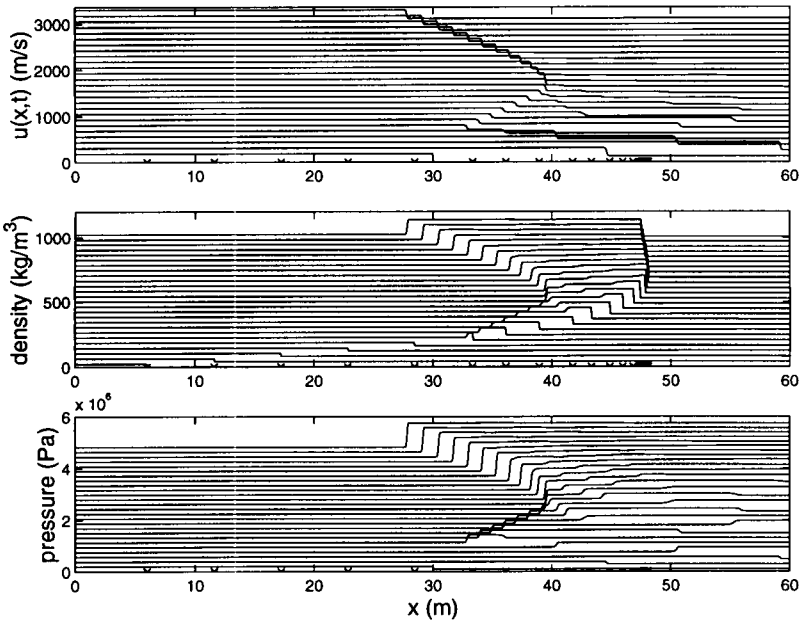


Figure 4.22: Velocity, density and pressure profiles are shown for a simulation of resonating shocks between a tunnel wall and melt-air interface. The tunnel wall is at the right. The crosses at the x-axis indicate the forward moving positions of the interface. Initial pressures are $p_1 = p_2 = 2 * 10^5\, Pa$ and $p_3 = 10^5\, Pa$. Observe the strong compression of air as time advances.

intricate details are left for future study, the shock amplification process is arguably generic because the high-speed reflected shock will remain essentially one-dimensional away from the wal

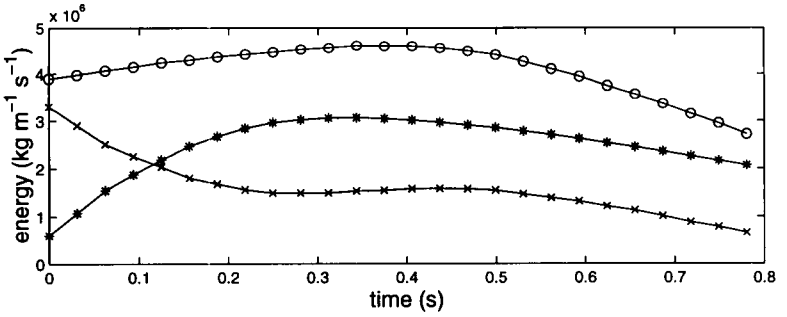


Figure 4.23: Kinetic (*), potential (x), and total (o) energy in a simulation of resonating shocks between the end wall of the tunnel and the magma-air interface (LLF2 code).

The above examination emphasizes that large shock amplification will occur at the drift wall in magma-air interactions. The basis simulation in the next section reveals that a similar shock amplification mechanism occurs in a dyke-drift geometry. Further simulations in Bokhove (1999) show how the stronly amplified reflected shock is partially transmitted at the dyke-drift transition, and descends into the magma dyke, but also partially reflected back into the drift.

Chapter 5

Magma-repository interactions

In this chapter, we define the geometry of the dyke and drift system used in several simulations. A basis simulation on magma-repository interactions will be described along with a modelling results for tracer advection and cannister movement. More simulations assessing the sensitivity to changing parameters and geometries are presented in Bokhove (1999).

5.1 Geometry and grid transformation

A cross section of the magma dyke and repository drift is given in Fig. 5.1. The connection between dyke and drift is represented as an elliptical arc.With a coordinate origin $\mathcal{O}$ at the South-East intersection of dyke and drift, the arc has coordinates: $x = -(1/2)w\cos[(\xi_1 - L_{dike})\,\pi/(2\,L_{arc})], z = (1/2)d\cos[(\xi_1 - L_{dike})\,\pi/(2\,L_{arc})]$.

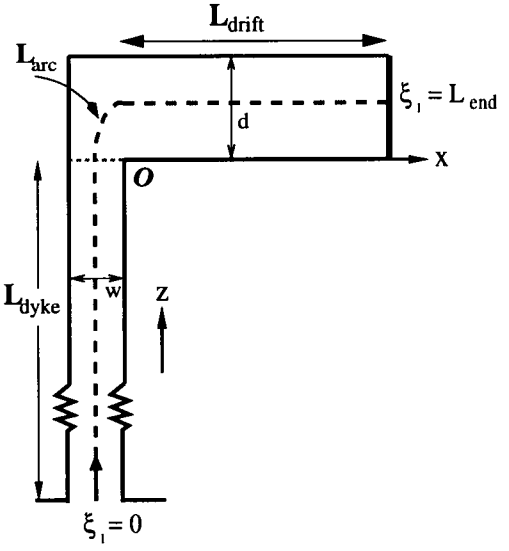


Figure 5.1: A vertical cross section of the magmatic dyke and drift system defines the various length scales involved in all the flow-tube model simulations.

Characteristic scales in the configuration are: the length of the magma dyke in the simulation L_{dyke}, the distance from the origin $\mathcal{O}$ to the end of the drift L_{drift}, arc length $L_{arc} = \frac{1}{2}\,d\,E(m)$ where $E(m)$ is

the elliptical integral of the second kind with $m = (d-w)^2/d^2$, and the overall length L_{end}. For longer dyke lengths the pressure rises above the critical pressure p_c for which all volatiles are dissolved in the then incompressible magma. Dyke lengths are therefore restricted to stay below critical pressure p_c. Given the geometry, gravity in the flow-tube model is present in the dyke, but diminishes in the arc to zero in the drift [cf. (10)].

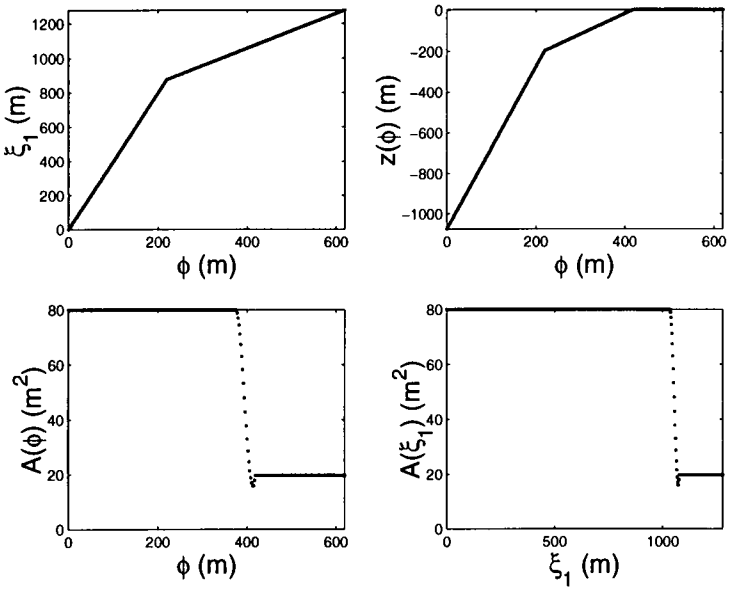


Figure 5.2: Dependence of (a) $\xi_1(\phi)$, (b) $z(\phi)$, (c) $A(\xi_1)$ and (d) $A(\phi)$ on the stretched coordinate ϕ for 300 ϕ-grid points. The numerical grid is coarser near the magmatic chamber.

The cross-sectional area $A(\xi_1, t)$ of the flow tube is constant in the drift and in most of the dyke, A_{dyke}, for $\xi_1 < L_{end} - L_{drift} - (1/2)\,L - d$ but narrows in a nozzle or transition zone with minimum area $A = \pi\, d\, w$ before it enlarges to the constant drift area, A_{drift}. The characteristic spacing between drifts is L. This transition zone is a crude model of the characteristic three-dimensional flow path and volume between dyke and drift. Most resolution is needed in the region of prime interest around the dyke-drift intersection and in the drift, so instead of ξ_1 a smooth coordinate transformation to another coordinate ϕ is made. The regular ϕ-grid yields a coarse ξ_1-grid near the bottom of the magma dyke. The coordinate transformation is, however, one-to-one in the upper part of the magma dyke, in the arc and in the drift. With some abuse of notation, the unit of ϕ is simply given in meters in subsequent graphs. The dependence of $\xi_1(\phi)$, $z(\phi)$, $A(\phi)$ and $A(\xi_1)$ in the basis simulations and most other simulations is shown in Fig. 5.2.

The formulas for $\xi_1(\phi)$, $h_1(\phi)$, $z(\phi), \partial z/\partial\phi$, and $A(\phi)$ are given by:

$$\xi_1(\phi) = \begin{cases} (1/\alpha)\,\phi, & 0 < \phi < \phi_L, \\ (1/\alpha)\left(e^{\eta\,(\phi-\phi_L)/\Delta\phi_L}\,(\phi-\phi_L) + \phi_L\right), & \phi_L < \phi < \phi_L + \Delta\phi_L, \\ \phi + L_{dyke} - \phi_{dyke}, & \phi_L + \Delta\phi_L < \phi < \phi_{end}, \end{cases} \tag{1}$$

$$h_1(\phi) = \begin{cases} (1/\alpha), & 0 < \phi < \phi_L, \\ (1/\alpha)\, e^{\eta\,(\phi-\phi_L)/\Delta\phi_L} \left(1 + \eta\,(\phi - \phi_L)/\Delta\phi_L\right), & \phi_L < \phi < \phi_L + \Delta\phi_L, \\ 1, & \phi_L + \Delta\phi_L < \phi < \phi_{end} - L_{drift} - L_{arc}, \\ \frac{\pi}{4\,L_{arc}} \sqrt{w^2\, \sin^2\left(\frac{\pi\,(\phi-\phi_{dyke})}{2\,L_{arc}}\right) + d^2\, \cos^2\left(\frac{\pi\,(\phi-\phi_{dyke})}{2\,L_{arc}}\right)}, & \phi_{end} - L_{drift} - L_{arc} < \phi < \phi_{end} - L_{drift}, \\ 1, & \phi_{end} - L_{drift} < \phi < \phi_{end}, \end{cases}$$

$$z(\phi) = \begin{cases} (1/\alpha)\phi - L_{dyke}, & 0 < \phi < \phi_L, \\ (1/\alpha)\,(\phi - \phi_L)\; e^{\eta\,(\phi-\phi_L)/\Delta\phi_L} + (1/\alpha)\,\phi_L - L_{dyke}, & \phi_L < \phi < \phi_L + \Delta\phi_L, \\ \phi - \phi_{dyke}, & \phi_L + \Delta\phi_L < \phi < \phi_{dyke}, \\ \frac{1}{2}\,d\,\sin\left(\frac{\pi\,(\phi-\phi_{dyke})}{2\,L_{arc}}\right), & \phi_{dyke} < \phi < \phi_{end} - L_{drift}, \\ \frac{1}{2}\,d, & \phi_{end} - L_{drift} < \phi < \phi_{end}, \end{cases}$$

$$\frac{dz(\phi)}{d\phi} = \begin{cases} (1/\alpha), & 0 < \phi < \phi_L, \\ (1/\alpha)\, e^{\eta\,(\phi-\phi_L)/\Delta\phi_L} \left(1 + \eta\,(\phi - \phi_L)/\Delta\phi_L\right), & \phi_L < \phi < \phi_L + \Delta\phi_L, \\ 1, & \phi_L + \Delta\phi_L < \phi < \phi_{dyke}, \\ \frac{\pi\,d}{4\,L_{arc}}\, \cos\left(\frac{\pi\,(\phi-\phi_{dyke})}{2\,L_{arc}}\right), & \phi_{dyke} < \phi < \phi_{end} - L_{drift}, \\ 0, & \phi_{end} - L_{drift} < \phi < \phi_{end}, \end{cases}$$

$$A(\phi) = \begin{cases} A_{dyke}, & 0 < \phi < \phi_{end} - L_{drift} - \frac{1}{2}\,L - d, \\ A_{dyke} + \frac{1}{2}\,(\pi\,d\,w - A_{dyke}) & \\ \left[1 - \cos\left(\frac{2\,\pi\,(\phi-\phi_{end}-L_{drift}-d-L/2)}{L}\right)\right], & \phi_{end} - L_{drift} - \frac{1}{2}\,L - d < \phi < \phi_{end} - L_{drift} - d, \\ \pi\,d\,w + \frac{1}{2}\,(A_{drift} - \pi\,d\,w) & \\ \left[1 - \cos\left(\frac{\pi\,(\phi-\phi_{end}-L_{drift}-d)}{d}\right)\right], & \phi_{end} - L_{drift} - d < \phi < \phi_{end} - L_{drift}, \\ A_{drift}, & \phi_{end} - L_{drift} < \phi < \phi_{end}, \end{cases}$$

with $\phi_{dyke} = \phi_{end} - L_{drift} - L_{arc}$; $\phi_{end} = L_{drift} + L_r - \Delta L_r + \phi_L + \Delta\phi_L$; $\phi_L = \alpha\,(L_{dyke} + L_{arc} - L_r)$; and $\Delta\phi = \alpha\,\Delta L_r\, e^{-\eta}$

These functions $1/h_1(\phi)$, $z(\phi)$, $\partial z/\partial\phi$, $A(\phi)$ and $\partial A(\phi,t)/\partial\phi$ are found in the core programs encmvs1.c, encmvt1.c and encmax1.c as functions H1scalef(x1), Zhydro(x1), Zdiff(x1), Area(x1) and Diffarea(x1), respectively.

5.2 Basis simulation

Characteristic parameter values for basaltic magma at Yucca Mountain are: $n_0 = 2\,wt\%, L = 80\,m, w = 1\,m, d = 5\,m, A_{drift} \approx 19.64\,m^2, A_{dyke} = 80\,m^2, L_{drift} = 200\,m, L_{dyke} = 1075\,m$; lithostatic pressure at $D = 400\,m$ $p_L = \sigma\,g\,D = 10^7\,Pa$ and overpressure $P_o = 10^7\,Pa$ give a total pressure at the dyke tip of $P_t = 2 * 10^7\,Pa$; and the critical void fraction at the point of fragmentation is $\alpha = 70\%$.

The initial condition is quiescent flow on either side of the drift entrance, hydrostatic balance in the magma dyke with a pressure $P_t = 2 * 10^7\,Pa$ at the dyke tip, and air at atmospheric pressure and room temperature in the drift. The initial magma pressure and density can be determined by combining the

equation of state and the hydrostatic balance condition in one relation, which root can be found numerically to yield the pressure. Except for hydrostatic balance, the initial condition is reminiscent of one for a classical shock tube.

The first basis simulation advances $1.139\,s$. Its flow profiles are shown in Fig. 5.3. Besides shock reflections between drift wall and magma-air interface similar to those studied in Fig. 4.22, a rarefaction wave is seen to travel into the magma dyke. Both pressure and velocity profiles reveal the interplay of resonating shocks between drift wall and magma-air interface during the first tens of seconds. The density profile again strikingly marks the movement of the interface and the transmitted shock. Just before the end of the simulation at $t = 1.139\,s$ a strongly amplified shock wave is seen to emerge in the magma. Its magnitude is about 28 times larger than the magnitude of the initial shock wave in air.

Further simulations and a parameter study, assessing the dependence of the rarefaction and shock wave propagation in the above and similar dyke-drift systems, can be found in Bokhove 1999.

5.3 Cannister movement and tracer advection

Once a cannister is submerged in basaltic fluid it experiences several forces. Since the air density is initially atmospheric, we ignore the force due to the shock wave in air. The static or dynamic friction of a cannister with the floor of the drift is

$$\lambda\,(\rho_{can} - \bar{\rho})\,g\,L_{can}, \tag{2}$$

where $\rho_{can} \sim 11000\,kg/m^3$ is the cannister density, length $L_{can} \sim 5\,m$, $\bar{\rho}$ is the average melt density across a cannister, and $\lambda = \lambda_s = 1.0$ and $\lambda = \lambda_d = 0.5$ are the static and dynamic friction, respectively. The viscous force exerted on the cannister by the moving magma scales like $\mu\nabla^2 u \sim \mu\,\bar{u}/r_{can}^2$ with $\mu = \mu_0$, cannister radius $r_{can} \sim 1\,m$ and average velocity $\bar{u}$. The drag force on a cannister is proportional to $\rho\,(u-v)\,\partial u/\partial x \sim \bar{\rho}\,(\bar{u}-v)\,|\bar{u}-v|\,C_D/r_c$ with drag coefficient $C_D \approx 1-2$. Newton's equations of motion for a cannister thus become:

$$\begin{aligned} \frac{d\xi_{can}}{dt} &= v, \\ \rho_{can}\frac{dv}{dt} &= -\lambda\,\rho_{can}\,g\,\mathrm{sign}(v) + \mu\frac{(\bar{u}-v)}{r_{can}^2} + \bar{\rho}\,C_D\frac{(\bar{u}-v)\,|\bar{u}-v|}{r_{can}} - \frac{\Delta P_{can}}{L_{can}}, \end{aligned} \tag{3}$$

where we have neglected Archimedes force since $\bar{\rho} \ll \rho_{can}$. The position of the cannister is ξ_{can}. When the total force exerted on a cannister exceeds the static friction, it will begin to move. The critical pressure drop across a cannister is approximately $3 * 10^5\,Pa$ [i.e. consider the right-hand-side of (3) for $\bar{u} - v = 0$]. As we have seen in earlier simulations, the shock wave in air is too weak so only the viscous forces or the pressure drop across the reflected strongly amplified shock will kick a cannister into motion. The time dependence of A in air, the right-hand side of (16c), is therefore ignored a priori. To assure that $\partial A/\partial\xi_1$ in (10) is finite the cannister's cross-section is assumed to have two meter long tapered sides. Total cannister length is therefore lengthened to seven meters. The trajectory of the lower corner of the cannister is graphed in Fig. 5.4. Parameter values are similar to the one in the basis simulation, except that the cross-sectional area of the drift is no longer constant because the cannister acts as a small, moving nozzle. The cannister begins to move when the frictional forces are strong enough to overcome the static resistance. Once the cannister has reached the magma-air interface we have stopped the simulation.

Discrete cannister movement can in principle be blocked by anchorage of the cannisters. An alternative scenario is that the immobile cannisters get damaged or partially melted by the hot, high-speed magma

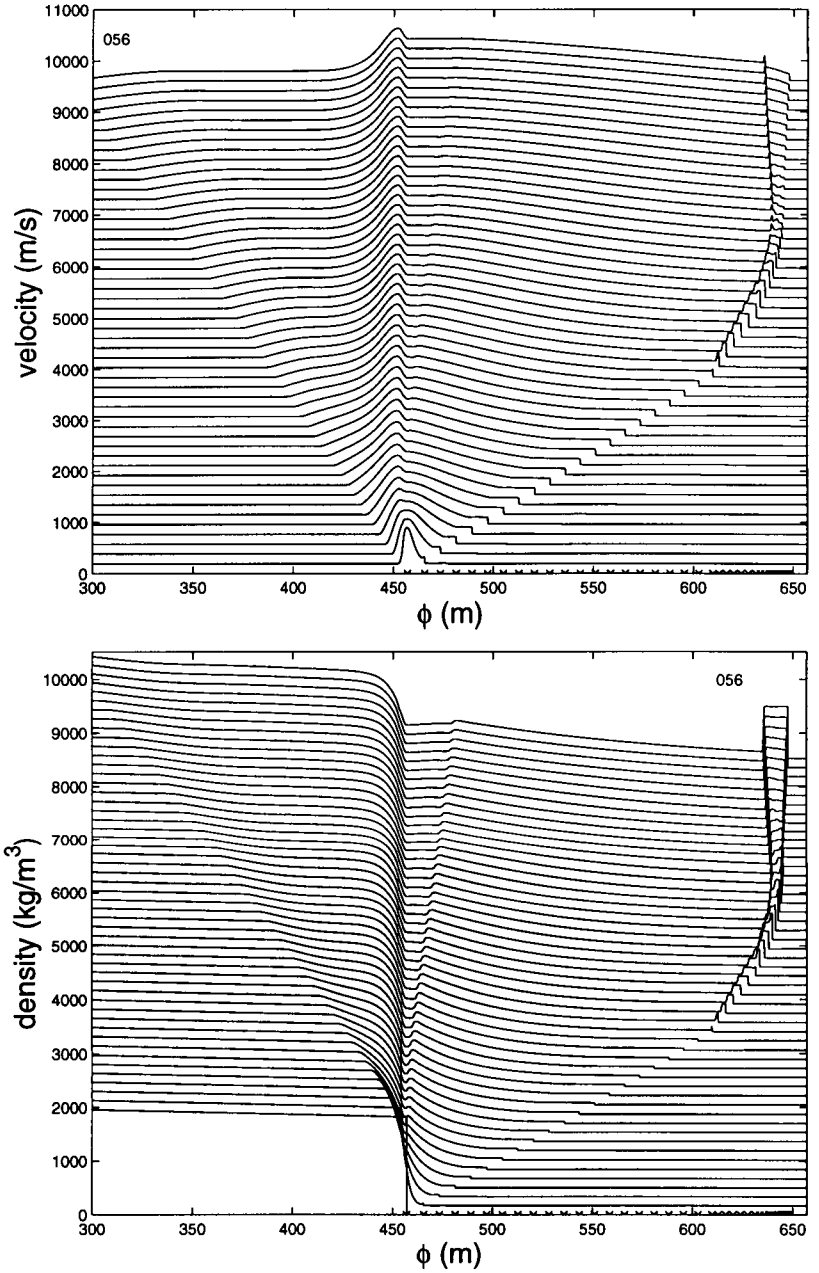


Figure 5.3: Velocity, density and pressure profiles are shown for magma-air interactions in a dyke-drift system. The drift starts at $\phi = 457\,m$. Simulation 056.

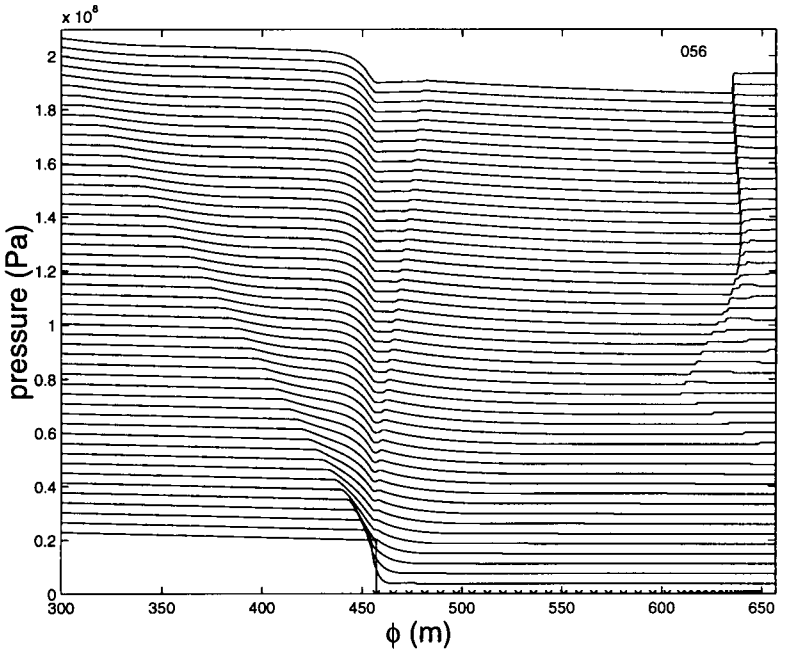


Figure 5.3: Continued simulation 056.

mixture. Such damage may cause small waste pellets to be released in the flow. To assess what happens to small pieces of waste, 33 passive particles are released in the drift 0.8 second after the start of the explosion when the shock in air has passed and the reflected shock has been formed at the end of the drift. As is clear from the convergence of particle tracks in Fig. 5.5, all particles get simply advected by the flow towards the magma-air interface and then their motion halts after the reflected shock wave has passed, Tracer transport by the incoming magma is therefore minor. Tracer transport would become much larger when a magma dyke breaks through at the end of the drift as a result of the large pressure build up.

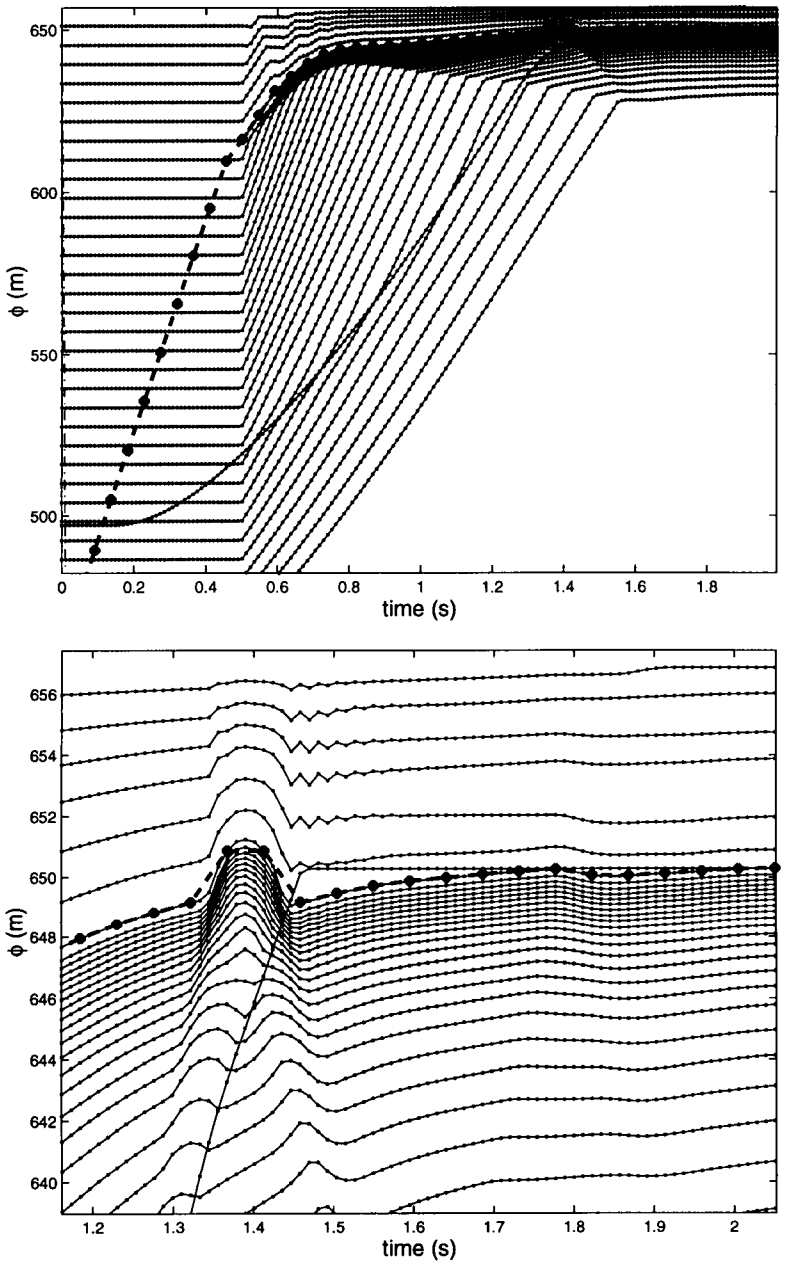


Figure 5.4: Trajectories of 33 passive particles, released at $t = 0.5\,s$, are shown along with the trajectory of one cannister. The cannisters is stopped when it hits the wall. A large view and a detail of several trajectories are shown for simulation 091.

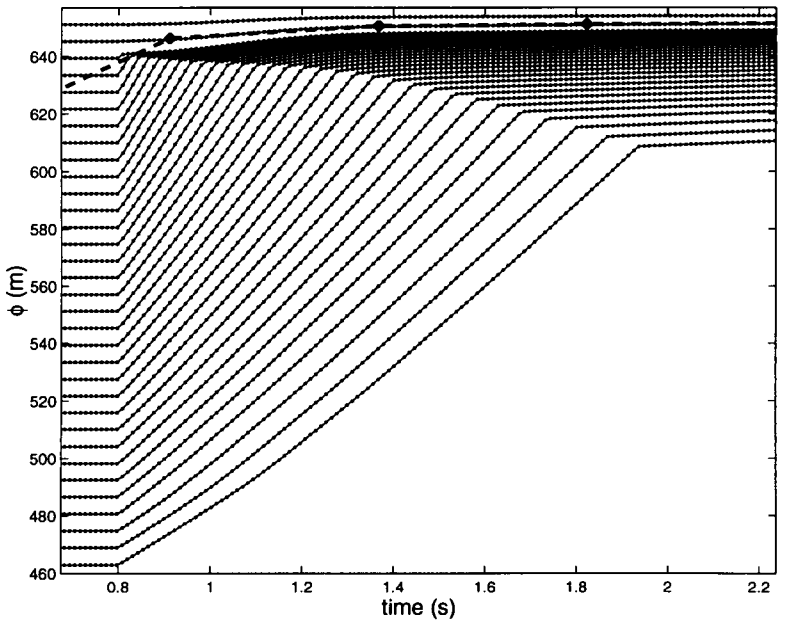


Figure 5.5: Passive particles released in the tunnel after 0.8 s get advected by the flow towards the magma-air interface. The interface is denoted by the dashed line and crosses. Simulation 064 has initial dyke-tip pressure $\Delta P_t = 2.5 * 10^7\, Pa$ and is otherwise the same as basis simulation 060.

Chapter 6

Conclusion

A detailed report has been given on numerical modelling of magma-repository interactions by explosive volcanic flows. In chapter 2, a mathematical flow-tube and flow area model have been introduced for magma-air interactions between a dyke with volatile-rich basaltic melt and a waste repository with air at atmospheric pressure. The flow-tube equations for the compressible magma and the gas dynamics of air are subsequently discretized in chapter 3. The presentation has focussed on shock-capturing numerical techniques since a major feature of magma-air interactions is the presence of shocks. The way the numerical algorithms were translated into code has been described in section 3.8. Various numerical tests in chapter 4 provided a validation of the developed programs. A basis simulation on magma-repository interactions was described in chapter 5 along with a modelling results for tracer advection and cannister movement. The programs underlying the results in this last chapter form the basis for the magma-repository interactions analyzed in Bokhove (1999).

Potential users of the programs for magma and magma-air flow in flow tube models with adjustable geometries have been given detailed insights in the numerical algorithms and the dyke-drift geometry. In parallel, it has also been indicated throughout the report how and where these aspects are implemented in the code. With the instructions and listings in the Appendices, and the comments in the codes, I hope we can all keep using the programs. Constructive comments are always welcome.

There are several important numerical aspects that merit further research. These include the following. (i) Further comparisons of the numerical flow tube model can be made with stationary hydraulically-controlled flows in vertical, horizontal and curvilinear dyke or conduit geometries; these stationary solutions are governed by ordinary differential equations [see Woods (1995) for explosive volcanic flows in vertical conduits]. (ii) Higher-order convex ENO schemes (Liu and Osher 1998) are expected to perform better for the mixed hyperbolic and forced flows considered here; there is research potential in developing numerical schemes that are optimal under transient and quasi-steady conditions. (iii) The numerical treatment of interface dynamics should be improved on a fundamental level, rather than by increasing the resolution globally or locally. While the ghost-fluid method of Fedkiw *et al.* (1999) is simple and reasonably accurate, its mathematical basis is imperfect. (iv) The spatially one-dimensional flow tube model can be extended to include gravity-curent formation at the end of the tunnel. The interface is then allowed to tilt, spread and thus form a compressible gravity-current, instead of remaining vertical after it slows down. (v) Two-dimensional modelling of magma-repository interactions would give detailed insights into the geometry of the magma flow around the dyke-drift transition where the vertical momentum of the flow is absorbed by the ceiling of the tunnel and the horizontal momentum is generated by horizontal pressure gradients. In addition, two-dimensional

modelling of the slumping interface at the end of the drift is feasible.

Acknowledgements The numerical work benefitted significantly from research of the author performed under an EC MAST-III Surf and Swash Zone Mechanics grant received by Professor D.H. Peregrine, who also kindly suggested involvement of the author in the project and shared his knowledge in several discussions. Dr. A. Rogerson assistance in the initial development of ENO numerical codes for shallow-water dynamics is gratefully acknowledged.

This manuscript is the result of work that the author performed for and in part at the Center for Nuclear Waste Regulatory Analysis (CNWRA) and for the U.S. Nuclear Regulatory Commission (NRC), through a contract of the CNWRA with the University of Bristol. The presented results do not necessarily reflect the views or regulatory position of the NRC. The computational framework at South West Research Institute was arranged well by L. Connor and Dr. C. Connor. Last but not least, I wish to thank SWRI for their hospitality during my stay.

References

Batchelor, G.K., 1988: *Fluid Dynamics.* Cambridge University Press, 615 pp.

Bokhove, O., 1999: Explosive magma-air interactions by volatile-rich basaltic melts in a dyke-drift geometry. 41 pp.

Fedkiw, R.P., Aslam, T, Merriman, B., and Osher,S., 1999: A non-oscillatory Eulerian approach to interfaces in multimaterial flows (the ghost fluid method),. *J. Comp. Phys.* **152**, 457–492.

Gill, A.E., 1982: *Atmosphere-Ocean Dynamics.* Academic Press, 662 pp.

Lamb, H., 1932: *Hydrodynamics.* Dover, Toronto, 508 pp.

Liu, X.-L. & Osher, S., 1998: Convex ENO high order multi-dimensional schemes without field by field decomposition on staggered grids. *J. Comp. Phys.* **142**, 304–330.

Leveque, R.J., 1997: Wave propagation algorithms for multidimensional hyperbolic systems. *J. Comp. Phys.* **131**, 327–353.

Leveque, R.J., 1998: Balancing source terms and flux gradients in high-resolution Gudonov schemes. The quasi-steady wave-propagation algorithm.. *J. Comp. Phys.* **146**, 346–365.

Phillips, J.C., Lane, S.J., Lejeune, A.M., and Hilton, M., 1995: Gum rosin-acetone system as an analogue to the degassing behaviour of hydrated magmas. *Bull. Volc.* **57**, 263–268.

Shu, C-W., 1997: Essentially non-oscillatory and weighted non-oscillatory schemes for hyperbolic conservation laws. NASA Langley research Center. 77 pp.

Shu, C-W. & Osher, S., 1988: Efficient implementation of essentially non-oscillatory shock-capturing schemes. *J. Comp. Phys.* **7**, 439–471.

Shu, C-W. & Osher, S., 1989: Efficient implementation of essentially non-oscillatory shock-capturing schemes II. *J. Comp. Phys.* **83**, 32–78.

Toro, E.F., 1989: A weighted average flux method for hyperbolic conservation laws. *Proc. Roy. Soc. London, Ser. A* **423**, 401–418.

Whitham, G.B., 1974: *Linear and nonlinear waves.* John Wiley, Toronto, 636 pp.

Wilson, L., Sparks, R.S.J., Huang, C.T., and Watkins, N.D., 1980: Explosive volcanic eruptions, IV, The control of magma properties and conduit geometry on eruption column behaviour. *Geophys. J. R. Astron. Soc.* **63**, 117–148.

Woods, A.W., 1995: The dynamics of explosive volcanic eruptions. *Reviews of Geophysics* **33**, 495–530.

Woods, A.W. and Sparks, S., 1998: *Report on scoping calculations for magma-repository interaction at Yucca mountain.* Centre for Environmental and Geophysical Flows, University of Bristol, Bristol, 24 pp.

Appendix A

Retrieval of package

The whole package of C-programs, header files, and plotting programs is available in a compressed form as magairint.tar.gz. After retrieval it is best to place the file in the directory where you wish to unfold the package. Unfolding the package creates subdirectories. The retrieved tar-file can be uncompressed by the command gzip -d magairint.tar on a LINUX or UNIX machine. The tar file is untarred by the command tar xvf magairint.tar (see man-pages for tar). What will appear is an instruction file magrepint.README, a postscript version of this document magnumrep99.ps, and directories magac with all the C-programs and magamat with all the Matlab-programs.

Table 3.1 summarizes the purpose of all core functions. They define the name of the executable. After inspection of the Makefile (see Appendix B) it will become clear which files belong together.

Appendix B

Makefile

```
#
#       COMPILERS:
#
# sun compile Ultrasparc 2200
CCsun = cc
# gnu compiler on PC's with LINUX and a pentium-II/III processor
CCgbu = gcc
#
#               COMPILER IN USE (change if required)
#
CC = gcc
#
#
#       OPTIMIZATION (consult local machine and manpages)
#
#
CFLAGSsun = -O -fast
CFLAGSgnu = -03
FLAGS = -O
#
#               OPTIMIZATION IN USE (change if required)
#
CFLAGS = -O3
#
#
#       LIBRARIES
#
#
LIBS = -lm
```

```
#
# LLF2 codes for melt-volatiles only:
#
ENCISO1 = enciso1.o enciso1cpu.o enciso1in.o enciso1inpu.o enoiso1ext.o
# Same in Lagrangian framework
ENCLSO1 = encliso1.o enciso1cpu.o enciso1in.o enciso1inpu.o enoiso1ext.o
# Curved pipe, regular grid with gravity for intermediate report
ENCISC1 = encisc1.o enciso1cpu.o enciso1in.o enciso1inpu.o enoiso1ext.o
#
# LLF2 codes for air only:
#
ENCAIR1 = encair1.o enciso1cpu.o enciso1in.o enciso1inpu.o enoiso1ext.o
# Same but in Lagrangian framework
ENCLIR1 = enclair1.o enciso1cpu.o enciso1in.o enciso1inpu.o enoiso1ext.o
#
#
# LLF2 codes for melt-volatiles and air:
#
# Straight pipe regular grid no gravity:
ENCMVA1 = encmva1.o enciso1cpu.o enciso1in.o enciso1inpu.o enoiso1ext.o
# Curved pipe regular grid with gravity
ENCMAX1 = encmax1.o enciso1cpu.o enciso1in.o enciso1inpu.o enoiso1ext.o
# Curved pipe, stretched grid, and gravity
ENCMVS1 = encmvs1.o enciso1cpu.o enciso1in.o enciso1inpu.o enoiso1ext.o
ENCMVT1 = encmvt1.o enciso1cpu.o enciso1in.o enciso1inpu.o enoiso1ext.o
#
#
# ENO code for melt-volatiles only
#
ENOISO1 = enoiso1.o enoiso1cpu.o enoiso1in.o enoiso1inp.o enoiso1ext.o
ENOISO3 = enoiso3.o enoiso1cpu.o enoiso1in.o enoiso1inp.o enoiso1ext.o
#
# ENO code for air only
#
ENOAIR1 = enoair1.o enoiso1cpu.o enoiso1in.o enoiso1inp.o enoiso1ext.o
#
# ENO code for melt-volatiles-air
#
ENOMVA1 = enomva1.o enoiso1cpu.o enoiso1in.o enoiso1inp.o enoiso1ext.o
#
#
#
ECOISC1 = ecoisc1.o enciso1cpu.o enciso1in.o enciso1inpu.o enoiso1ext.o
#
```

```
#

enciso1: $(ENCISO1)
$(CC) $(CFLAGS) $(ENCISO1) $(LIBS) -o enciso1
encliso1: $(ENCLSO1)
$(CC) $(CFLAGS) $(ENCLSO1) $(LIBS) -o encliso1
encisc1: $(ENCISC1)
$(CC) $(CFLAGS) $(ENCISC1) $(LIBS) -o encisc1
encair1: $(ENCAIR1)
$(CC) $(CFLAGS) $(ENCAIR1) $(LIBS) -o encair1
enclair1: $(ENCLIR1)
$(CC) $(CFLAGS) $(ENCLIR1) $(LIBS) -o enclair1
encmva1: $(ENCMVA1)
$(CC) $(CFLAGS) $(ENCMVA1) $(LIBS) -o encmva1
encmax1: $(ENCMAX1)
$(CC) $(CFLAGS) $(ENCMAX1) $(LIBS) -o encmax1
encmvs1: $(ENCMVS1)
$(CC) $(CFLAGS) $(ENCMVS1) $(LIBS) -o encmvs1
encmvt1: $(ENCMVT1)
$(CC) $(CFLAGS) $(ENCMVT1) $(LIBS) -o encmvt1
enoiso1: $(ENOISO1)
$(CC) $(CFLAGS) $(ENOISO1) $(LIBS) -o enoiso1
enoair1: $(ENOAIR1)
$(CC) $(CFLAGS) $(ENOAIR1) $(LIBS) -o enoair1
enomva1: $(ENOMVA1)
$(CC) $(CFLAGS) $(ENOMVA1) $(LIBS) -o enomva1

#
# One-dimensional gas dynamics enc code (2nd-order LLF2).
#
enciso1cpu.o: enciso1cpu.c
$(CC) $(CFLAGS) $(LIBS) -c enciso1cpu.c
enciso1in.o: enciso1in.c
$(CC) $(CFLAGS) $(LIBS) -c enciso1in.c
enciso1inpu.o: enciso1inpu.c
$(CC) $(CFLAGS) $(LIBS) -c enciso1inpu.c
enciso1.o: enciso1.c
$(CC) $(CFLAGS) $(LIBS) -c enciso1.c
encliso1.o: encliso1.c
$(CC) $(CFLAGS) $(LIBS) -c encliso1.c
encisc1: $(ENCISC1)
$(CC) $(CFLAGS) $(ENCISC1) $(LIBS) -o encisc1
ecoair1.o: ecoair1.c
$(CC) $(CFLAGS) $(LIBS) -c ecoair1.c
```

```
encmva1.o: encmva1.c
$(CC) $(CFLAGS) $(LIBS) -c encmva1.c
encmax1.o: encmax1.c
$(CC) $(CFLAGS) $(LIBS) -c encmax1.c
encmvs1.o: encmvs1.c
$(CC) $(CFLAGS) $(LIBS) -c encmvs1.c
encmvt1.o: encmvt1.c
$(CC) $(CFLAGS) $(LIBS) -c encmvt1.c

#
# One-dimensional air dynamics enc code (2nd-order LLF2).
#
encair1.o: encair1.c
$(CC) $(CFLAGS) $(LIBS) -c encair1.c
enclair1.o: enclair1.c
$(CC) $(CFLAGS) $(LIBS) -c enclair1.c

#
# One-dimensional gas dynamics eno code.
#
enoiso1cpu.o: enoiso1cpu.c
$(CC) $(CFLAGS) $(LIBS) -c enoiso1cpu.c
enoiso1in.o: enoiso1in.c
$(CC) $(CFLAGS) $(LIBS) -c enoiso1in.c
enoiso1inp.o: enoiso1inp.c
$(CC) $(CFLAGS) $(LIBS) -c enoiso1inp.c
enoiso1ext.o: enoiso1ext.c
$(CC) $(CFLAGS) $(LIBS) -c enoiso1ext.c
enoiso1.o: enoiso1.c
$(CC) $(CFLAGS) $(LIBS) -c enoiso1.c
enomva1.o: enomva1.c
$(CC) $(CFLAGS) $(LIBS) -c enomva1.c
enoiso3.o: enoiso3.c
$(CC) $(CFLAGS) $(LIBS) -c enoiso3.c
```

Appendix C

Examples

In this appendix, we will step-by-step go through a couple of examples. Once the user is familiar with these examples, the effects of a small parameter change within the same example can be investigated before larger changes are made within the actual program.

Compiling and linking on different compilers and machine may require modifications in: iso1.h, enciso1inpu.c, enciso1in.c, enoiso1inp.c and enoiso1in.c. These programs are short and the changes for several compilers and machines can be found in the comments.

C.1 Example 1: stationary shock in magma

The stationary shock in magma can be simulated with programs enciso1 and enoiso1 and should result in graphs equivalent to Fig. 4.2a,b). The initial conditions are found in the input files enciso1.in and enoiso1.in, respectively. The proper set of parameters can be found under the phrase Numerical tests melt volatile fluid ... or by searching for simulation000 in either of these two input files. Copy, but don't move, these parameter sets to the top of the input file and save. Boundary conditions in both simulations are extrapolating ones; in programs enciso1.c and enoiso1.c, respectively, we have to #define EXTRAP near the top of these programs under the Boundary conditions. Alternatively, we may #define WESTEXTRAP and #define EASTEXTRAP together.

Compile and link the programs on the command line with command: < *prompt* >make enciso1< *return* > or < *prompt* >make enoiso1< *return* >. At this point, the computer may either issue a warning or error message and the IO-treatment my have to be adjusted to fit your machine. The program will run with command < *prompt* >enciso1< *return* > or < *prompt* >enoiso1< *return* >. It is finished within a few seconds on a PC with Pentium-III processor.

Internal error message may now appear, for example because the boundary conditions were specified wrongly as EASTSOLID. But if all proceeds fine, four or five output files will be generated, either e1ce000, e1ct000, e1cq000, e1cd000, e1ci000 or e1se000, e1st000, e1sq000, e1si000 for LLF2 or ENO, respectively. Some of the two output files are not used in the magma simulations. A first check can be made by viewing the energetics in e1ce000 or e1se000. The stationarity implies that energy should stay constant. The parameter set of the simulation is saved in ec000/es000, respectively.

The data format can be found in the program enciso1in.c/enoiso1in.c in the measurement routine Measure() in enciso1.c/enoiso1.c, respectively. At this point, we can use our own graphics package or the provided Matlab-routines. The Matlab-routines do provide insights into the data structure. A Matlab-animation can

be made with isot1bfilm.m. Stack plots, energetics and comparisons with exact solutions can be generated by isot1.m.

C.2 Example 2: shock tube in air

A shock tube in air can be simulated with programs encair1 and enoair1 and the graphs should be equal to Fig. 4.11a,b). The initial conditions are found in the input files enciso1.in and enoiso1.in, respectively. The proper set of parameters can be found under the phrase Air tests or by searching for simulationair011 in either of the two input files. Copy, but don't move, these parameter sets to the top of the input file and save. Boundary conditions in these simulations are extrapolating ones; in programs encair1.c and enoair1.c, respectively, we have to #define EXTRAP near the top of these programs under the Boundary conditions. Alternatively, we may #define WESTEXTRAP and #define EASTEXTRAP together.

Compile and link the programs on the command line with command < *prompt* >make encair1< *return* > or < *prompt* >make enoair1< *return* >. At this point, the computer may either issue a warning or error message and the IO-treatment my have to be adjusted to fit your machine. The program will run with command < *prompt* >encair1< *return* > or < *prompt* >enoair1< *return* >. It is finished within a few seconds on a PC with Pentium-III processor.

Internal error messages may now appear, for example because the boundary conditions were specified wrongly as EASTSOLID. But if all proceeds fine, four or five output files will be generated, either e1ce011, e1ct011, e1cq011, e1cd011, e1ci011 or e1se011, e1st011, e1sq011, e1si011 for LLF2 or ENO, respectively. Not all of these files are used in air simulations. A first check of the program can be made by viewing the energetics in e1ce011 or e1se011. Energy should stay constant as longs as no information has left the domain. The parameter set of the simulation is saved in ec011/es011, respectively.

The data format can be found in the program enciso1in.c/enoiso1in.c in the measurement routine Measure() in encair1.c/enoair1.c, respectively. A Matlab-animation can be made with isoair1bfilm.m. Stack plots, energetics and comparisons with exact solutions can be generated by isoair1.m.

C.3 Example 3: basis simulation run 056

The first basis simulation of magma-repository interactions is simulated with program encmvs1. Graphs should be equivalent to Fig. 5.3. The initial conditions are found in the input file enciso1.in. The proper set of parameters can be found under the phrase Magma-repository interactions or by searching for simulationair056. Copy, but don't move, these sets to the top of the input file and save. Boundary conditions in these simulations are extrapolating ones; in programs encmvs1.c. we have to #define WESTCHAR, #define EASTSOLID and #define INTERFACE near the top of this program under the Boundary conditions.

Compile and link the program on the command line with command < *prompt* >make encmvs1< *return* > or < *prompt* >make enoair1< *return* >. At this point, the computer may either issue a warning or error message and the IO-treatment my have to be adjusted for your machine. The program will run with command < *prompt* >encmvs1< *return* >. It is finished within after a couple of hours on a PC with Pentium-III processor.

If all proceeds fine, four or five output files will be generated, either e1ce056, e1ct056, e1cq056, e1cd056, e1ci056. for LLF2 or ENO. A first check of the program can be made by checking the energetics in e1ce056. Note that the potential energy is dominating, we really should record the *pseudo-energy*. Pseudo energy is an exact combination of an energy and a mass invariant which measures the deviation from a (hydrostatic)

basic state and is a second-order quantity in a small-amplitude perturbation expansion around the basic state (e.g. Shepherd 1993).

The data format can be found in the program enciso1in.c in the measurement routine Measure() in encmvs11.c. A Matlab-animation may be made with isoair1bfilm.m. Stack plots, energetics and comparisons with exact solutions can be generated by mva1.m.

Appendix D

Listings of enciso1.c and enoiso1.c

```
/*************************************************************************
  enciso1.c
  Timestepping routine based on
  CONVEX Essentially Non-Oscillatory (ENC) numerical scheme
  for:

  (i) 1D isothermal  equations of melt and volatiles:
     m1_t  + [m1^2/rho + p]_x = 0, x-momentum equation;
     rho_t +  m1_x            = 0, continuity equation
  with
  m1    = rho u = momentum
  rho   = rho(p) = density
        = [n(p) R T p^{-1} + (1-n(p)/sigma)]^{-1}
  n(p)  = n0 - s sqrt(p) = mass fraction exsolved volatiles
                           (Henry's law)
  n0    = total volatile content of melt = 1-3%
  s     = 3 10^{-6} Pa^{-1/2}
  s     = saturation constant of approximately 3 20^(-6) (sqrt(Pa))^{-1}
  T     = 1000-1200 K = constant temperature
  R     = R d m_v = 462 J (kg K)^{-1}
  m_v   = molecular mass volatiles
  Rs    = 8.31436 J (mole K)^{-1} = gas constant
  sigma= 2500 kg m^{-3} = melt density
  coupled to
  (ii) 1D compressible gas dynamics of air:
  ma1_t  + [ma1^2/rha + p]_x = 0, x-momentum equation;
  rha_t  +  ma1_x            = 0, continuity equation;
  Om_t   + [(Om+p) m1 rho^{-1}]_x = 0, energy equation
  with
  ma1 = rha  ua = momentum in air
  rha = density air
  Om  = rha ( e+ (1/2) ua^2 )
      = p (gamma-1)^{-1}+ ma1^2/rha
      = energy density air
  gamma = cp cv^{-1} = ratio of specific heats
        = approximately 1.4 in air.
  R     = Rs m_ma^{-1} = 287.04 J (kg K)^{-1}
  m_ma = 28.966  = molecular mass air

  ''Nrk''th-order Runge Kutta scheme.

  Matlab plotting program:
```

```
  isot1.m     (line plots of density, pressure,energy and velocity)
  isot1film.m (movie sequence of line plots density, velocity and pressure).
*************************************************************************/

#include <stdio.h>
#include <math.h>
#include <malloc.h>
#include "iso1.h"
#include <stdlib.h>

/*
  Definitions of constants.
*/
#define ONET 1.e0/3.e0
#define WET 1.e-07
#define PI 4.e0*atan(1.e0)

/*
  Equations.
  q[][nqi(,1)] = m1  = rho u, x momentum,
  q[][nqi(,2)] = rho = density,
  q[][nqi(,3)] = 0m  = energy density or pressure.
*/
/*
  Boundary conditions.
#define PERIODIC
#define EXTRAP
#define EASTEXTRAP
#define WESTEXTRAP
#define MIDWESTSOLID
#define MIDEASTSOLID
#define WESTSOLID
#define EASTSOLID
#define WESTCHAR
#define INTERFACE
*/
#define WESTEXTRAP
#define EASTEXTRAP

/*
  Forcing
#define FORCEGRAV
#define FORCEDAMPn1
#define FORCEDAMPn2
*/

/*
  External routines.
*/
extern double *Dvector();
extern void free_Dvector();
extern int *vector();
extern void free_vector();
extern double **Matrix();
extern void free_Matrix();
extern int **iMatrix();
extern void free_iMatrix();
extern double zbrent();
/*
  Files.
*/
FILE *fp1, *fp2, *fp3, *fp6;
/*
INTERNAL Routines
*/
double dmin();
double dmax();
double MIM();
double Dens();
```

```
double Density();
int Min();
int Max();
int npif();
int nqi();
int nri();
void Measure();           /* Routine where measurements are stored in files. */
void Initial_Condition(); /* Routine where arrays allocated and initial conditions set.*/
void freedom();           /* De-allocate or free flexible arrays. */
void BounCond();          /* Boundary conditions. */
void Fluxiso();
void Fluxeul();
void Runge_Kutta();
void Forcing();
void Wavemaker();
int Jb, Niz, Nizp, imin, Ni;
int *kmin;       /* counting array in EnoRoe and EnoLLF.                */
double dx;       /* Grid spacing.                                       */
double dt;       /* Time step.                                          */
double *x;       /* Flux at cell boundaries x_{i+1/2} -> x[i]           */
double **q;      /* Cell averaged flux variable.                        */
double **qt;     /* Intermediate flux variable used in Runge-Kutta steps.*/
double **fhat;   /* Flux at cell boundaries.                            */
double **fi;     /* f(q_i)                                              */
double **fp;     /* f+                                                  */
double **fm;     /* f_                                                  */
double **alpha;  /* f'(u_i)                                             */
double **eval;   /* Entropy violation tell tale.                        */
double **R;      /* Matrix of right eigenvectors at all grid boundaries. */
double **Rm;     /* Matrix of left eigenvectors  at all grid boundaries. */
double *qh;      /* Help variable for Roe speeds at x_(i+1/2).          */
double **hp;     /* Local divided difference tables in LLF.             */
double **hm;     /* Local divided difference tables in LLF.             */
double *xb, *ub, xjbn1, epsil, phib, evall;
double *aph, *apm;
double *Sea0;    /* Specification of incoming wave at seaward boundary.  */
double *eb, *eb0;
double *bou, *ebnew;
double pmin, pmax, toll;
double eps, P0;
/************************************************************************
  Heart of the one-dimensional ENC algorithm.

  ************************************************************************/
void TimeStep(Tskeuze)
     int Tskeuze;
{
  int nnt, nrk, np, ntrue, i, j, k, m;
  double c1, c2, cc, sqhl, sqhr, sqh, du, dd, fpmax;
  double x1, x2, xs, P1, P2, qbm, qbh;
/************************************************************************
    Start timestepping routine.
************************************************************************/
  toll = 1.e-8;
  P0   = rcoef[0]*rcoef[1]*rcoef[2]*rcoef[7];
  eps  = rcoef[4]*sqrt(P0)/rcoef[0];
  pmin = 1.e3/P0;
  rcoef[3]/=rcoef[7];
  /*
    Maximum pressure is given by critical pressure for which
```

```
    the speed of sound becomes infinite.
  */
  du = rcoef[3]/rcoef[0];
  if (rcoef[4] > 0.e0) dd = 2.e0*du/eps;
  else
    {
      fprintf(stderr," Saturation constant s<=0. Abort.\n");
      exit(1);
    }
  dd = 108.e0*dd+12.e0*sqrt(12.e0*du*du*du+81.e0*dd*dd);
  dd = pow(dd,1.e0/3.e0);
  pmax = (dd/6.e0-2.e0*du/dd);
  pmax*=pmax;
  /*
    CHANGE
    pmin = 0.01e0;
    pmax = 100.e0;
  */
  if ( (fp6 = fopen("enciso1xu", "w")) == NULL)
    {
      fprintf(stderr, "\n Error: can not open testsw. \n");
      fprintf(stderr, "\n Function: enciso1.c. \n");
      exit(1);
    }
  /*
    Initialize variables. Initial conditions.
  */
  Initial_Condition(Tskeuze, &Ni);
  /*
    Store initial conditions.
  */
  Measure(0, Ni);
#ifdef INTERFACE
  fprintf(fp6, "%g %g\n", xb[0], ub[0]);
#endif
  fpmax = 0.e0;
  fprintf(stderr, " Measure ic's done.\n");
  /*
    Start main timestepping routine.
    Counter nnt.
  */
  for (nnt = 1; nnt <= Nt; nnt++)
    {
      /*
Runge-Kutta predictor-corrector step.
Counter nrk.
      */
      for (nrk = 0; nrk <= 1; nrk++)
{
  evall = 0.e0;
  /*
    Boundary conditions.
  */
  BounCond(nrk);
  /*
    Compute shallow-water Roe speeds,
    eigenvalues, and alpha's.
  */
  imin = 0;
  Ni  = Npxts;
  Niz = Ni-1;
  /*
    Compute max |f'(q)|, fi=f_i
    for convex eno LLF.
```

```
  */
  for (i=-1; i<=Ni; i++)
    {
      /*
Definition alpha's used in LLF.
        Find p from rho(p).
        Calculate cc=a(p)=(drho/dp)^{-1/2}.
      */
      cc=q[i][nqi(nrk,2)];
      c1=zbrent(Dens,cc,pmin,pmax,toll*pmin);
      c2=c1;
      if (c1 > pmax)
{
  fprintf(stderr," Pressure > pmax. \n");
  exit(1);
}
      cc=(cc*cc/(c1*c1))*(1.e0-0.5e0*eps*sqrt(c1)-0.5e0*rcoef[0]*eps*c1*sqrt(c1)/rcoef[3]);
      cc = sqrt(1.e0/cc);
      /*
Change below
cc = sqrt(q[i][nqi(nrk,2)]);
Change above
      */
      c1 = q[i][nqi(nrk,1)]/q[i][nqi(nrk,2)];
      /*
fprintf(stderr," i:%d cc:%g c2:%g",i,cc, c2);
Definition eval=f'(u) in component form:
eigenvalues.
      */
      eval[i][1] = c1-cc;
      eval[i][2] = c1+cc;
      eval[i][3] = c1;
      du = dmax(fabs(eval[i][1]),fabs(c1));
      fpmax = dmax( dmax( du,fabs(eval[i][2]) ), fpmax);
      eval[i][1] = dmax(fabs(eval[i][1]),fabs(eval[i][2]));
      eval[i][1] = dmax(eval[i][1],fabs(eval[i][3]));
      evall = dmax(eval[i][1],evall);
    }
  for (i=-1; i<=Niz; i++)
    {
      alpha[i][1]=dmax(fabs(eval[i][1]),fabs(eval[i+1][1]));
    }
  for (i=-2; i<=Niz+2; i++)
    {
      fi[i][1]  = zbrent(Dens,q[i][nqi(nrk,2)],pmin,pmax,toll*pmin);
      /*
if( fabs(fi[i][1]-0.5e0*q[i][nqi(nrk,2)]*
q[i][nqi(nrk,2)])> 0.00001e0)
{
fprintf(stderr, " abort %g %g ",
fi[i][1],0.5e0*q[i][nqi(nrk,2)]*q[i][nqi(nrk,2)]);
exit(1);
}
      */
      if (q[i][nqi(nrk,2)] < WET)
{
  fi[i][1]  = 0.0e0;
  fi[i][2]  = 0.0e0;
  fi[i][3]  = 0.0e0;
}
      else
{
  fi[i][1] += q[i][nqi(nrk,1)]*q[i][nqi(nrk,1)]/q[i][nqi(nrk,2)];
  fi[i][2]  = q[i][nqi(nrk,1)];
```

```
  fi[i][3]  = 0.e0;
}
    }
  /*
    Calculate second order LLF flux fhat[i] at cell edge
    i+1/2 between cell i and i+1.
    f_(i+1/2) = fhat[i].
  */
  for (i=-1; i<=Niz; i++) /* Niz = Ni-3 if ZEROH1 otherwise Ni-1. */
    {
      for (np=1; np<=Nsy; np++)
{
  fp[i][np] = 0.5e0*(fi[i][np]+alpha[i][1]*q[i][nqi(nrk,np)]);
  du  = fi[i][np]-fi[i-1][np];
  du += alpha[i][1]*(q[i][nqi(nrk,np)]-q[i-1][nqi(nrk,np)]);
  dd  = fi[i+1][np]-fi[i][np];
  dd += alpha[i][1]*(q[i+1][nqi(nrk,np)]-q[i][nqi(nrk,np)]);
  fp[i][np] += 0.25e0*MIM(dd,du);
  fm[i][np]  = 0.5e0*(fi[i+1][np]-alpha[i][1]*q[i+1][nqi(nrk,np)]);
  du  = fi[i+1][np]-fi[i][np];
  du -= alpha[i][1]*(q[i+1][nqi(nrk,np)]-q[i][nqi(nrk,np)]);
  dd  = fi[i+2][np]-fi[i+1][np];
  dd -= alpha[i][1]*(q[i+2][nqi(nrk,np)]-q[i+1][nqi(nrk,np)]);
  fm[i][np]  -= 0.25e0*MIM(dd,du);
  fhat[i][np] = fp[i][np]+fm[i][np];
}
    }
  /*
    f[i-1] -> f_(i-1/2)
    For i = 0 one finds f[-1] -> f_(-1/2), i.e. at the left boundary edge.
    */
  for (i=imin; i<=Niz; i++)
    {
      for (np = 1; np<=Nsy; np++)
{
  /* Note that the peculiar minus sign comes from the definition
     in Shu and Osher's and Shu's papers. */
  qt[i][np] = -(fhat[i][np]-fhat[i-1][np])/(x[i]-x[i-1]);
}
    }
  /*
    Forcing
  Forcing(nrk, Niz);*/
  /*
    Second-order Runge-Kutta.
  */
  if (nrk == 0)
    {
      for (i=imin; i<=Niz; i++)
{
  for (np=1; np<=Nsy; np++)
    {
      q[i][nqi(1,np)] = q[i][nqi(0,np)]+dt*qt[i][np];
    }
}
#ifdef INTERFACE
      xb[1] = xb[0]+dt*ub[0];
      ub[1] = ub[0]-dt*(aph[1]+2.e0*aph[2]*(xb[0]-xjbn1));
#endif
    }
  else
    {
      for (i=imin; i<=Niz; i++)
```

```
{
  for (np = 1; np<=Nsy; np++)
    {
      q[i][nqi(0,np)]  = 0.5e0*(q[i][nqi(0,np)]+q[i][nqi(1,np)]);
      q[i][nqi(0,np)] += 0.5e0*dt*qt[i][np];
    }
}
#ifdef INTERFACE
      xb[0] = 0.5e0*(xb[0]+xb[1]+dt*ub[1]);
      ub[0] = 0.5e0*(ub[0]+ub[1]-dt*(aph[1]+2.e0*aph[2]*(xb[1]-xjbn1)));
#endif
    }

}
      for (i=0; i<=(Ni-1); i++)
{
   if (q[i][nqi(0,2)] < WET)
    {
      for (np=1; np<=Nsy; np++) q[i][nqi(0,np)] = 0.e0;
    }
}
      if ((nnt % Nmeas) == 0)
{
  Measure(nnt, Ni);
}
    }
  fprintf(stderr, " Time and rk loop completed. \n");
   /* Free matrix and array allocation. */
  freedom();
  fprintf(stderr, " Allocated space free. \n");
}
/* End timestepping routine. */

/*************************************************************
  Definition arrays, initial_Conditions and
  initialization cell boundaries.
  *************************************************************/
void Initial_Condition(Tk, Nii)
  int Tk, *Nii;
{
  int i, j, m, n, nrk;
  double cc, h1, h2, p1, p2, u1, u2, x0, x1, x2, ls;
  double fpmax;
  /* Array definitions. */
  kmin  = vector(0, Nor+1);
  x     = Dvector(-(Nor+1), Npxts+1);
  qh    = Dvector(1, Nsy);
  xb    = Dvector(0, Nrk-1);
  ub    = Dvector(0, Nrk-1);
  aph   = Dvector(0, 3);
  apm   = Dvector(0, 3);
  Sea0  = Dvector(0, Nsy);
  bou   = Dvector(0, Nsy);
  eb    = Dvector(0, Nsy);
  eb0   = Dvector(0, Nsy);
  ebnew = Dvector(0, Nsy);
  q     = Matrix(-(Nor+1), Npxts+Nor+1, 1, Nrk*Nsy);
  qt    = Matrix(0, Npxts, 1, Nsy);
  fhat  = Matrix(-1, Npxts, 1, Nsy);
  fi    = Matrix(-2, Npxts+2, 1, Nsy);
  fp    = Matrix(-1, Npxts, 1, Nsy);
  fm    = Matrix(-1, Npxts, 1, Nsy);
```

```
  alpha = Matrix(-1, Npxts, 1, Nsy);
  eval  = Matrix(-1, Npxts, 1, Nsy);
  R     = Matrix(-1, Npxts, 1, Nsy*Nsy);
  Rm    = Matrix(-1, Npxts, 1, Nsy*Nsy);
  hp    = Matrix(-(Nor+1), 0, 1, (Nor+1)); /* 1, Nor+1, 1, Nsy */
  hm    = Matrix(-(Nor+1), 0, 1, (Nor+1)); /* 1, Nor+1, 1, Nsy */
  /* Cell boundaries array. */
  dx = Lx/(double)(Npxts);
  for (i=-2; i<=Npxts+1; i++)
    {
      /*
The zeroth cell has right boundary at dx
when starting at x = 0.
*/
      x[i] = (double)(i)*dx+dx;
    }
  /*
    Set time step given cfl.
  */
  dt = dx*cfl;
  /*
    Define boundary conditions.
  */
  /*
    Print garb.
  */
#ifdef EASTEXTRAP
  fprintf(stderr, " EASTEXTRAP ");
#endif
#ifdef WESTSOLID
  fprintf(stderr, " WESTSOLID ");
#endif
#ifdef EXTRAP
  fprintf(stderr, " EXTRAP ");
#endif
#ifdef INTERFACE
  fprintf(stderr, " INTERFACE ");
#endif
#ifdef WESTCHAR
  fprintf(stderr, " WESTCHAR ");
#endif
#ifdef PERIODIC
  fprintf(stderr, " PERIODIC ");
#endif
  fprintf(stderr, "\n");
  /* Steady shock and shock tube test problems. */
  x0 = 0.5e0*Lx;
  /* Initial conditions. */
  for (i = 0; i<=(Npxts-1); i++)
    {
      /*
x[i] denotes right cell boundary, hence midpoint
of cell is 0.5*(x[i]-x[i-1]) = 0.5*dx to left,
latter equality is for regular grid.
      */
      switch (icno)
{
case 0:
  {
    /* Isothermal stationary shock.
    h1 = 100.e0/rcoef[7];
    h2 = 600.e0/rcoef[7];
    p1 = zbrent(Dens,h1,pmin,pmax,toll*pmin);
    p2 = zbrent(Dens,h2,pmin,pmax,toll*pmin);
    */
```

```
    p2 = 5.e06/P0;
    p1 = 1.e05/P0;
    h1 = Density(p1);
    h2 = Density(p2);
    u1 = -sqrt(h2*(p2-p1)/(h1*(h2-h1)));
    u2 = u1*h1/h2;
    if (i==1)
      {
fprintf(stderr,"p1=%g p2=%g u1=%g u2=%g \n",p1, p2, u1, u2);
fprintf(stderr,"r1=%g r2=%g  \n",h1, h2);
      }
    if ((x[i]-0.5e0*dx) < x0)
      {
q[i][nqi(0,2)] = h2;
q[i][nqi(0,1)] = h2*u2;
q[i][nqi(0,3)] = 0.e0;
      }
    else
      {
if ((x[i]-0.5e0*dx) == x0)
  {
    q[i][nqi(0,2)] = 0.5e0*(h1+h2);
    q[i][nqi(0,1)] = q[i][nqi(0,2)]*0.5e0*(u1+u2);
    q[i][nqi(0,3)] = 0.e0;
  }
else
  {
    q[i][nqi(0,2)] = h1;
    q[i][nqi(0,1)] = h1*u1;
    q[i][nqi(0,3)] = 0.e0;
  }
      }
  } break;
case 1:
  {
    /* Isothermal shock tube.
    h1 = 100e0/rcoef[7];
    h2 = 600.e0/rcoef[7];
    p1 = zbrent(Dens,h1,pmin,pmax,toll*pmin);
    p2 = zbrent(Dens,h2,pmin,pmax,toll*pmin);*/
    p2 = 5.e06/(rcoef[0]*rcoef[1]*rcoef[2]*rcoef[7]);
    p1 = 1.e05/(rcoef[0]*rcoef[1]*rcoef[2]*rcoef[7]);
    h1 = Density(p1);
    h2 = Density(p2);
    u1 = u2 = 0.e0;
    if ((x[i]-0.5e0*dx) < x0)
      {
q[i][nqi(0,2)] = h2;
q[i][nqi(0,1)] = h2*u2;
q[i][nqi(0,3)] = 0.e0;
      }
    else
      {
if ((x[i]-0.5e0*dx) == x0)
  {
    q[i][nqi(0,2)] = 0.5e0*(h1+h2);
    q[i][nqi(0,1)] = q[i][nqi(0,2)]*0.5e0*(u1+u2);
    q[i][nqi(0,3)] = 0.e0;
  }
else
  {
    q[i][nqi(0,2)] = h1;
    q[i][nqi(0,1)] = h1*u1;
    q[i][nqi(0,3)] = 0.e0;
  }
```

```
      }
  } break;
case 3:
  {
    /*
      Isothermal moving shock.
    */
    p2 = 2.e06/(rcoef[0]*rcoef[1]*rcoef[2]*rcoef[7]);
    p1 = 1.e06/(rcoef[0]*rcoef[1]*rcoef[2]*rcoef[7]);
    h1 = Density(p1);
    h2 = Density(p2);
    u2 = (1.e0-h1/h2)*sqrt(h2*(p2-p1)/(h1*(h2-h1)));
    u1 = 0.e0;
    if (i==1)
      {
fprintf(stderr,"p1=%g p2=%g u1=%g u2=%g \n",p1, p2, u1, u2);
fprintf(stderr,"r1=%g r2=%g  \n",h1, h2);
      }
    if ((x[i]-0.5e0*dx) < x0)
      {
q[i][nqi(0,2)] = h2;
q[i][nqi(0,1)] = h2*u2;
q[i][nqi(0,3)] = 0.e0;
      }
    else
      {
if ((x[i]-0.5e0*dx) == x0)
  {
    q[i][nqi(0,2)] = 0.5e0*(h1+h2);
    q[i][nqi(0,1)] = q[i][nqi(0,2)]*0.5e0*(u1+u2);
    q[i][nqi(0,3)] = 0.e0;
  }
else
  {
    q[i][nqi(0,2)] = h1;
    q[i][nqi(0,1)] = h1*u1;
    q[i][nqi(0,3)] = 0.e0;
  }
      }
  } break;
case 8:
  {
    /* Isothermal shock tube. */
    p2 = 5.e06/(rcoef[0]*rcoef[1]*rcoef[2]*rcoef[7]);
    p1 = 1.e05/(rcoef[0]*rcoef[1]*rcoef[2]*rcoef[7]);
    h1 = Density(p1);
    h2 = Density(p2);
    if (i == 1)
      {
fprintf(stderr,"%g %g %g %g \n", p1, p2, h1, h2);
      }
    u1 = u2 = 0.e0;
    if ((x[i]-0.5e0*dx) < x0)
      {
q[i][nqi(0,2)] = h2;
q[i][nqi(0,1)] = h2*u2;
q[i][nqi(0,3)] = 0.e0;
      }
    else
      {
if ((x[i]-0.5e0*dx) == x0)
  {
    q[i][nqi(0,2)] = 0.5e0*(h1+h2);
    q[i][nqi(0,1)] = q[i][nqi(0,3)]*0.5e0*(u1+u2);
    q[i][nqi(0,3)] = 0.e0;
  }
```

```
else
  {
    q[i][nqi(0,2)] = h1;
    q[i][nqi(0,1)] = h1*u1;
    q[i][nqi(0,3)] = 0.e0;
  }
     }
  } break;
case 4:
  {
    /* Isothermal shock tube with gravity term. */
    p2 = 5.e06/(rcoef[0]*rcoef[1]*rcoef[2]*rcoef[7]);
    p1 = 1.e05/(rcoef[0]*rcoef[1]*rcoef[2]*rcoef[7]);
    h1 = Density(p1);
    h2 = Density(p2);
    if (i == 1)
      {
fprintf(stderr,"%g %g %g %g \n", p1, p2, h1, h2);
      }
    u1 = u2 = 0.e0;
    if ((x[i]-0.5e0*dx) < x0)
      {
q[i][nqi(0,2)] = h2;
q[i][nqi(0,1)] = h2*u2;
q[i][nqi(0,3)] = 0.e0;
      }
    else
      {
if ((x[i]-0.5e0*dx) == x0)
  {
    q[i][nqi(0,2)] = 0.5e0*(h1+h2);
    q[i][nqi(0,1)] = q[i][nqi(0,3)]*0.5e0*(u1+u2);
    q[i][nqi(0,3)] = 0.e0;
  }
else
  {
    q[i][nqi(0,2)] = h1;
    q[i][nqi(0,1)] = h1*u1;
    q[i][nqi(0,3)] = 0.e0;
  }
     }
  } break;
default:
  {
    fprintf(stderr," Try again, invalid choice of initial conditions.");
    exit (1);
  } break;
}
     }
  *Nii = Npxts;
  return;
}

/*
  Free allocated memory of arrays and matrices.
*/
void freedom()
{
  fclose(fp6);
  free_vector(kmin, 0, Nor+1);
  free_Dvector(rcoef,0,Nrcoef);
  free_Dvector(x, -(Nor+1), Npxts+1);
  free_Dvector(qh, 1, Nsy);
  free_Dvector(xb, 0, Nrk-1);
  free_Dvector(ub, 0, Nrk-1);
  free_Dvector(aph, 0, 3);
```

```
  free_Dvector(apm, 0, 3);
  free_Dvector(Sea0, 0, 3);
  free_Dvector(bou, 0, 3);
  free_Dvector(ebnew, 0, 3);
  free_Dvector(eb, 0, 3);
  free_Dvector(eb0, 0, 3);
  free_Matrix(q, -(Nor+1), Npxts+Nor+1, 1, Nsy*Nrk);
  free_Matrix(qt, 0, Npxts, 1, Nsy);
  free_Matrix(fhat, -1, Npxts, 1, Nsy);
  free_Matrix(fi, -2, Npxts+2, 1, Nsy);
  free_Matrix(fp, -1, Npxts, 1, Nsy);
  free_Matrix(fm, -1, Npxts, 1, Nsy);
  free_Matrix(alpha, -1, Npxts, 1, Nsy);
  free_Matrix(eval, -1, Npxts, 1, Nsy);
  free_Matrix(R, -1, Npxts, 1, Nsy*Nsy);
  free_Matrix(Rm, -1, Npxts, 1, Nsy*Nsy);
  free_Matrix(hp, -(Nor+1), 0, 1, (Nor+1));
  free_Matrix(hm, -(Nor+1), 0, 1, (Nor+1));
  return;
}

void Wavemaker(h, u, tijd)
     double *h, *u, tijd;
{
  double Am;
  Am = 0.5e0;
  *h = 1.e0+Am*sin(12.e0*PI*tijd);
  *u = Am*sin(12.e0*PI*tijd);
  return;
}
void Fluxiso(nrk, fpmax)
     int nrk;
     double *fpmax;
{
  return;
}

/*
  Second-order Runge-Kutta.
*/
void Runge_Kutta(nk)
     int nk;
{
  int i, j, k, np;
  if (nk == 0)
    {
      for (i=imin; i<=Niz; i++)
{
  for (np=1; np<=Nsy; np++)
    {
      q[i][nqi(1,np)] = q[i][nqi(0,np)]+dt*qt[i][np];
    }
}
    }
  else
    {
      for (i=imin; i<=Niz; i++)
{
  for (np = 1; np<=Nsy; np++)
    {
      q[i][nqi(0,np)]  = 0.5e0*(q[i][nqi(0,np)]+q[i][nqi(1,np)]);
      q[i][nqi(0,np)] += 0.5e0*dt*qt[i][np];
    }
}
```

```
    }
  return;
}
/*
  Forcing terms.
*/
void Forcing(nk, Nz)
     int nk, Nz;
{
  int i;
#ifdef FORCEDAMPn1
  for (i=0; i<=(Nz-1); i++)
    {
      qt[i][1] -= rcoef[6]*q[i][nqi(nk,1)];
    }
#endif
  return;
}

void BounCond(nrk)
  int nrk;
{
  int i, j, k, np;
  double epsilon, h1, h2, u2, cc, uvel, vvel;
  epsilon = 1.0e-10;
#ifdef PERIODIC
  /*
    Periodic boundary conditions at left and right boundary.
   */
  for (i=-(Nor+1); i<=-1; i++)
    {
      for (np=1; np<=Nsy; np++) q[Npxts-i-1][nqi(nrk,np)]=q[-i-1][nqi(nrk,np)];
      for (np=1; np<=Nsy; np++) q[i][nqi(nrk,np)]=q[i+Npxts][nqi(nrk,np)];
    }
#endif
#ifdef SOLID
  /*
    No normal flow boundary conditions for nonrotating case.
  */
  for (i=Npxts; i<=Npxts+Nor; i++)
    {
      q[i][nqi(nrk,1)]=-q[2*Npxts-i-1][nqi(nrk,1)];
      q[i][nqi(nrk,2)]= q[2*Npxts-i-1][nqi(nrk,2)];
      q[i][nqi(nrk,3)]= q[2*Npxts-i-1][nqi(nrk,3)];
    }
  for (i=-(Nor+1); i<=-1; i++)
    {
      q[i][nqi(nrk,1)]=-q[-i-1][nqi(nrk,1)];
      q[i][nqi(nrk,2)]= q[-i-1][nqi(nrk,2)];
      q[i][nqi(nrk,3)]= q[-i-1][nqi(nrk,3)];
    }
#endif
#ifdef EASTSOLID
  /*
    No normal flow boundary conditions for nonrotating case
    at east/right boundary.
  */
  for (i=Npxts; i<=Npxts+Nor; i++)
    {
      q[i][nqi(nrk,1)]=-q[2*Npxts-i-1][nqi(nrk,1)];
      q[i][nqi(nrk,2)]= q[2*Npxts-i-1][nqi(nrk,2)];
      q[i][nqi(nrk,3)]= q[2*Npxts-i-1][nqi(nrk,3)];
    }
#endif
#ifdef WESTSOLID
```

```
  /*
    No normal flow boundary conditions for nonrotating case
    at west/left wall.
    */
  for (i=-(Nor+1); i<=-1; i++)
    {
      q[i][nqi(nrk,1)]=-q[-i-1][nqi(nrk,1)];
      q[i][nqi(nrk,2)]= q[-i-1][nqi(nrk,2)];
      q[i][nqi(nrk,3)]= q[-i-1][nqi(nrk,3)];
    }
#endif
#ifdef EASTEXTRAP
  /*
    Extrapolating boundary conditions on right/east boundary.
    Make extrapolating boundary conditions fourth order for uniform grid.
    */
  for (i=Npxts; i<=(Npxts+Nor); i++)
    {
      for (np=1; np<=Nsy; np++) q[i][nqi(nrk,np)]=q[Npxts-1][nqi(nrk,np)];
    }
#endif
#ifdef WESTEXTRAP
  /*
    Extrapolating boundary conditions on left/west boundary.
    Make extrapolating boundary conditions fourth order for uniform grid.
  */
  for (i=-(Nor+1); i<=-1; i++)
    {
      for (np=1; np<=Nsy; np++) q[i][nqi(nrk,np)]=q[0][nqi(nrk,np)];
    }
#endif
#ifdef EXTRAP
  /*
    Extrapolating boundary conditions on left and right boundary.
    Make extrapolating boundary conditions fourth order for uniform grid.
  */
  for (i=Npxts; i<=(Npxts+Nor); i++)
    {
      for (np=1; np<=Nsy; np++) q[i][nqi(nrk,np)]=q[Npxts-1][nqi(nrk,np)];
    }
  for (i=-(Nor+1); i<=-1; i++)
    {
      for (np=1; np<=Nsy; np++) q[i][nqi(nrk,np)]=q[0][nqi(nrk,np)];
    }
#endif
}

double MIM(al,be)
     double al, be;
{
  if (al*be > 0.e0)
    {
      if (al > 0) return dmin(fabs(al),fabs(be));
      else return -dmin(fabs(al),fabs(be));
    }
    else return 0.e0;
  /*return (fabs(al) <= fabs(be) ? al:be); */
}

double Dens(pp, d0)
     double pp, d0;
{
  double du;

  du = 1.e0-eps*sqrt(pp);
  du = du/pp+(1.e0-rcoef[0]*du)/rcoef[3];
  return (d0 - 1.e0/du);
}
```

```
double Density(pp)
     double pp;
{
  double du;
  du = 1.e0-eps*sqrt(pp);
  du = du/pp+(1.e0-rcoef[0]*du)/rcoef[3];
  return 1.e0/du;
}

/*****************************************************************
  Measure(ntime) subroutine for measurements at integer time ntime.
*****************************************************************/
void Measure(ntime, Ni)
  int ntime, Ni;
{
  int i;
  double ke, pe, pp, p0, du;
  fp2 = fopen(Ts1, "a");
  fp3 = fopen(Ts2, "a");
  fprintf(fp2," %16.10lf %d \n", ntime*dt, Npxts);
  ke   = 0.e0;
  pe   = 0.e0;
  p0   = 1.e0; /* Dimensionally P0. */
  if (ntime == 0) fprintf(stderr, " %d Ni = %d %g \n", ntime, Ni,x[Ni]);
  /* Sum to obtain integral of energy density for inviscid case. */
  for (i=0; i<=(Npxts-1);i++)
    {
      pp = zbrent(Dens,q[i][nqi(0,2)],pmin,pmax,toll*pmin);
      fprintf(fp2," %16.10lf %16.10lf ",x[i]-0.5e0*dx,q[i][nqi(0,1)]);
      fprintf(fp2," %16.10lf %16.10lf \n",q[i][nqi(0,2)],pp);
      /* Kinetic component. */
      ke+= 0.5e0*q[i][nqi(0,1)]*q[i][nqi(0,1)]/q[i][nqi(0,2)]; /* (1/2) m^2/rho */
      /* Potential energy. */
      du = log(pp/p0)+2.e0*eps*(sqrt(p0)-sqrt(pp));
      du+= ((1.e0-rcoef[0])*(pp-p0)+2.e0*ONET*eps*
rcoef[0]*(pow(pp,1.5e0)-pow(p0,1.5e0)))/rcoef[3]-pp/q[i][nqi(0,2)];
      pe+= du*q[i][nqi(0,2)];
    }
#ifdef PERIODIC
  i = 0;
  pp = zbrent(Dens,q[i][nqi(0,2)],pmin,pmax,toll*pmin);
  fprintf(fp2," %16.10lf %16.10lf",x[Npxts]-0.5e0*dx,q[i][nqi(0,1)]);
  fprintf(fp2," %16.10lf %16.10lf\n",q[i][nqi(0,2)],pp);
#endif
#ifdef INTERFACE
  fprintf(fp2," %16.10lf %16.10lf",0.5e0*(x[Jb]+xb[0]),q[Jb][0]);
  fprintf(fp2," %16.10lf %16.10lf\n",xb[0],ub[0]);
#endif
  fprintf(fp3," %16.10lf %16.10lf %16.10lf %16.10lf\n", ntime*dt,(pe+ke)*dx,pe*dx,ke*dx);
  fclose(fp2);
  fclose(fp3);
  return;
}

/*********************************************************
Function Max(,) determines the maximum of
two integer values and returns the biggest.
*/
int Max(k, l) int k, l;
{
return (k > l ? k:l);
```

```
}

/*********************************************************
Function Min(,) determines the minimum of
two integer values and returns the smallest.
*/
int Min(k, l) int k, l;
{
return (k < l ? k:l);
}
/*********************************************************
Function dmax(,) determines the maximum of
two double values and returns the biggest.
*/
double dmax(dk, dl) double dk, dl;
{
return (dk > dl ? dk:dl);
}

/*********************************************************
Function dmin(,) determines the minimum of
two integer values and returns the smallest.
*/
double dmin(dk, dl) double dk, dl;
{
return (dk < dl ? dk:dl);
}

/*********************************************************
  kk = 1, ..., Nor+1
  pp = 1, ..., Nsy
 */
int npif(nkk, pp)
  int nkk, pp;
{
  return ((nkk-1)*Nsy + pp);
}
/*********************************************************
  nkk= 0,1, .., Nrk-1
  pp = 1,2,..., Nsy
 */
int nqi(nkk, pp)
  int nkk, pp;
{
  return (nkk*Nsy + pp);
}
/*********************************************************
 */
int nri(nkk, pp)
  int nkk, pp;
{
  return ((nkk-1)*Nsy + pp);
}
```

```
/*************************************************************************
  enoiso1.c
  Timestepping routine based on
  Essentially Non-Oscillatory numerical scheme
  for:

  (i) 1D isothermal  equations of melt and volatiles:
     m1_t  + [m1^2/rho + p]_x = 0, x-momentum equation;
     rho_t +  m1_x          = 0, continuity equation
  with
  m1   = rho u = momentum
  rho  = rho(p) = density
       = [n(p) R T p^{-1} + (1-n(p)/sigma)]^{-1}
  n(p) = n0 - s sqrt(p) = mass fraction exsolved volatiles
                          (Henry's law)
  n0   = total volatile content of melt = 1-3%
  s    = saturation constant of approximately 3 20^(-6) (sqrt(Pa))^{-1}
  T    = 1000-1200 K = constant temperature
  R    = R d m_v = 462 J (kg K)^{-1}
  m_v  = molecular mass volatiles
  Rs   = 8.31436 J (mole K)^{-1} = gas constant
  sigma= 2500 kg m^{-3} = melt density
  coupled to
  (ii) 1D compressible gas dynamics of air:
  ma1_t  + [ma1^2/rha + p]_x = 0, x-momentum equation;
  rha_t  +  ma1_x            = 0, continuity equation;
  Om_t   + [(Om+p) m1 rho^{-1}]_x = 0, energy equation
  with
  ma1 = rha  ua = momentum in air
  rha = density air
  Om  = rha ( e+ (1/2) ua^2 )
      = p (gamma-1)^{-1}+ ma1^2/rha
      = energy density air
  gamma = cp cv^{-1} = ratio of specific heats
        = approximately 1.4 in air.
  R    = Rs m_ma^{-1} = 287.04 J (kg K)^{-1}
  m_ma = 28.966  = molecular mass air
  Includes generating-radiation boundary conditions.

  ''Nrk''th-order Runge Kutta scheme.
  isot1.m
  isot1film.m
  ciso1.m     (line plots of density, pressure,energy and velocity)
  ciso1film.m (movie sequence of line plots density, velocity and pressure).

*************************************************************************/
#include <stdio.h>
#include <math.h>
#include <malloc.h>
#include "iso1.h"
#include <stdlib.h>

/*
  Definition constants.
  XPOR: porous bed damping on height field for x > XPOR.
  CPOR: coefficient porous bed damping on height field.
  CSET: particle settling damping on velocity field.
  VS:   particle settling velocity on height field.
*/
#define ONET 1.e0/3.e0
#define WET 1.e-07
```

```
#define PI atan(1.e0)
#define XPOR 14.e0
#define CPOR 0.7e0
#define CSET 0.05e0
#define VS 0.05e0

/*
  Definition equations.
  q[][nqi(,1)] = m1  = rho u, x momentum,
  q[][nqi(,2)] = rho = density,
  q[][nqi(,3)] = Om  = energy density.
*/

/*
  Boundary conditions.
#define PERIODIC
#define EXTRAP
#define EASTEXTRAP
#define WESTEXTRAP
#define MIDWESTSOLID
#define MIDEASTSOLID
#define WESTSOLID
#define EASTSOLID
#define WESTCHAR
#define INTERFACE
*/
#define EASTEXTRAP
#define WESTEXTRAP


/*
  Forcing
#define FORCEGRAV
#define FORCEDAMPn1
#define FORCEDAMPn2
*/

/*
  External routines.
*/
extern double *Dvector();
extern void free_Dvector();
extern int *vector();
extern void free_vector();
extern double **Matrix();
extern void free_Matrix();
extern int **iMatrix();
extern void free_iMatrix();
extern double zbrent();

/*
  Files.
*/
FILE *fp1, *fp2, *fp3, *fp6;

/*
INTERNAL routines.
*/
double dmin();
double dmax();
double Dens();
double Density();
int Min();
int Max();
int npif();
int nqi();
int nri();

void Measure();           /* Routine where measurements are stored in files. */
void Initial_Condition(); /* Routine where arrays and initial conditions are allocated.*/
void freedom();           /* De-allocate or free flexible arrays. */
void Setcoeff();          /* Define ENO coefficient, either for regular/irregular grid.*/
```

```
void EnoRoe();            /* Eno-Roe routine shock-capturing. */
void EnoLLF();            /* Eno-LLF routine which takes over Eno-Roe
     in case of entropy violation. */
void BounCond();          /* Boundary conditions. */
void Fluxiso();
void Fluxeul();
void Forcing();
void Runge_Kutta();
void Wavemaker();
/*
  Program global variables.
*/
int Jb, Ni, Niz, imin;
int *kmin;        /* counting array in EnoRoe and EnoLLF.                */
double dx;        /* Grid spacing.                                       */
double dt;        /* Time step.                                          */
double *x;        /* Flux at cell boundaries x_{i+1/2} -> x[i]           */
double **q;       /* Cell averaged flux variable.                        */
double **qt;      /* Intermediate flux variable used in Runge-Kutta steps.*/
double **fhat;    /* Flux at cell boundaries.                            */
double **ftat;    /* Flux at cell boundaries.                            */
double **coeff;   /* Coefficients used in Newton polynomial.             */
double **roe;     /* Roe speed at cell boundaries.                       */
double **alpha;   /* f'(u_i)                                             */
double **eval;    /* Entropy violation tell tale.                        */
double **R;       /* Matrix of right eigenvectors at all grid boundaries. */
double **Rm;      /* Matrix of left eigenvectors  at all grid boundaries. */
double *qh;       /* Help variable for Roe speeds at x_(i+1/2).          */
double **hp;      /* Local divided difference tables in LLF.             */
double **hm;      /* Local divided difference tables in LLF.             */
double **qtiff;
double **ftiff;
double *xb, *ub, xjbn1, epsil, phib;
double *aph, *apm;
double *Sea0;     /* Specification of incoming wave at seaward boundary.  */
double *eb, *eb0;
double *bou, *ebnew;
double pmin, pmax, toll;
double eps, P0;
/*************************************************************************
  Heart of the one-dimensional ENO algorithm.

  *************************************************************************/
void TimeStep(Tskeuze)
     int Tskeuze;
{
  int nnt, nrk, np, ntrue, i, j, k, m;
  double du, dd, fpmax;
  double x1, x2, xs, P1, P2, qbm, qbh;
  /*************************************************************************
   Start timestepping routine.
  *************************************************************************/
  toll = 1.e-8;
  P0   = rcoef[0]*rcoef[1]*rcoef[2]*rcoef[7];
  eps  = rcoef[4]*sqrt(P0)/rcoef[0];
  pmin = 1.e3/P0;
  rcoef[3]/=rcoef[7];
  /*
    Maximum pressure is given by critical pressure for which
    the speed of sound becomes infinite.
  */
```

```
  du = rcoef[3]/rcoef[0];
  if (rcoef[4] > 0.e0) dd = 2.e0*du/eps;
  else
    {
      fprintf(stderr," Saturation constant s<=0. Abort.\n");
      exit(1);
    }
  dd = 108.e0*dd+12.e0*sqrt(12.e0*du*du*du+81.e0*dd*dd);
  dd = pow(dd,1.e0/3.e0);
  pmax = (dd/6.e0-2.e0*du/dd);
  pmax*=pmax;
  if ( (fp6 = fopen("enoiso1xu", "w")) == NULL)
    {
      fprintf(stderr, "\n Error: can not open testsw. \n");
      fprintf(stderr, "\n Function: enoiso1.c. \n");
      exit(1);
    }
  /* Initialize variables. Initial conditions. */
  Initial_Condition(Tskeuze, &Ni);
  /* Store initial conditions. */
  Measure(0, Ni);
#ifdef INTERFACE
  fprintf(fp6, "%g %g\n", xb[0], ub[0]);
#endif
  fpmax = 0.e0;
  fprintf(stderr, " Measure ic's done.\n");
  /*
    Set coefficients of polynomial used in ENO approximation.
  */
  Setcoeff();
  /*********************************************************************
    Start timestepping routine.
    Nt  = # of timeloops.
    Nrk = # of Runge-Kutta cycles.
  */
  for (nnt = 1; nnt <= Nt; nnt++)
    {
       for (nrk = 0; nrk <= (Nrk-1); nrk++)
{
  /*
    Boundary conditions.
  */
  BounCond(nrk);
  /*
    Flux in x direction.
  */
  Fluxiso(nrk, &fpmax);
  /*
    Forcing
  Forcing(nrk, Niz);*/
  /*
    Third-order Runge-Kutta.
  */
  Runge_Kutta(nrk);
}
       for (i=0; i<=Ni-1; i++)
 {
   if (q[i][nqi(0,2)] < WET)
     {
       for (np=1; np<=Nsy; np++) q[i][nqi(0,np)] = 0.e0;
     }
 }
```

```
      if ((nnt % Nmeas) == 0)
 {
   Measure(nnt, Ni);
 }
    }
  fprintf(stderr, " Time and rk loop completed. \n");
  /* Free matrix and array allocation. */
  freedom();
  fprintf(stderr, " Allocated space free. \n");
}
/* End timestepping routine. */

/*********************************************************
  Definition arrays, initial_Conditions and
  initialization cell boundaries.
*********************************************************/
void Initial_Condition(Tk, Nii)
  int Tk, *Nii;
{
  int i, j, m, n, np;
  double cc, h1, h2, p1, p2, u1, u2, x0, x1, x2, ls;
  double fpmax;
  /*
    Print garb.
  */
#ifdef EASTEXTRAP
  fprintf(stderr, " EASTEXTRAP ");
#endif
#ifdef WESTSOLID
  fprintf(stderr, " WESTSOLID ");
#endif
#ifdef EXTRAP
  fprintf(stderr, " EXTRAP ");
#endif
#ifdef ZEROH1
  fprintf(stderr, " ZEROH1 ");
#endif
#ifdef RADGEN
  fprintf(stderr, " RADGEN ");
#endif
#ifdef WESTCHAR
  fprintf(stderr, " WESTCHAR ");
#endif
#ifdef PERIODIC
  fprintf(stderr, " PERIODIC on");
#endif
  fprintf(stderr, "\n");
  /* Array definitions. */
  kmin  = vector(0, Nor+1);
  x     = Dvector(-(Nor+1), Npxts+1);
  qh    = Dvector(1, Nsy);
  xb    = Dvector(0, Nrk-1);
  ub    = Dvector(0, Nrk-1);
  Sea0  = Dvector(0, 3);
  bou   = Dvector(0, 3);
  eb    = Dvector(0, 3);
  eb0   = Dvector(0, 3);
  ebnew = Dvector(0, 3);
  q     = Matrix(-(Nor+1), Npxts+Nor+1, 1, Nrk*Nsy);
  qt    = Matrix(0, Npxts, 1, Nsy);
  fhat  = Matrix(-1, Npxts, 1, Nsy);
  ftat  = Matrix(-1, Npxts, 1, Nsy);
  coeff = Matrix(0, Nor+1, 0, Nor);
  roe   = Matrix(-1, Npxts, 1, Nsy);
  alpha = Matrix(-1, Npxts, 1, Nsy);
```

```
  eval  = Matrix(-1, Npxts, 1, Nsy);
  qdiff = Matrix(-(Nor+2), Npxts+Nor, 1, (Nor+1)*Nsy);
  fdiff = Matrix(-(Nor+2), Npxts+Nor, 1, (Nor+1)*Nsy);
  qtiff = Matrix(-Nor-1, 1, 1, (Nor+1));
  ftiff = Matrix(-Nor-1, 1, 1, (Nor+1));
  R     = Matrix(-1, Npxts, 1, Nsy*Nsy);
  Rm    = Matrix(-1, Npxts, 1, Nsy*Nsy);
  hp    = Matrix(-(Nor+1), 0, 1, (Nor+1)); /* 1, Nor+1, 1, Nsy */
  hm    = Matrix(-(Nor+1), 0, 1, (Nor+1)); /* 1, Nor+1, 1, Nsy */
  for (i=-1; i<=Npxts; i++)
  {
    /* Preset slots in matrices which will not change anymore. */
    for (np=1; np<=Nsy; np++)
{
  for (j=1; j<=Nsy; j++)
    {
      R[i][npif(np,j)]  = 0.e0;
      Rm[i][npif(np,j)] = 0.e0;
    }
}
      R[i][npif(2,1)]  = 1.e0;
      R[i][npif(2,2)]  = 1.e0;
      R[i][npif(3,3)]  = 1.e0;
      Rm[i][npif(3,3)] = 1.e0;
    }
  /* Cell boundaries array. */
  dx = Lx/(double)(Npxts);
  for (i=-2; i<=Npxts+1; i++)
  {
    /*
      The zeroth cell has right boundary at dx
      when starting at x = 0.
    */
    x[i] = (double)(i)*dx+dx;
  }
  /*
    Set time step given cfl.
  */
  dt = dx*cfl;
  /* Steady shock and shock tube test problems. */
  x0 = 0.5e0*Lx;
  x1 = 0.25e0*Lx;
  /* Initial conditions. */
  for (i = 0; i<=(Npxts-1); i++)
    {
      /*
x[i] denotes right cell boundary, hence midpoint
of cell is 0.5*(x[i]-x[i-1]) = 0.5*dx to left,
latter equality is for regular grid.
      */
 switch (icno)
{
case 0:
  {
    /* Isothermal stationary shock.
    h1 = 100.e0/rcoef[7];
    h2 = 600.e0/rcoef[7];
    p1 = zbrent(Dens,h1,pmin,pmax,toll*pmin);
    p2 = zbrent(Dens,h2,pmin,pmax,toll*pmin);
    */
    p2 = 5.e06/(rcoef[0]*rcoef[1]*rcoef[2]*rcoef[7]);
    p1 = 1.e05/(rcoef[0]*rcoef[1]*rcoef[2]*rcoef[7]);
```

```
    h1 = Density(p1);
    h2 = Density(p2);
    u1 = -sqrt(h2*(p2-p1)/(h1*(h2-h1)));
    u2 = u1*h1/h2;
    if (i==1)
      {
fprintf(stderr,"p1=%g p2=%g u1=%g u2=%g \n",p1, p2, u1, u2);
      }
    if ((x[i]-0.5e0*dx) < x0)
      {
q[i][nqi(0,2)] = h2;
q[i][nqi(0,1)] = h2*u2;
q[i][nqi(0,3)] = 0.e0;
      }
    else
      {
if ((x[i]-0.5e0*dx) == x0)
  {
    q[i][nqi(0,2)] = 0.5e0*(h1+h2);
    q[i][nqi(0,1)] = q[i][nqi(0,2)]*0.5e0*(u1+u2);
    q[i][nqi(0,3)] = 0.e0;
  }
else
  {
    q[i][nqi(0,2)] = h1;
    q[i][nqi(0,1)] = h1*u1;
    q[i][nqi(0,3)] = 0.e0;
  }
      }
  } break;
case 1:
  {
    /* Isothermal shock tube.
    h1 = 100e0/rcoef[7];
    h2 = 600.e0/rcoef[7];
    p1 = zbrent(Dens,h1,pmin,pmax,toll*pmin);
    p2 = zbrent(Dens,h2,pmin,pmax,toll*pmin);
    */
    p2 = 5.e06/(rcoef[0]*rcoef[1]*rcoef[2]*rcoef[7]);
    p1 = 1.e05/(rcoef[0]*rcoef[1]*rcoef[2]*rcoef[7]);
    h1 = Density(p1);
    h2 = Density(p2);
    u1 = u2 = 0.e0;
    if ((x[i]-0.5e0*dx) < x0)
      {
q[i][nqi(0,2)] = h2;
q[i][nqi(0,1)] = h2*u2;
q[i][nqi(0,3)] = 0.e0;
      }
    else
      {
if ((x[i]-0.5e0*dx) == x0)
  {
    q[i][nqi(0,2)] = 0.5e0*(h1+h2);
    q[i][nqi(0,1)] = q[i][nqi(0,3)]*0.5e0*(u1+u2);
    q[i][nqi(0,3)] = 0.e0;
  }
else
  {
    q[i][nqi(0,2)] = h1;
    q[i][nqi(0,1)] = h1*u1;
    q[i][nqi(0,3)] = 0.e0;
  }
      }
  } break;
case 2:
```

```
  {
    /* Isothermal moving shock. Reflecting.*/
    h1 = 100.e0/rcoef[7];
    h2 = 600.e0/rcoef[7];
    p1 = zbrent(Dens,h1,pmin,pmax,toll*pmin);
    p2 = zbrent(Dens,h2,pmin,pmax,toll*pmin);
    u2 = (1.e0-h1/h2)*sqrt(h2*(p2-p1)/(h1*(h2-h1)));
    u1 = 0.e0;
    if (i==1)
      {
fprintf(stderr,"p1=%g p2=%g u1=%g u2=%g \n",p1, p2, u1, u2);
      }
    if ((x[i]-0.5e0*dx) < x1)
      {
q[i][nqi(0,2)] = h2;
q[i][nqi(0,1)] = h2*u2;
q[i][nqi(0,3)] = 0.e0;
      }
    else
      {
if ((x[i]-0.5e0*dx) == x1)
  {
    q[i][nqi(0,2)] = 0.5e0*(h1+h2);
    q[i][nqi(0,1)] = q[i][nqi(0,2)]*0.5e0*(u1+u2);
    q[i][nqi(0,3)] = 0.e0;
  }
else
  {
    q[i][nqi(0,2)] = h1;
    q[i][nqi(0,1)] = h1*u1;
    q[i][nqi(0,3)] = 0.e0;
  }
      }
  } break;
case 3:
  {
    /* Isothermal moving and reflecting shock. */
    p2 = 2.e06/(rcoef[0]*rcoef[1]*rcoef[2]*rcoef[7]);
    p1 = 1.e06/(rcoef[0]*rcoef[1]*rcoef[2]*rcoef[7]);
    h1 = Density(p1);
    h2 = Density(p2);
    u2 = (1.e0-h1/h2)*sqrt(h2*(p2-p1)/(h1*(h2-h1)));
    u1 = 0.e0;
    if (i == 1)
      {
fprintf(stderr,"%g %g %g %g \n", p1, p2, h1, h2);
      }
    if ((x[i]-0.5e0*dx) < x0)
      {
q[i][nqi(0,2)] = h2;
q[i][nqi(0,1)] = h2*u2;
q[i][nqi(0,3)] = 0.e0;
      }
    else
      {
if ((x[i]-0.5e0*dx) == x0)
  {
    q[i][nqi(0,2)] = 0.5e0*(h1+h2);
    q[i][nqi(0,1)] = q[i][nqi(0,2)]*0.5e0*(u1+u2);
    q[i][nqi(0,3)] = 0.e0;
  }
else
  {
    q[i][nqi(0,2)] = h1;
    q[i][nqi(0,1)] = h1*u1;
    q[i][nqi(0,3)] = 0.e0;
```

```
  }
      }
  } break;
default:
  {
    fprintf(stderr," Try again, invalid choice of initial conditions.");
    exit (1);
  } break;
}
    }
 *Nii = Npxts;
 return;
}

void freedom()
{
  fclose(fp6);
  free_vector(kmin, 0, Nor+1);
  free_Dvector(x, -(Nor+1), Npxts+1);
  free_Dvector(qh, 1, Nsy);
  free_Dvector(xb, 0, Nrk-1);
  free_Dvector(ub, 0, Nrk-1);
  free_Dvector(Sea0, 0, 3);
  free_Dvector(bou, 0, 3);
  free_Dvector(ebnew, 0, 3);
  free_Dvector(eb, 0, 3);
  free_Dvector(eb0, 0, 3);
  free_Matrix(q, -(Nor+1), Npxts+Nor+1, 1, Nsy*Nrk);
  free_Matrix(qt, 0, Npxts, 1, Nsy);
  free_Matrix(fhat, -1, Npxts, 1, Nsy);
  free_Matrix(ftat, -1, Npxts, 1, Nsy);
  free_Matrix(coeff, 0, Nor+1, 0, Nor);
  free_Matrix(roe, -1, Npxts, 1, Nsy);
  free_Matrix(alpha, -1, Npxts, 1, Nsy);
  free_Matrix(eval, -1, Npxts, 1, Nsy);
  free_Matrix(qdiff, -(Nor+2), Npxts+Nor, 1, Nsy*(Nor+1));
  free_Matrix(fdiff, -(Nor+2), Npxts+Nor, 1, Nsy*(Nor+1));
  free_Matrix(qtiff, -Nor-1, 1, 1, Nor+1);
  free_Matrix(ftiff, -Nor-1, 1, 1, Nor+1);
  free_Matrix(R, -1, Npxts, 1, Nsy*Nsy);
  free_Matrix(Rm, -1, Npxts, 1, Nsy*Nsy);
  free_Matrix(hp, -(Nor+1), 0, 1, (Nor+1));
  free_Matrix(hm, -(Nor+1), 0, 1, (Nor+1));
  return;
}

void Wavemaker(h, u, tijd)
     double *h, *u, tijd;
{
  double Am;
  Am = 0.5e0;
  *h = 1.e0+Am*sin(12.e0*PI*tijd);
  *u = Am*sin(12.e0*PI*tijd);
  return;
}
void Fluxiso(nrk,fpmax)
     int nrk;
     double *fpmax;
{
  int i, j, k, m, n2, ntrue, imin, np;
  double c1, cc, cd, sqhl, sqhr, sqh, du, dd;
  double x1, x2, xs, P1, P2, qbm, qbh;
  /*
```

```
    Compute (divided) difference tables for q_i and fhat_i.
    WARNING: DIVISIONS by 0 IF rho is ZERO.
    Find pressure by using zbrent(Dens,h1,pmin,pmax,toll*pmin);
  */
  for (i=-(Nor+1); i<=(Npxts+Nor); i++)
    {
      qdiff[i-1][npif(1,1)] = q[i][nqi(nrk,1)];
      qdiff[i-1][npif(1,2)] = q[i][nqi(nrk,2)];
      qdiff[i-1][npif(1,3)] = q[i][nqi(nrk,3)];
      /* */
      if (q[i][nqi(nrk,2)] > WET)
{
  fdiff[i-1][npif(1,1)] = q[i][nqi(nrk,1)]*q[i][nqi(nrk,1)]/q[i][nqi(nrk,2)];
  fdiff[i-1][npif(1,1)] += zbrent(Dens,q[i][nqi(nrk,2)],pmin,pmax,toll*pmin);
  fdiff[i-1][npif(1,2)] = q[i][nqi(nrk,1)];
  fdiff[i-1][npif(1,3)] = 0.e0;
}
      else
{
  fdiff[i-1][npif(1,1)] = 0.e0;
  fdiff[i-1][npif(1,2)] = 0.e0;
  fdiff[i-1][npif(1,3)] = 0.e0;
}
    }
  for (k = 1; k <= Nor; k++)
    {
      for (i=-(Nor+1); i<=(Npxts+Nor-k); i++)
{
  for (np=1; np<=Nsy; np++)
    {
      qdiff[i-1][npif(k+1,np)]=qdiff[i][npif(k,np)]-qdiff[i-1][npif(k,np)];
      fdiff[i-1][npif(k+1,np)]=fdiff[i][npif(k,np)]-fdiff[i-1][npif(k,np)];
    }
}
    }

  /*
    Compute Roe speeds for isothermal gasdynamics,
    left and right eigenvectors, eigenvalues, and alpha's.
    WARNING: DIVISIONS by 0 IF q[i][nqi(nrk,2)] is ZERO, i.e. zero density.
  */
  imin = 0;
  Ni   = Npxts;
  Niz = Ni-1;
  /*
    Eigenvalues U-a(p), u+a(p), u.
    Definition alpha's used in LLF.
    Find p from rho(p).
    Calculate cc=a(p)=(drho/dp)^{-1/2}.
  */
  for (i=-1; i<=Niz+1; i++) /* Change from Npxts-1 to Npxts.*/
    {
      if (q[i][nqi(nrk,2)] < WET) c1 = cc = 0.e0;
      else
{
  c1 = q[i][nqi(nrk,1)]/q[i][nqi(nrk,2)]; /* u_i */
  cc = q[i][nqi(nrk,2)];
  cd = zbrent(Dens,cc,pmin,pmax,toll*pmin);
  cc =(cc*cc/(cd*cd))*(1.e0-0.5e0*eps*sqrt(cd)-0.5e0*rcoef[0]*eps*cd*sqrt(cd)/rcoef[3]);
  cc = sqrt(1.e0/cc);
}
      /*
Definition eval=f'(u) at cell centers used to choose ENO Roe or LLF.
In eigenvector space eigenvalues are f'(u).
      */
```

```
      eval[i][1] = c1-cc;
      eval[i][2] = c1+cc;
      eval[i][3] = c1;
      du = dmax(fabs(eval[i][1]),fabs(c1));
      *fpmax = dmax( dmax( du,fabs(eval[i][3]) ), *fpmax);
    }
  /*
    Roe averages u, v and h at i+1/2.
  */
  for (i=-1; i<=Niz; i++)
    {
      /*
alpha's used in Local Lax Friedrich (LLF).
      */
      for (np=1; np<=Nsy; np++)
{
  alpha[i][np]=dmax(fabs(eval[i][np]),fabs(eval[i+1][np]));
}
      if (q[i+1][nqi(nrk,2)] < WET)
{
  if (q[i][nqi(nrk,2)] < WET) qh[1] = qh[2] = qh[3] = 0.e0;
  else
    {
      qh[1]  = q[i][nqi(nrk,1)]/q[i][nqi(nrk,2)];
      qh[2]  = 0.5e0*q[i][nqi(nrk,2)];
      qh[3]  = q[i][nqi(nrk,2)]/q[i][nqi(nrk,2)];
    }
}
      else
{
  if (q[i][nqi(nrk,2)] < WET)
    {
      qh[1]  = q[i+1][nqi(nrk,1)]/q[i+1][nqi(nrk,2)];
      qh[2]  = 0.5e0*q[i+1][nqi(nrk,2)];
      qh[3]  = q[i+1][nqi(nrk,3)]/q[i+1][nqi(nrk,2)];
    }
  else
    {
      /* Roe averages at cell boundary. */
      sqhl = sqrt(q[i][nqi(nrk,2)]);
      sqhr = sqrt(q[i+1][nqi(nrk,2)]);
      sqh  = sqhl + sqhr;
      qh[1]  = (q[i][nqi(nrk,1)]/sqhl+q[i+1][nqi(nrk,1)]/sqhr)/sqh;
      qh[3]  = (q[i][nqi(nrk,3)]/sqhl+q[i+1][nqi(nrk,3)]/sqhr)/sqh;
      qh[2]  = 0.5e0*(q[i][nqi(nrk,2)]+q[i+1][nqi(nrk,2)]);
    }
}
      /*
Roe average of density usual form: sqrt(qh[2]), then
``simply'' use this to define speed of sound.
      */
      cc = qh[2];
      cd = zbrent(Dens,cc,pmin,pmax,toll*pmin);
      cc =(cc*cc/(cd*cd))*(1.e0-0.5e0*eps*sqrt(cd)-0.5e0*rcoef[0]*eps*cd*sqrt(cd)/rcoef[3]);
      cc = sqrt(1.e0/cc);
      /* Definition eigenvalues. */
      roe[i][1] = (qh[1]-cc);
      roe[i][2] = (qh[1]+cc);
      roe[i][3] =  qh[1];
      /* Definition right eigenvector matrix. */
      R[i][npif(1,1)] = roe[i][1];
      R[i][npif(1,2)] = roe[i][2];
      R[i][npif(1,3)] = 0.e0;
      R[i][npif(2,1)] = 1.e0;
```

```
      R[i][npif(2,2)] = 1.e0;
      R[i][npif(2,3)] = 0.e0;
      R[i][npif(3,1)] = qh[3];
      R[i][npif(3,2)] = qh[3];
      R[i][npif(3,3)] = 1.e0;
      /* Definition left eigenvector matrix. */
      if (qh[2] > WET)
{
  cc = 0.5e0/cc;
  Rm[i][npif(3,1)] = 0.e0;
  Rm[i][npif(3,2)] = -qh[3];
  Rm[i][npif(3,3)] = 1.e0;
  Rm[i][npif(1,1)] = -cc;
  Rm[i][npif(1,2)] = cc*roe[i][2];
  Rm[i][npif(1,3)] = 0.e0;
  Rm[i][npif(2,1)] = cc;
  Rm[i][npif(2,2)] = -cc*roe[i][1];
  Rm[i][npif(2,3)] = 0.e0;
}
      else
{
  Rm[i][npif(1,3)] = 0.e0;
  Rm[i][npif(2,3)] = 0.e0;
  Rm[i][npif(3,3)] = 0.e0;
  Rm[i][npif(1,1)] = 0.e0;
  Rm[i][npif(3,1)] = 0.e0;
      }
    }

  /*
    Combination of EnoROE and EnoLLF.
  */
  for (i=-1; i<=Niz; i++)
    {
      ntrue = 0;
      for (np=1; np<=Nsy; np++)
{
  /*
    Entropy fix? Yes: EnoLLF. No: EnoROE.
    if (ntrue == 0) EnoLLF(i,np,Npxts);
    if ((eval[i][np] < 0.e0) && (0.e0 <
    eval[i+1][np])) EnoLLF(i,np,Npxts);
  */
  if ((eval[i][np] < 0.e0) && (0.e0 < eval[i+1][np]))
    {
      EnoLLF(i,np,Npxts);
    }
  else EnoRoe(i,np,Npxts);
}
      /*
Project ftat back from eigenspace to fhat in normal space.
      */
      for (np=1; np<=Nsy; np++)
{
  fhat[i][np] =R[i][npif(np,1)]*ftat[i][1];
  for (n2=2; n2<=Nsy; n2++) fhat[i][np]+=R[i][npif(np,n2)]*ftat[i][n2];
}
    }

  /*
    f[i]   -> f_(i+1/2)
    f[i-1] -> f_(i-1/2)
    For i = 0 one finds f[-1] -> f_(-1/2), i.e. at the left boundary edge.
  */
  for (i=imin; i<=Niz; i++)
```

```
    {
      for (np = 1; np<=Nsy; np++)
{
  /* Note that the peculiar minus sign comes from the definition
     in Shu and Osher's and Shu's papers. */
  qt[i][np] = -(fhat[i][np]-fhat[i-1][np])/(x[i]-x[i-1]);
}
    }
  return;
}

/*
  Third-order Runge-Kutta.
*/
void Runge_Kutta(nk)
     int nk;
{
  int i, j, np;
  double du, dd;
  if (nk == 0)
    {
      for (i=0; i<=Niz; i++)
{
  for (np=1; np<=Nsy; np++)
    {
      q[i][nqi(1,np)] = q[i][nqi(0,np)]+dt*qt[i][np];
    }
}
    }
  else
    {
      if (nk == 1)
{
  for (i=0; i<=Niz; i++)
    {
      for (np = 1; np<=Nsy; np++)
{
  q[i][nqi(2,np)]  = 0.75e0*q[i][nqi(0,np)];
  q[i][nqi(2,np)] += 0.25e0*(q[i][nqi(1,np)]+dt*qt[i][np]);
}
    }
}
      else
{
  for (i=0; i<=Niz; i++)
    {
      for (np=1; np<=Nsy; np++)
{
  q[i][nqi(0,np)]=(q[i][nqi(0,np)]+2.e0*(q[i][nqi(2,np)]+dt*qt[i][np]));
  q[i][nqi(0,np)] *=ONET;
}
    }
}
    }
  return;
}

void Forcing(nk, Nz)
     int nk, Nz;
{
  int i;
#ifdef FORCEDAMPn1
  for (i=0; i<=(Nz-1); i++)
    {
      qt[i][1] -= rcoef[6]*q[i][nqi(nk,1)];
    }
```

```
#endif
  return;
}

void BounCond(nrk)
  int nrk;
{
  int i, j, k, np;
  double epsilon, h1, h2, u2, cc, vvel, uvel;
  epsilon = 1.0e-10;
#ifdef WESTCHAR
  /*
    Characteristic treatment of imposed flow on western (seaward) boundary.
    Values at ghost points are all based on values at the first interior
    ''0'' point. Values fo each variable are equal at different
    ghost points. This guarantees a constant incoming or outgoing flux.
    Get new boundary data (at current partial time)
    and characteristic quantities at the current full time.
   */
  bou[1] = q[0][nqi(nrk,1)];
  bou[2] = q[0][nqi(nrk,2)];
  bou[3] = q[0][nqi(nrk,3)];
  cc   = q[0][nqi(0,2)];
  cd =zbrent(Dens,cc,pmin,pmax,toll*pmin);
  cc =(cc*cc/(cd*cd))*(1.e0-0.5e0*eps*sqrt(cd)-0.5e0*rcoef[0]*eps*cd*sqrt(cd)/rcoef[3]);
  cc = sqrt(1.e0/cc);
  uvel = q[0][nqi(0,1)]/q[0][nqi(0,2)];
  vvel = q[0][nqi(0,3)]/q[0][nqi(0,2)];
  /*
     Set the characteristic quantities based on previous full time.
  */
  roe[0][1] = uvel-cc;
  roe[0][2] = uvel+cc;
  roe[0][3] = uvel;
  /*
    Definition right eigenvector matrix.
  */
  R[0][npif(1,1)] = roe[0][1];
  R[0][npif(1,2)] = roe[0][2];
  R[0][npif(1,3)] = 0.e0;
  R[0][npif(2,1)] = 1.e0;
  R[0][npif(2,2)] = 1.e0;
  R[0][npif(2,3)] = 0.e0;
  R[0][npif(3,1)] = vvel;
  R[0][npif(3,2)] = vvel;
  R[0][npif(3,3)] = 1.e0;
  /* Definition left eigenvector matrix. */
  cc = 0.5e0/cc;
  Rm[0][npif(3,1)] = 0.e0;
  Rm[0][npif(3,2)] = -vvel;
  Rm[0][npif(3,3)] = 1.e0;
  Rm[0][npif(1,1)] = -cc;
  Rm[0][npif(1,2)] = cc*roe[0][2];
  Rm[0][npif(1,3)] = 0.e0;
  Rm[0][npif(2,1)] = cc;
  Rm[0][npif(2,2)] = -cc*roe[0][1];
  Rm[0][npif(2,3)] = 0.e0;
  /*
    Project Sea0 -> eb0 and b -> b0
   */
  for (np=1; np<=3; np++)
    {
      eb0[np] = Rm[0][npif(np,1)]*Sea0[1]+Rm[0][npif(np,2)]*Sea0[2];
```

```
      eb0[np]+= Rm[0][npif(np,3)]*Sea0[3];
      eb[np]  = Rm[0][npif(np,1)]*bou[1]+Rm[0][npif(np,2)]*bou[2];
      eb[np] += Rm[0][npif(np,3)]*bou[3];
    }
  /*
    Check direction of waves to determine whether ghostpoints
    obtain prescribed boundary values or interior values.
  */
   for (np=1; np<=3; np++)
    {
      if (roe[0][np] > 0.e0) ebnew[np] = eb0[np];
      else ebnew[np] = eb[np];
    }
   /*
     Project back onto values of q[][] for ghost points.
   */
   for (np=1; np<=3; np++)
     {
       for (i=-(Nor+1); i<=-1; i++)
 {
   q[i][nqi(nrk,np)] = R[0][npif(np,1)]*ebnew[1]+R[0][npif(np,2)]*ebnew[2];
   q[i][nqi(nrk,np)]+= R[0][npif(np,3)]*ebnew[3];
 }
     }
#endif
#ifdef PERIODIC
   /*
     Periodic boundary conditions at left and right boundary.
   */
  for (i=-(Nor+1); i<=-1; i++)
    {
      for (np=1; np<=Nsy; np++) q[Npxts-i-1][nqi(nrk,np)]=q[-i-1][nqi(nrk,np)];
      for (np=1; np<=Nsy; np++) q[i][nqi(nrk,np)]=q[i+Npxts][nqi(nrk,np)];
    }
#endif
#ifdef EXTRAP
  /*
    Extrapolating boundary conditions on left and right boundary.
    Make extrapolating boundary conditions fourth order for uniform grid.
  */
  for (i=Npxts; i<=(Npxts+Nor); i++)
    {
      for (np=1; np<=Nsy; np++) q[i][nqi(nrk,np)]=q[Npxts-1][nqi(nrk,np)];
    }
  for (i=-(Nor+1); i<=-1; i++)
    {
      for (np=1; np<=Nsy; np++) q[i][nqi(nrk,np)]=q[0][nqi(nrk,np)];
    }
#endif
#ifdef WESTEXTRAP
  /*
    Extrapolating boundary conditions on left/west boundary.
    Make extrapolating boundary conditions fourth order for uniform grid.
  */
  for (i=-(Nor+1); i<=-1; i++)
    {
      for (np=1; np<=Nsy; np++) q[i][nqi(nrk,np)]=q[0][nqi(nrk,np)];
    }
#endif
#ifdef EASTEXTRAP
  /*
    Extrapolating boundary conditions on right/east boundary.
    Make extrapolating boundary conditions fourth order for uniform grid.
  */
  for (i=Npxts; i<=(Npxts+Nor); i++)
    {
      for (np=1; np<=Nsy; np++) q[i][nqi(nrk,np)]=q[Npxts-1][nqi(nrk,np)];
```

```
    }
#endif
#ifdef SOLID
  /*
    No normal flow boundary conditions for nonrotating case.
  */
  for (i=Npxts; i<=Npxts+Nor; i++)
    {
      q[i][nqi(nrk,1)]=-q[2*Npxts-i-1][nqi(nrk,1)];
      q[i][nqi(nrk,2)]= q[2*Npxts-i-1][nqi(nrk,2)];
      q[i][nqi(nrk,3)]= q[2*Npxts-i-1][nqi(nrk,3)];
    }
  for (i=-(Nor+1); i<=-1; i++)
    {
      q[i][nqi(nrk,1)]=-q[-i-1][nqi(nrk,1)];
      q[i][nqi(nrk,2)]= q[-i-1][nqi(nrk,2)];
      q[i][nqi(nrk,3)]= q[-i-1][nqi(nrk,3)];
    }
#endif
#ifdef EASTSOLID
  /*
    No normal flow boundary conditions for nonrotating case
    at east/right boundary.
  */
  for (i=1; i<=Nor+1; i++)
    {
      q[Npxts-1+i][nqi(nrk,1)]=-q[Npxts-i-1][nqi(nrk,1)];
      q[Npxts-1+i][nqi(nrk,2)]= q[Npxts-i-1][nqi(nrk,2)];
      q[Npxts-1+i][nqi(nrk,3)]= q[Npxts-i-1][nqi(nrk,3)];
    }
#endif
#ifdef WESTSOLID
  /*
    No normal flow boundary conditions for nonrotating case
    at west/left wall.
    */
  for (i=-(Nor+1); i<=-1; i++)
    {
      q[i][nqi(nrk,1)]=-q[-i-1][nqi(nrk,1)];
      q[i][nqi(nrk,2)]= q[-i-1][nqi(nrk,2)];
      q[i][nqi(nrk,3)]= q[-i-1][nqi(nrk,3)];
    }
#endif
}

double Density(pp)
     double pp;
{
  double du;
  du = 1.e0-eps*sqrt(pp);
  du = du/pp+(1.e0-rcoef[0]*du)/rcoef[3];
  return 1.e0/du;
}
double Dens(pp, d0)
     double pp, d0;
{
  double du;
  du = 1.e0-eps*sqrt(pp);
  du = du/pp+(1.e0-rcoef[0]*du)/rcoef[3];
  return (d0 - 1.e0/du);
}

/*****************************************************************
  Measure(ntime) subroutine for measurements at integer time ntime.
```

```
*************************************************************/
void Measure(ntime, Ni)
  int ntime, Ni;
{
  int i;
  double ke, pe, pp, p0, du;
  fp2 = fopen(Ts1, "a");
  fp3 = fopen(Ts2, "a");
  fprintf(fp2," %16.10lf %d \n", ntime*dt, Npxts);
  ke   = 0.e0;
  pe   = 0.e0;
  p0   = 1.e0; /* Dimensionally P0. */
  if (ntime == 0) fprintf(stderr, " %d Ni = %d %g \n", ntime, Ni,x[Ni]);
  /*
    Simple sum to obtain integral of energy density for inviscid case.
  */
  for (i=0; i<=(Npxts-1);i++)
    {
      pp = zbrent(Dens,q[i][nqi(0,2)],pmin,pmax,toll*pmin);
      fprintf(fp2," %16.10lf %16.10lf ",x[i]-0.5e0*dx,q[i][nqi(0,1)]);
      fprintf(fp2," %16.10lf %16.10lf \n",q[i][nqi(0,2)],pp);
      ke+= 0.5e0*q[i][nqi(0,1)]*q[i][nqi(0,1)]/q[i][nqi(0,2)];
      du = log(pp/p0)+2.e0*eps*(sqrt(p0)-sqrt(pp));
      du+= ((1.e0-rcoef[0])*(pp-p0)+2.e0*ONET*eps*rcoef[0]*
(pow(pp,1.5e0)-pow(p0,1.5e0)))/rcoef[3]-pp/q[i][nqi(0,2)];
      pe+= du*q[i][nqi(0,2)];
    }
#ifdef PERIODIC
  i = 0;
  pp = zbrent(Dens,q[i][nqi(0,2)],pmin,pmax,toll*pmin);
  fprintf(fp2," %16.10lf %16.10lf",x[Npxts]-0.5e0*dx,q[i][nqi(0,1)]);
  fprintf(fp2," %16.10lf %16.10lf\n",q[i][nqi(0,2)],pp);
#endif
#ifdef INTERFACE
  fprintf(fp2," %16.10lf %16.10lf",0.5e0*(x[Jb]+xb[0]),q[Jb][0]);
  fprintf(fp2," %16.10lf %16.10lf\n",xb[0],ub[0]);
#endif
  fprintf(fp3," %16.10lf %16.10lf %16.10lf %16.10lf\n", ntime*dt,(pe+ke)*dx,pe*dx,ke*dx);
  fclose(fp2);
  fclose(fp3);
}

/*********************************************************
Set ENO coefficients. UNIFORM grid.
*********************************************************/
void Setcoeff()
{
  int i, j, k, nal, nbe, nprod;
  for (j=0; j<=(Nor+1); j++)
    {
      for (k=0; k<=Nor; k++)
{
  coeff[j][k] = 0.e0;
  for (nal=0; nal<=k; nal++)
    {
      nprod = 1;
      for (nbe=0; nbe<=k; nbe++)
{
  if (nal != nbe) nprod *= (j-nbe);
}
      coeff[j][k] += (double)(nprod);
    }
  /* Divide by (k+1)! when using UN-divided differences. */
```

```
  for (i=2; i<=k+1; i++) coeff[j][k] /= (double)(i);
}
    }
  return;
}

/*****************************************************************
  subroutine ENORoe
  Use the (hybrid) ENO-Roe algorithm to compute the numerical
  approximation to the flux at x(j+1/2) or y(j+1/2).
  roe(j):     Roe speed: f'(u) at j+1/2
  fhat(j):    numerical flux at j+1/2
  h(j,k):     Difference table for the flux
              H[z(j+1/2),...,z(j+1/2+k)]   ( z= x or y)
              j: spacial range       -(Nor+2) <= j <= JJ+Nor
              k: order of difference       1 <= k <= Nor+1
  kmin(k):    left-most stencil pt for evaluation of flux at kth stage
 */
void EnoRoe(ii, nnp, Nx)
  int ii, nnp, Nx;
{
  int i, j, k, n2;
  double weight, epsilon;
  epsilon = 1.e-10;
  weight = 2.e0;
  /*
    Project utilised portion of divided differences
    onto the eigenspace.
    Utilised portion for given k at i runs from
    H^{k}_{j-1/2-k+1} to H^{k}_{j+1/2}
    */
  for (k=1; k<=(Nor+1); k++)
    {
      for (j=-k; j<=0; j++)
{
  /* Note that for j=-1 we get the expected ordering
     h^p(i-1) = R_i h(i-1).
  */
  qtiff[j][k]  = Rm[ii][npif(nnp,1)]*qdiff[ii+j][npif(k,1)];
  ftiff[j][k]  = Rm[ii][npif(nnp,1)]*fdiff[ii+j][npif(k,1)];
  for (n2=2; n2<=Nsy; n2++)
    {
      qtiff[j][k] += Rm[ii][npif(nnp,n2)]*qdiff[ii+j][npif(k,n2)];
      ftiff[j][k] += Rm[ii][npif(nnp,n2)]*fdiff[ii+j][npif(k,n2)];
    }
}
    }

  /* Set initial point via upwinding based on Roe speed. */
  if(roe[ii][nnp] >= 0.e0) kmin[1] = 0; /* Note that in ftiff ii -> 0. */
  else kmin[1] = 1;
  /* Set first term in polynomial. */
  ftat[ii][nnp] = ftiff[kmin[1]-1][1];
  for (k=2; k<=Nor+1; k++)
    {
      /* Select the next stencil point. */
      if (((k%2 == 0) && (roe[ii][nnp]<0.e0)) || ((k%2 == 1) && (roe[ii][nnp]>=0.e0)))
{
  /* Favour backing up. */
  if ((weight*fabs(ftiff[kmin[k-1]-1][k]))>fabs(ftiff[kmin[k-1]-2][k]))
    {
```

```
      kmin[k] = kmin[k-1]-1;
    }
  else kmin[k] = kmin[k-1];
}
      else
{
  /* Favour staying put. */
  if (fabs(ftiff[kmin[k-1]-1][k])>(weight*fabs(ftiff[kmin[k-1]-2][k])))
    {
      kmin[k] = kmin[k-1]-1;
    }
  else kmin[k] = kmin[k-1];
}
      ftat[ii][nnp] += ftiff[kmin[k]-1][k]*coeff[1-kmin[k-1]][k-1]; /* Adjust coeff as well. */
    }
  return;
}
/*****************************************************************
  subroutine ENOLLF

  Use the (hybrid) ENO-LLF algorithm to compute the numerical
  approximation to the flux at x(j+1/2).

  qdiff(j,k): difference table for the state variable u
  fdiff(j,k): difference table for the flux, f(u)
  fdiff[x(j+1/2),...,x(j+1/2+k)]
                 j: spacial range       -(Nor+2) <= j <= JJ+Nor
                 k: order of difference       1 <= k <= Nor+1
  alpha(j):   alpha(j+1/2) = max|f'(u)| over j and j+1
  h(j,k,npm): difference table for H+ (npm=1) and H- (npm=2)
                 H+/- = 1/2*(fdiff +/- alpha*udiff)
  dq(j,npm):  d(Q+)/dx and d(Q-)/dx   at j+1/2
  kmin(k,npm):  left-most stencil pt minus j for evaluation of
                 fhat(j+1/2) at the kth stage

  ftat(j):    numerical flux at j+1/2  =  d(Q+)/dx + d(Q-)/dx
  efix:       efix=0 => this is the ENOLLF scheme
              efix=1 => this is the entropy fix for the ENORoe scheme.

*/
void EnoLLF(ii, nnp, Nx)
  int ii, nnp, Nx;
{
  int i, j, jcentered, k, n2;
  double weight, epsilon;

  epsilon = 1.e-10;
  weight = 2.e0;

  /*
    Project to eigenvalue space.
    */
  for (k=1; k<=(Nor+1); k++)
    {
      for (j=-k; j<=0; j++)
{
  /* Note that in hp, hm ii -> 0. */
  hp[j][k]  = Rm[ii][npif(nnp,1)]*(fdiff[ii+j][npif(k,1)]+
alpha[ii][1]*qdiff[ii+j][npif(k,1)]);
  hm[j][k]  = Rm[ii][npif(nnp,1)]*(fdiff[ii+j][npif(k,1)]-
alpha[ii][1]*qdiff[ii+j][npif(k,1)]);
  for (n2=2; n2<=Nsy; n2++)
    {
      hp[j][k] += Rm[ii][npif(nnp,n2)]*(fdiff[ii+j][npif(k,n2)]+
alpha[ii][n2]*qdiff[ii+j][npif(k,n2)]);
      hm[j][k] += Rm[ii][npif(nnp,n2)]*(fdiff[ii+j][npif(k,n2)]-
alpha[ii][n2]*qdiff[ii+j][npif(k,n2)]);
```

```
    }
  hp[j][k] *= 0.5e0;
  hm[j][k] *= 0.5e0;
}
    }
  /*
    H+ part.
  */
  kmin[1] = 0;
  ftat[ii][nnp] = hp[kmin[1]-1][1];
  for (k=2; k<=(Nor+1); k++)
    {
      jcentered = -(k-1)/2;
      if (kmin[k-1] > jcentered)
{
  if ((weight*fabs(hp[kmin[k-1]-1][k])) > fabs(hp[kmin[k-1]-2][k]))
    {
      kmin[k] = kmin[k-1]-1;
    }
  else kmin[k] = kmin[k-1];
}
      else
{
  if (fabs(hp[kmin[k-1]-1][k]) > (weight*fabs(hp[kmin[k-1]-2][k])))
    {
      kmin[k] = kmin[k-1]-1;
    }
  else kmin[k] = kmin[k-1];
}
      ftat[ii][nnp] += hp[kmin[k]-1][k]*coeff[1-kmin[k-1]][k-1]; /* Adjust coeff as well. */
    }
  /*
    H- part.
    */
  kmin[1] = 1;
  ftat[ii][nnp] += hm[kmin[1]-1][1];
  for (k=2; k<=Nor+1; k++)
    {
      jcentered = 1-(k/2);
      if (kmin[k-1] > jcentered)
{
  if (weight*fabs(hm[kmin[k-1]-1][k]) > fabs(hm[kmin[k-1]-2][k]))
    {
      kmin[k] = kmin[k-1]-1;
    }
  else kmin[k] = kmin[k-1];
}
      else
{
  if (fabs(hm[kmin[k-1]-1][k]) > weight*fabs(hm[kmin[k-1]-2][k]))
    {
      kmin[k] = kmin[k-1]-1;
    }
  else kmin[k] = kmin[k-1];
}
      ftat[ii][nnp] += hm[kmin[k]-1][k]*coeff[1-kmin[k-1]][k-1]; /* Adjust coeff as well. */
    }
  return;
}

/*******************************************************
Function Max(,) determines the maximum of
two integer values and returns the biggest.
*/
int Max(k, l) int k, l;
```

```
{
return (k > l ? k:l);
}

/*********************************************************
Function Min(,) determines the minimum of
two integer values and returns the smallest.
*/
int Min(k, l) int k, l;
{
return (k < l ? k:l);
}
/*********************************************************
Function dmax(,) determines the maximum of
two double values and returns the biggest.
*/
double dmax(dk, dl) double dk, dl;
{
return (dk > dl ? dk:dl);
}

/*********************************************************
    Function dmin(,) determines the minimum of
    two integer values and returns the smallest.
*/
  double dmin(dk, dl) double dk, dl;
{
  return (dk < dl ? dk:dl);
}

/*********************************************************
  kk = 1, ..., Nor+1
  pp = 1, ..., Nsy
*/
int npif(nkk, pp)
  int nkk, pp;
{
  return ((nkk-1)*Nsy + pp);
}

/*********************************************************
  nkk= 0,1, .., Nrk-1
  pp = 1,2,..., Nsy
*/
int nqi(nkk, pp)
  int nkk, pp;
{
  return (nkk*Nsy + pp);
}

/*********************************************************
*/
int nri(nkk, pp)
  int nkk, pp;
{
  return ((nkk-1)*Nsy + pp);
}
```

Scientific Notebook
20-1402-461
Inititals: CC

Chuck Connor
July 4,1999

The following pages contain Cerro Negro magnetic data collected in August, 1999 at Cerro Negro volcano by Chuck Connor and others (P. La Femina, Nole Rogers, Stewart Sandberg)

A >2 km-long fracture set developed south of Cerro Negro volcano during and following seismic activity on August 4, 1999. The largest earthquakes, magnitude 4.9 and 4.6, preceded by ≈ 12 and 4 hrs, respectively, the onset of the August 5 - 7 small-volume basaltic eruption from new vents on the south flank of Cerro Negro. Our integrated geophysical surveys concentrated on an area ≈1 km south of Cerro Negro, where total dilation across the fracture set was up to 1 m over a distance of < 20 m, with little evidence of vertical or strike-slip displacement. Steam flow was visible from this fracture set in the days following the eruption.

Surveys included ground magnetic, SP, radon, and temperature monitoring on profiles perpendicular to the fracture set. A linear, normally-magnetized 1200 nT anomaly was identified within the thermal area, which we have modeled as a 1 m wide, steeply dipping tabular body of magnetized rock (1-5 A/m), at a depth of 25-35 m beneath the surface. SP (160 mV), radon (> 60 pCi/l), and temperature (90 °C) anomalies across the fracture set all indicate forced convective flow of water vapor and gases through the fractures.

One explanation for our observations is that lateral dike injection occurred from beneath Cerro Negro toward the south, following seismic activity along a preexisting fault. This interpretation is also consistent with the geology of Cerro Negro, where N-S to NW-SE trending structures are common, including the N-S trending Cerro La Mula cinder cones and phreatic pit craters north of Cerro Negro, the 1968 Cristo Rey vent, and fracture sets formed in Cerro Las Pilas volcano, south of Cerro Negro, in 1955. In 1994-1995, we identified soil radon anomalies on the south side of Cerro Negro, in the location of the new vents and the new fracture set, suggesting a permeable structure (fault / fracture) was already present at that time. Based on this model, the 1999 eruption and the pre-eruption seismicity were a response to crustal extension across the volcano alignment. In this case, the resulting strain was partitioned between seismic deformation and magma intrusion.

```
#! /bin/bash
# set the frame width to 1.0 point - not really needed
gmtset FRAME_PEN 1.0p

# surface does the interpolation with a minimum curvature algorithm
# -I2  means the grid spacing is 2  units in each direction
# -Gfilename indicates the output file
# -R specifies the west,east,south, and north bounds of the map
surface -I2 cnswap.dat -Gcnswap.grd -R532260/532380/1381000/1381100


#grdcontour draws the contours from the grid
# -JX6.0i means the map will be 6 inches wide
# -C250 means there is a 250 nT contour interval
# -A500 means the contours are annotated every 500 nT
# -B25a50f5/WSne draw the frame, 25 m tick with 50 m label, add 5 m ticks, label on
#                    west and south side only
# -W0.25p set line width
# -P draw in portrait mode
grdcontour cnswap.grd -JX6.0i -C100 -A500 -B25a50f5/WSne -W0.25p  -P   > cn.ps

#draw the image
ghostview cn.ps
```

```
#! /bin/bash
# set the frame width to 1.0 point - not really needed
gmtset FRAME_PEN 2.0p

# surface does the interpolation with a minimum curvature algorithm
# -I2  means the grid spacing is 2  units in each direction
# -Gfilename indicates the output file
# -R specifies the west,east,south, and north bounds of the map
surface -I2 cnswap.dat -Gcnswap.grd -R532260/532380/1381000/1381100

# makecpt creates a color table
# -Cno_green specifies the basic color table
# -T specifies the range and interval
 makecpt -Cno_green -T37600/40800/200 > cn.cpt

# grdimage plots the image (color map)
# -JX6.0i is the scale (must match the following)
# -Ccn.cpt is the color scale created with makecpt
# -P portait mode (must match the following)
# -E is the dpi of the color shading
# -K more postscript to be appended to cn.ps in the following
grdimage cnswap.grd -JX6.0i -Ccn.cpt -P -E20 -K > cn.ps

#grdcontour draws the contours from the grid
# -JX6.0i means the map will be 6 inches wide
# -C250 means there is a 250 nT contour interval
# -A500 means the contours are annotated every 500 nT
# -B25a50f5/WSne draw the frame, 25 m tick with 50 m label, add 5 m ticks, label on
#                  west and south side only
# -W0.25p set line width
# -P draw in portrait mode
# -O overlay contours on the image
grdcontour cnswap.grd -JX6.0i -C100 -A500 -B25a50f5/WSne -W0.25p  -P  -O  >> cn.ps

#draw the image
ghostview cn.ps
```

```
#! /bin/bash
# set the frame width to 1.0 point - not really needed
gmtset FRAME_PEN 2.0p

# surface does the interpolation with a minimum curvature algorithm
# -I2  means the grid spacing is 2  units in each direction
# -Gfilename indicates the output file
# -R specifies the west,east,south, and north bounds of the map
surface -I2 cnswap.dat -Gcnswap.grd -R532260/532380/1381000/1381100

# makecpt creates a color table
# -Cno_green specifies the basic color table
# -T specifies the range and interval
 makecpt -Cno_green -T37600/40800/200 > cn.cpt

psmask -I2 -R532260/532380/1381000/1381100 -JX6.0i cnswap.dat -S10  -B25a50f5/WSne  -P -K > c
+ n.ps

# grdimage plots the image (color map)
# -JX6.0i is the scale (must match the following)
# -Ccn.cpt is the color scale created with makecpt
# -P portait mode (must match the following)
# -E is the dpi of the color shading
# -K more postscript to be appended to cn.ps in the following
grdimage cnswap.grd -JX6.0i -Ccn.cpt -P -E20 -O -K >> cn.ps

#grdcontour draws the contours from the grid
# -JX6.0i means the map will be 6 inches wide
# -C250 means there is a 250 nT contour interval
# -A500 means the contours are annotated every 500 nT
# -B25a50f5/WSne draw the frame, 25 m tick with 50 m label, add 5 m ticks, label on
#                   west and south side only
# -W0.25p set line width
# -P draw in portrait mode
# -O overlay contours on the image
grdcontour cnswap.grd -JX6.0i -C100 -A500 -P  -O -K  >> cn.ps

# psxy plots the data points as solid squares
# -S specifies the size and shape
# - G0 makes the squarews solid
psxy cnswap.dat -R532260/532380/1381000/1381100 -JX6.0i -Ss0.05i -G0 -O -P >> cn.ps

#draw the image
ghostview cn.ps
```

```
#! /bin/bash
# set the frame width to 1.0 point - not really needed
gmtset FRAME_PEN 2.0p

# surface does the interpolation with a minimum curvature algorithm
# -I2  means the grid spacing is 2  units in each direction
# -Gfilename indicates the output file
# -R specifies the west,east,south, and north bounds of the map
surface -I2 cnswap.dat -Gcnswap.grd -R532260/532380/1381000/1381100

# makecpt creates a color table
# -Cno_green specifies the basic color table
# -T specifies the range and interval
 makecpt -Cno_green -T37600/40800/200 > cn.cpt

# grdimage plots the image (color map)
# -JX6.0i is the scale (must match the following)
# -Ccn.cpt is the color scale created with makecpt
# -P portait mode (must match the following)
# -E is the dpi of the color shading
# -K more postscript to be appended to cn.ps in the following
grdimage cnswap.grd -JX6.0i -Ccn.cpt -P -E20 -K > cn.ps

#grdcontour draws the contours from the grid
# -JX6.0i means the map will be 6 inches wide
# -C250 means there is a 250 nT contour interval
# -A500 means the contours are annotated every 500 nT
# -B25a50f5/WSne draw the frame, 25 m tick with 50 m label, add 5 m ticks, label on
#                  west and south side only
# -W0.25p set line width
# -P draw in portrait mode
# -O overlay contours on the image
grdcontour cnswap.grd -JX6.0i -C100 -A500 -B25a50f5/WSne -W0.25p  -P  -O -K  >> cn.ps

# psxy plots the data points as solid squares
# -S specifies the size and shape
# - G0 makes the squarews solid
psxy cnswap.dat -R532260/532380/1381000/1381100 -JX6.0i -Ss0.05i -G0 -O -P >> cn.ps

#draw the image
ghostview cn.ps
```

39039.05	1381034.17	532371.72
39034.79	1381033.55	532370.93
39036.00	1381032.93	532370.15
39028.16	1381032.31	532369.36
39034.43	1381031.69	532368.57
39031.28	1381031.07	532367.79
39023.83	1381030.45	532367.00
39011.08	1381029.83	532366.22
38997.12	1381029.21	532365.43
38990.5.0	1381028.60	532364.65
38976.53	1381027.98	532363.86
38963.20	1381027.36	532363.08
38957.54	1381026.74	532362.29
38948.38	1381026.12	532361.51
38942.14	1381025.50	532360.72
38928.41	1381024.88	532359.93
38924.51	1381024.26	532359.15
38919.78	1381023.64	532358.36
38910.88	1381023.02	532357.58
38900.43	1381022.41	532356.79
38883.35	1381021.79	532356.01
38861.79	1381021.17	532355.22
38870.41	1381020.55	532354.44
38862.27	1381019.93	532353.65
38858.04	1381019.31	532352.87
38852.86	1381018.69	532352.08
38844.02	1381018.07	532351.30
38839.45	1381017.45	532350.51
38829.58	1381016.84	532349.72
38807.93	1381016.22	532348.94
38792.00	1381015.60	532348.15
38767.49	1381014.98	532347.37
38766.64	1381014.36	532346.58
38893.56	1381013.74	532345.80
38892.24	1381013.12	532345.01
38750.42	1381012.50	532344.23
38729.14	1381011.88	532343.44
38718.23	1381011.26	532342.66
38832.28	1381010.65	532341.87
38679.90	1381010.03	532341.08
38666.07	1381009.41	532340.30
38659.39	1381008.79	532339.51
38653.92	1381008.17	532338.73
38773.22	1381007.55	532337.94
38625.85	1381006.93	532337.16
38611.46	1381006.31	532336.37
38605.77	1381005.69	532335.59
38595.72	1381005.07	532334.80
38590.91	1381004.46	532334.02
38569.34	1381003.84	532333.23
38532.96	1381003.22	532332.44
38686.37	1381007.98	532322.33
38656.59	1381008.60	532323.11
38666.29	1381009.22	532323.90
38691.92	1381009.83	532324.68
38710.15	1381010.45	532325.47
38732.36	1381011.07	532326.26
38744.36	1381011.69	532327.04
38756.49	1381012.31	532327.83
38767.84	1381012.93	532328.61
38784.76	1381013.55	532329.40
38801.22	1381014.17	532330.18
38814.03	1381014.79	532330.97
38828.43	1381015.41	532331.75

38842.73	1381016.02	532332.54
38858.15	1381016.64	532333.32
38875.01	1381017.26	532334.11
38889.61	1381017.88	532334.89
38905.29	1381018.50	532335.68
38922.78	1381019.12	532336.47
38930.39	1381019.74	532337.25
38948.61	1381020.36	532338.04
38964.35	1381020.98	532338.82
38981.69	1381021.59	532339.61
38997.08	1381022.21	532340.39
39035.23	1381022.83	532341.18
39066.63	1381023.45	532341.96
39083.63	1381024.07	532342.75
39108.49	1381024.69	532343.53
39119.84	1381025.31	532344.32
39126.90	1381025.93	532345.11
39146.47	1381026.55	532345.89
39174.37	1381027.17	532346.68
39191.85	1381027.78	532347.46
39195.39	1381028.40	532348.25
39210.47	1381029.02	532349.03
39229.43	1381029.64	532349.82
39248.84	1381030.26	532350.60
39266.93	1381030.88	532351.39
39286.72	1381031.50	532352.17
39281.78	1381032.12	532352.96
39287.21	1381032.74	532353.75
39311.35	1381033.35	532354.53
39333.22	1381033.97	532355.32
39350.84	1381034.59	532356.10
39363.95	1381035.21	532356.89
39357.39	1381035.83	532357.67
39356.92	1381036.45	532358.46
39367.46	1381037.07	532359.24
39338.39	1381037.69	532360.03
39334.17	1381038.31	532360.81
39334.59	1381038.93	532361.60
39334.55	1381039.54	532362.38
39316.59	1381040.16	532363.17
39308.25	1381040.78	532363.96
39291.92	1381041.40	532364.74
39276.87	1381042.02	532365.53
39291.63	1381049.87	532359.34
39314.25	1381049.26	532358.55
39335.96	1381048.64	532357.77
39358.02	1381048.02	532356.98
39374.52	1381047.40	532356.20
39404.39	1381046.78	532355.41
39428.59	1381046.16	532354.62
39462.65	1381045.54	532353.84
39488.45	1381044.92	532353.05
39512.66	1381044.30	532352.27
39542.00	1381043.68	532351.48
39571.52	1381043.07	532350.70
39595.12	1381042.45	532349.91
39622.63	1381041.83	532349.13
39641.73	1381041.21	532348.34
39655.00	1381040.59	532347.56
39665.64	1381039.97	532346.77
39672.01	1381039.35	532345.98
39670.17	1381038.73	532345.20
39685.67	1381038.11	532344.41
39673.26	1381037.50	532343.63

39656.98	1381036.26	532342.06
39663.48	1381035.64	532341.27
39648.21	1381035.02	532340.49
39634.14	1381034.40	532339.70
39645.46	1381033.78	532338.92
39627.24	1381033.16	532338.13
39585.82	1381032.54	532337.35
39536.17	1381031.92	532336.56
39480.74	1381031.31	532335.77
39452.58	1381030.69	532334.99
39411.30	1381030.07	532334.20
39373.22	1381029.45	532333.42
39340.15	1381028.83	532332.63
39308.23	1381028.21	532331.85
39277.78	1381027.59	532331.06
39252.93	1381026.97	532330.28
39221.93	1381026.35	532329.49
39183.59	1381025.74	532328.71
39147.11	1381025.12	532327.92
39113.77	1381024.50	532327.13
39088.04	1381023.88	532326.35
39061.54	1381023.26	532325.56
39048.09	1381022.64	532324.78
39040.06	1381022.02	532323.99
39029.66	1381021.40	532323.21
39014.28	1381020.78	532322.42
38984.87	1381020.16	532321.64
38940.23	1381019.55	532320.85
38890.91	1381018.93	532320.07
38866.04	1381018.31	532319.28
38845.50	1381017.69	532318.49
38825.78	1381017.07	532317.71
38804.27	1381016.45	532316.92
38779.35	1381015.83	532316.14
38757.43	1381015.21	532315.35
38734.90	1381014.59	532314.57
38715.45	1381013.98	532313.78
38700.42	1381013.36	532313.00
38682.65	1381012.74	532312.21
38930.52	1381021.83	532307.59
38943.84	1381022.45	532308.38
38990.17	1381023.07	532309.16
39019.19	1381023.69	532309.95
39056.71	1381024.31	532310.73
39101.90	1381024.92	532311.52
39171.83	1381025.54	532312.31
39231.05	1381026.16	532313.09
39268.09	1381026.78	532313.88
39282.33	1381027.40	532314.66
39290.10	1381028.02	532315.45
39295.45	1381028.64	532316.23
39310.91	1381029.26	532317.02
39334.06	1381029.88	532317.80
39370.73	1381030.49	532318.59
39407.91	1381031.11	532319.37
39449.79	1381031.73	532320.16
39495.98	1381032.35	532320.95
39554.38	1381032.97	532321.73
39607.27	1381033.59	532322.52
39688.61	1381034.21	532323.30
39735.57	1381034.83	532324.09
39781.05	1381035.45	532324.87
39820.59	1381036.07	532325.66
39865.29	1381036.68	532326.44

39901.77	1381037.30	532327.23
39923.60	1381037.92	532328.01
39953.72	1381038.54	532328.80
39957.07	1381039.16	532329.58
39976.88	1381039.78	532330.37
39956.25	1381040.40	532331.16
39939.01	1381041.02	532331.94
39916.50	1381041.64	532332.73
39896.24	1381042.25	532333.51
39873.90	1381042.87	532334.30
39838.20	1381043.49	532335.08
39803.51	1381044.11	532335.87
39767.57	1381044.73	532336.65
39737.32	1381045.35	532337.44
39709.96	1381045.97	532338.22
39678.09	1381046.59	532339.01
39650.90	1381047.21	532339.80
39620.02	1381047.83	532340.58
39562.96	1381048.44	532341.37
39523.23	1381049.06	532342.15
39482.48	1381049.68	532342.94
39442.50	1381050.30	532343.72
39422.60	1381050.92	532344.51
39388.10	1381051.54	532345.29
39360.86	1381052.16	532346.08
39335.92	1381052.78	532346.86
39316.80	1381053.40	532347.65
39286.71	1381054.01	532348.44
39261.94	1381054.63	532349.22
39243.34	1381055.25	532350.01
39234.62	1381055.87	532350.79
39225.06	1381056.49	532351.58
39209.69	1381057.11	532352.36
39185.85	1381057.73	532353.15
39089.25	1381065.58	532346.96
39086.29	1381064.96	532346.17
39090.05	1381064.34	532345.39
39096.71	1381063.73	532344.60
39107.59	1381063.11	532343.82
39120.26	1381062.49	532343.03
39139.01	1381061.87	532342.25
39154.04	1381061.25	532341.46
39182.15	1381060.63	532340.67
39204.63	1381060.01	532339.89
39230.14	1381059.39	532339.10
39252.03	1381058.77	532338.32
39275.97	1381058.16	532337.53
39297.06	1381057.54	532336.75
39331.01	1381056.92	532335.96
39355.82	1381056.30	532335.18
39389.20	1381055.68	532334.39
39419.04	1381055.06	532333.61
39446.25	1381054.44	532332.82
39468.52	1381053.82	532332.03
39499.23	1381053.20	532331.25
39525.98	1381052.58	532330.46
39556.58	1381051.97	532329.68
39587.94	1381051.35	532328.89
39620.91	1381050.73	532328.11
39649.40	1381050.11	532327.32
39688.24	1381049.49	532326.54
39729.81	1381048.87	532325.75
39774.30	1381048.25	532324.97
39823.74	1381047.63	532324.18

39930.44	1381046.40	532322.61
39985.63	1381045.78	532321.82
40060.14	1381045.16	532321.04
40091.39	1381044.54	532320.25
40128.21	1381043.92	532319.47
40157.32	1381043.30	532318.68
40161.82	1381042.68	532317.90
40138.81	1381042.06	532317.11
40113.53	1381041.44	532316.33
40037.33	1381040.82	532315.54
39946.87	1381040.21	532314.76
39876.07	1381039.59	532313.97
39767.11	1381038.97	532313.18
39683.10	1381038.35	532312.40
39612.97	1381037.73	532311.61
39555.31	1381037.11	532310.83
39491.42	1381036.49	532310.04
39440.46	1381035.87	532309.26
39384.31	1381035.25	532308.47
39342.75	1381034.64	532307.69
39306.73	1381034.02	532306.90
39278.38	1381033.40	532306.12
39249.67	1381032.78	532305.33
39224.53	1381032.16	532304.54
39185.30	1381031.54	532303.76
39140.78	1381030.92	532302.97
39108.16	1381030.30	532302.19
39082.05	1381029.68	532301.40
39052.55	1381029.06	532300.62
39025.23	1381028.45	532299.83
38993.49	1381027.83	532299.05
38957.59	1381027.21	532298.26
38928.91	1381026.59	532297.48
38901.12	1381025.97	532296.69
38880.13	1381025.35	532295.91
38861.74	1381024.73	532295.12
38845.35	1381024.11	532294.33
38834.81	1381023.49	532293.55
38818.68	1381022.88	532292.76
38789.04	1381022.26	532291.98
38976.34	1381030.11	532285.79
38836.01	1381030.73	532286.57
38833.16	1381030.73	532286.57
38856.79	1381031.35	532287.36
38884.45	1381031.97	532288.14
38910.65	1381032.59	532288.93
38944.56	1381033.21	532289.72
38963.04	1381033.82	532290.50
39002.45	1381034.44	532291.29
39039.82	1381035.06	532292.07
39076.50	1381035.68	532292.86
39110.42	1381036.30	532293.64
39147.40	1381036.92	532294.43
39187.17	1381037.54	532295.21
39231.18	1381038.16	532296.00
39299.17	1381038.78	532296.78
39338.99	1381039.39	532297.57
39379.60	1381040.01	532298.36
39443.85	1381040.63	532299.14
39510.98	1381041.25	532299.93
39595.09	1381041.87	532300.71
39678.21	1381042.49	532301.50
39831.77	1381043.11	532302.28
39885.71	1381043.73	532303.07

39980.66	1381044.35	532303.85
40107.60	1381044.97	532304.64
40214.65	1381045.58	532305.42
40271.06	1381046.20	532306.21
40288.88	1381046.82	532307.00
40274.88	1381047.44	532307.78
40229.91	1381048.06	532308.57
40172.86	1381048.68	532309.35
40097.16	1381049.30	532310.14
40004.29	1381049.92	532310.92
39934.36	1381050.54	532311.71
39880.11	1381051.15	532312.49
39814.86	1381051.77	532313.28
39782.06	1381052.39	532314.06
39749.98	1381053.01	532314.85
39693.38	1381053.63	532315.63
39646.90	1381054.25	532316.42
39603.00	1381054.87	532317.21
39603.00	1381055.49	532317.99
39526.00	1381056.11	532318.78
39484.76	1381056.73	532319.56
39426.89	1381057.34	532320.35
39387.42	1381057.96	532321.13
39349.69	1381058.58	532321.92
39314.80	1381059.20	532322.70
39284.67	1381059.82	532323.49
39252.17	1381060.44	532324.27
39224.39	1381061.06	532325.06
39191.85	1381061.68	532325.85
39180.74	1381062.30	532326.63
39180.74	1381062.91	532327.42
39126.84	1381063.53	532328.20
39110.15	1381064.15	532328.99
39093.05	1381064.77	532329.77
39083.10	1381065.39	532330.56
39078.79	1381066.01	532331.34
39071.97	1381066.63	532332.13
39067.92	1381067.25	532332.91
39064.04	1381067.87	532333.70
39059.79	1381068.49	532334.49
39062.89	1381069.10	532335.27
39061.36	1381069.72	532336.06
39056.90	1381070.34	532336.84
39051.22	1381070.96	532337.63
39070.08	1381071.58	532338.41
39082.01	1381072.20	532339.20
39092.34	1381072.82	532339.98
39116.25	1381073.44	532340.77
39699.18	1381097.00	532322.20
39791.29	1381096.38	532321.41
39860.95	1381095.76	532320.63
39911.76	1381095.14	532319.84
39931.30	1381094.52	532319.06
39925.86	1381093.91	532318.27
39924.70	1381093.29	532317.49
39906.66	1381092.67	532316.70
39883.15	1381092.05	532315.92
39854.13	1381091.43	532315.13
39817.04	1381090.81	532314.35
39811.32	1381090.19	532313.56
39816.27	1381089.57	532312.77
39821.00	1381088.95	532311.99
39860.40	1381088.33	532311.20
39913.22	1381087.10	532309.63

39920.76	1381086.48	532308.85
39905.07	1381085.86	532308.06
39871.17	1381085.24	532307.28
39835.71	1381084.62	532306.49
39795.88	1381084.00	532305.71
39745.08	1381083.38	532304.92
39699.41	1381082.76	532304.14
39647.59	1381082.15	532303.35
39576.48	1381081.53	532302.56
39518.01	1381080.91	532301.78
39470.06	1381080.29	532300.99
39441.82	1381079.67	532300.21
39420.29	1381079.05	532299.42
39405.72	1381078.43	532298.64
39400.40	1381077.81	532297.85
39399.63	1381077.19	532297.07
39389.75	1381076.57	532296.28
39395.16	1381075.96	532295.50
39408.36	1381075.34	532294.71
39414.31	1381074.72	532293.92
39433.21	1381074.10	532293.14
39432.85	1381073.48	532292.35
39430.99	1381072.86	532291.57
39547.44	1381072.24	532290.78
39603.54	1381071.62	532290.00
39682.68	1381071.00	532289.21
39733.11	1381070.39	532288.43
39710.11	1381069.77	532287.64
39620.04	1381069.15	532286.86
39620.04	1381068.53	532286.07
39568.62	1381067.91	532285.28
39604.97	1381067.29	532284.50
39703.58	1381066.67	532283.71
39847.74	1381066.05	532282.93
39951.87	1381065.43	532282.14
39951.87	1381064.81	532281.36
40001.91	1381064.20	532280.57
39950.71	1381063.58	532279.79
39795.51	1381062.96	532279.00
39633.88	1381062.34	532278.22
39493.81	1381061.72	532277.43
39390.81	1381061.10	532276.65
39311.90	1381060.48	532275.86
39260.67	1381059.86	532275.07
39225.49	1381059.24	532274.29
39205.71	1381058.62	532273.50
39171.58	1381058.01	532272.72
39133.10	1381057.39	532271.93
39101.42	1381056.77	532271.15
39067.11	1381056.15	532270.36
39034.53	1381055.53	532269.58
39005.45	1381054.91	532268.79
38978.02	1381054.29	532268.01
38974.84	1381053.67	532267.22
38781.25	1381034.87	532275.67
38800.49	1381035.49	532276.46
38812.95	1381036.11	532277.24
38833.21	1381036.73	532278.03
38851.54	1381037.35	532278.81
38873.00	1381037.96	532279.60
38891.50	1381038.58	532280.38
38922.84	1381039.20	532281.17
38948.00	1381039.82	532281.96
38982.50	1381040.44	532282.74

39022.32	1381041.06	532283.53
39061.40	1381041.68	532284.31
39096.38	1381042.30	532285.10
39140.37	1381042.92	532285.88
39189.38	1381043.54	532286.67
39243.65	1381044.15	532287.45
39294.50	1381044.77	532288.24
39347.28	1381045.39	532289.02
39430.24	1381046.01	532289.81
39498.01	1381046.63	532290.60
39574.87	1381047.25	532291.38
39653.33	1381047.87	532292.17
39732.54	1381048.49	532292.95
39872.48	1381049.11	532293.74
39969.27	1381049.72	532294.52
40115.23	1381050.34	532295.31
40186.15	1381050.96	532296.09
40235.90	1381051.58	532296.88
40265.80	1381052.20	532297.66
40293.67	1381052.82	532298.45
40273.25	1381053.44	532299.23
40053.96	1381054.06	532300.02
39883.53	1381054.68	532300.81
39775.72	1381055.30	532301.59
39705.53	1381055.91	532302.38
39684.36	1381056.53	532303.16
39667.87	1381057.15	532303.95
39669.53	1381057.77	532304.73
39664.59	1381058.39	532305.52
39629.62	1381059.01	532306.30
39603.32	1381059.63	532307.09
39586.00	1381060.25	532307.87
39549.43	1381060.87	532308.66
39519.83	1381061.48	532309.45
39475.19	1381062.10	532310.23
39402.94	1381062.72	532311.02
39327.58	1381063.34	532311.80
39271.67	1381063.96	532312.59
39223.73	1381064.58	532313.37
39183.52	1381065.20	532314.16
39140.61	1381065.82	532314.94
39140.61	1381066.44	532315.73
39126.74	1381067.06	532316.51
39110.06	1381067.67	532317.30
39100.03	1381068.29	532318.09
39101.92	1381068.91	532318.87
39100.80	1381069.53	532319.66
39091.30	1381070.15	532320.44
39072.35	1381070.77	532321.23
39062.15	1381071.39	532322.01
39062.99	1381072.01	532322.80
39074.66	1381072.63	532323.58
39089.96	1381073.25	532324.37
39089.96	1381073.86	532325.15
39113.33	1381074.48	532325.94
39137.25	1381075.10	532326.72
39161.88	1381075.72	532327.51
39211.72	1381076.34	532328.30
39248.84	1381076.96	532329.08
39276.20	1381077.58	532329.87
39322.35	1381078.20	532330.65
39374.97	1381078.82	532331.44
39468.05	1381080.05	532333.01
39540.41	1381080.67	532333.79

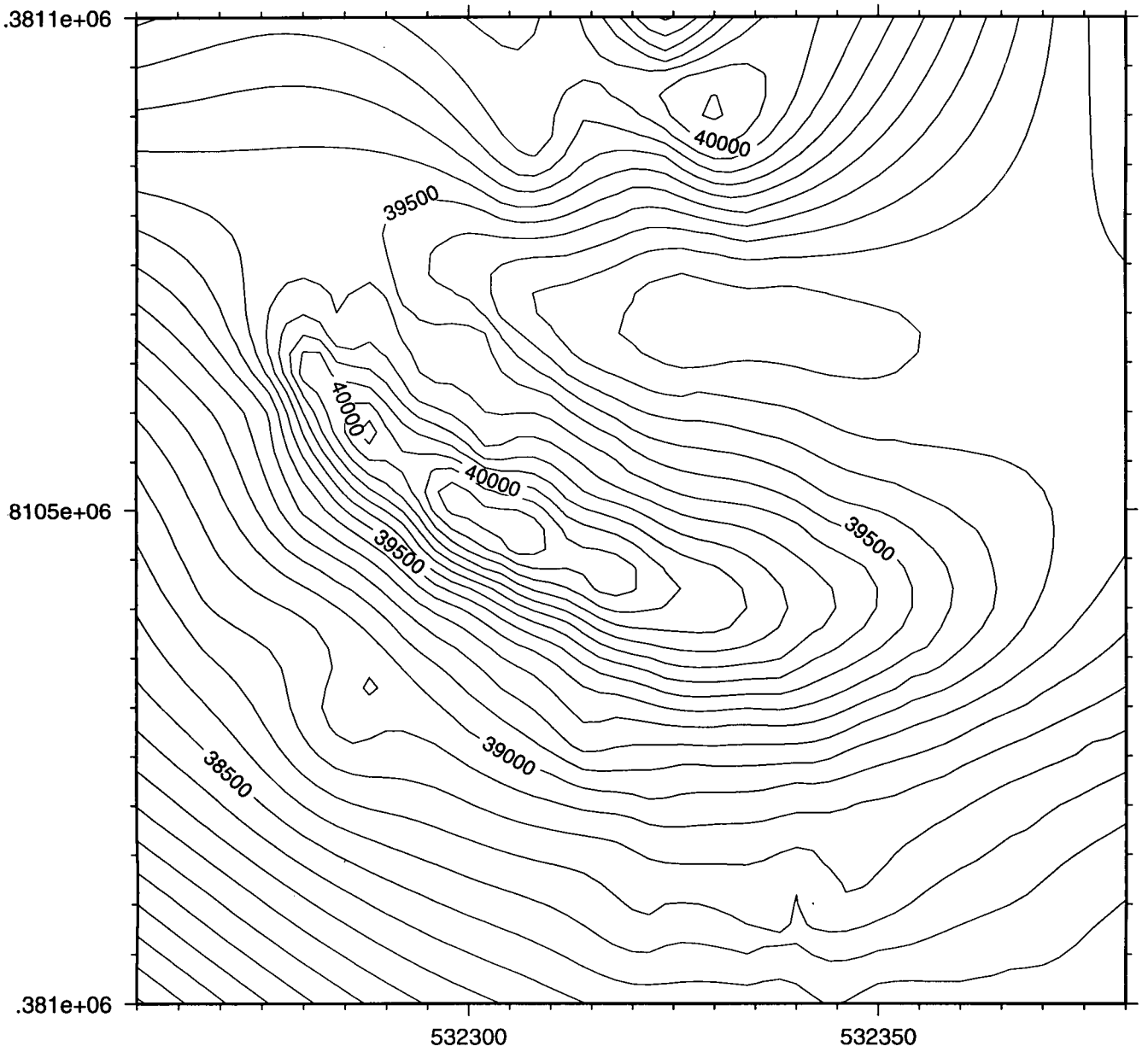
40104.55	1381081.29	532334.58
40091.97	1381089.15	532328.39
40033.52	1381088.53	532327.60
39975.41	1381087.91	532326.82
39909.86	1381087.29	532326.03
39810.12	1381086.67	532325.25
39717.06	1381086.05	532324.46
39641.84	1381085.43	532323.68
39585.44	1381084.81	532322.89
39585.44	1381084.19	532322.11
39539.14	1381083.58	532321.32
39499.54	1381082.96	532320.54
39480.82	1381082.34	532319.75
39463.29	1381081.72	532318.96
39457.44	1381081.10	532318.18
39445.42	1381080.48	532317.39
39415.37	1381079.86	532316.61
39415.37	1381079.24	532315.82
39388.25	1381078.62	532315.04
39365.79	1381078.00	532314.25
39337.98	1381077.39	532313.47
39330.30	1381076.77	532312.68
39307.56	1381076.15	532311.90
39289.88	1381075.53	532311.11
39255.83	1381074.91	532310.32
39236.79	1381074.29	532309.54
39214.16	1381073.67	532308.75
39214.16	1381073.05	532307.97
39209.76	1381072.43	532307.18
39207.79	1381071.82	532306.40
39219.93	1381071.20	532305.61
39239.55	1381070.58	532304.83
39275.80	1381069.96	532304.04
39338.63	1381069.34	532303.26
39420.54	1381068.72	532302.47
39514.72	1381068.10	532301.68
39543.00	1381067.48	532300.90
39543.00	1381066.86	532300.11
39565.24	1381066.24	532299.33
39580.87	1381065.63	532298.54
39573.95	1381065.01	532297.76
39561.31	1381064.39	532296.97
39578.25	1381063.77	532296.19
39557.44	1381063.15	532295.40
39541.72	1381062.53	532294.62
39646.95	1381061.91	532293.83
39754.94	1381061.29	532293.05
39754.94	1381060.67	532292.26
39869.61	1381060.05	532291.47
39918.47	1381059.44	532290.69
39997.22	1381058.82	532289.90
40078.53	1381058.20	532289.12
40184.48	1381057.58	532288.33
40193.38	1381056.96	532287.55
40110.26	1381056.34	532286.76
39940.44	1381055.72	532285.98
39822.90	1381055.10	532285.19
39822.90	1381054.48	532284.41
39646.10	1381053.87	532283.62
39543.77	1381053.25	532282.83
39465.24	1381052.63	532282.05
39396.73	1381052.01	532281.26
39362.62	1381051.39	532280.48
39339.52	1381050.77	532279.69

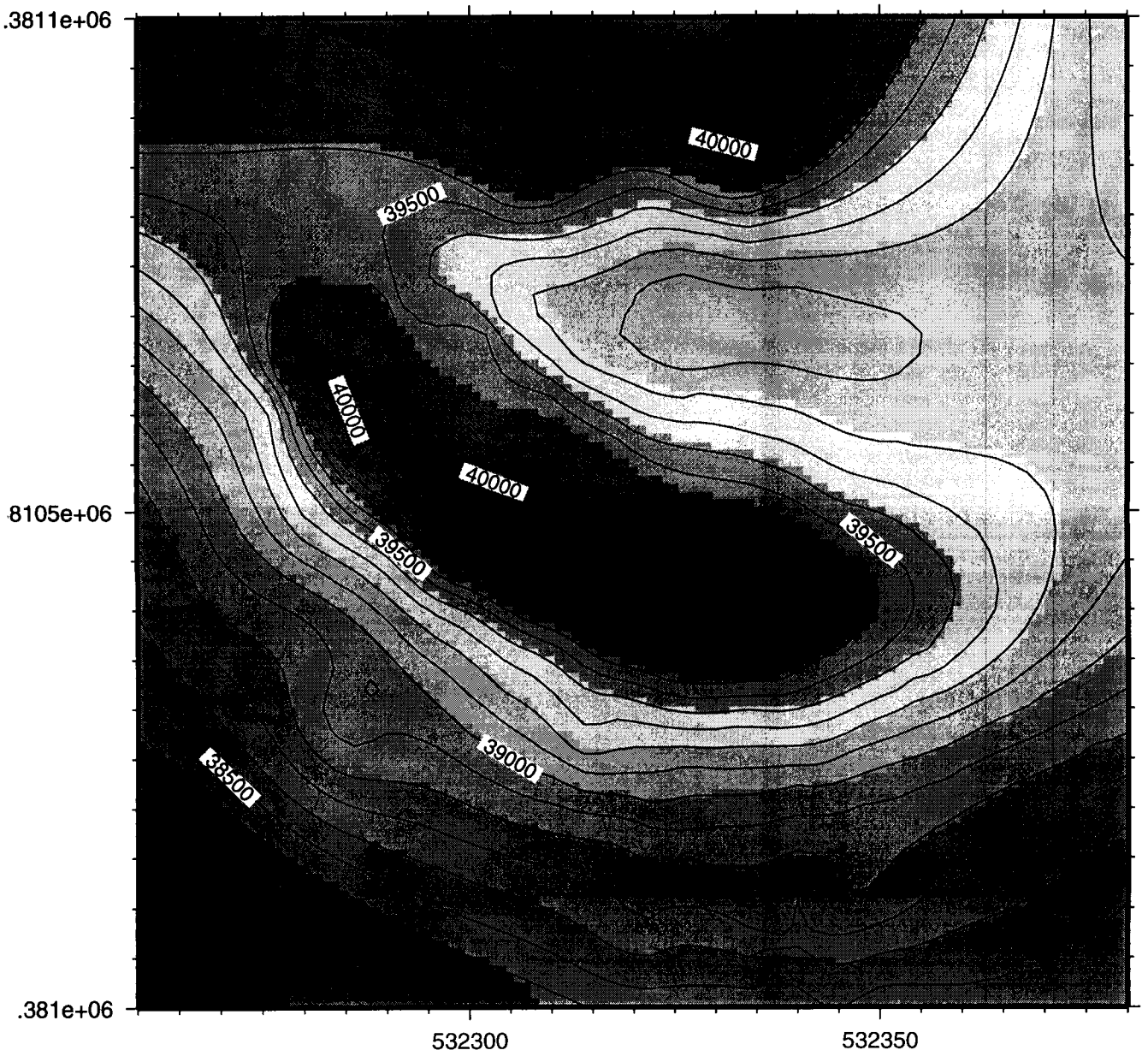
39282.88	1381050.15	532278.91
39222.15	1381049.53	532278.12
39172.76	1381048.91	532277.34
39130.85	1381048.29	532276.55
39085.81	1381047.68	532275.77
39044.63	1381047.06	532274.98
39007.80	1381046.44	532274.19
38984.97	1381045.82	532273.41
38945.44	1381045.20	532272.62
38918.80	1381044.58	532271.84
38900.27	1381043.96	532271.05
38878.00	1381043.34	532270.27
38861.00	1381042.72	532269.48
38838.98	1381042.11	532268.70
38809.06	1381041.49	532267.91
38785.24	1381040.87	532267.13
38741.82	1381040.25	532266.34
38712.00	1381039.63	532265.56
38579.71	1381018.46	532384.10
38588.16	1381019.24	532383.48
38610.09	1381020.03	532382.86
38628.49	1381020.81	532382.24
38648.71	1381021.60	532381.62
38672.66	1381022.38	532381.00
38692.19	1381023.17	532380.38
38716.73	1381023.96	532379.76
38743.27	1381024.74	532379.14
38766.84	1381025.53	532378.52
38787.22	1381026.31	532377.91
38813.37	1381027.10	532377.29
38837.41	1381027.88	532376.67
38864.34	1381028.67	532376.05
38889.39	1381029.45	532375.43
38914.76	1381030.24	532374.81
38940.06	1381031.02	532374.19
38971.54	1381031.81	532373.57
38997.39	1381032.59	532372.95
39021.55	1381033.38	532372.34
39048.61	1381034.17	532371.72
39075.57	1381034.95	532371.10
39107.23	1381035.74	532370.48
39137.93	1381036.52	532369.86
39162.28	1381037.31	532369.24
39184.16	1381038.09	532368.62
39204.37	1381038.88	532368.00
39222.30	1381039.66	532367.38
39238.05	1381040.45	532366.76
39248.86	1381041.23	532366.15
39266.65	1381042.02	532365.53
39280.28	1381042.81	532364.91
39293.92	1381043.59	532364.29
39302.72	1381044.38	532363.67
39307.99	1381045.16	532363.05
39314.28	1381045.95	532362.43
39320.48	1381046.73	532361.81
39314.49	1381047.52	532361.19
39305.28	1381048.30	532360.58
39303.03	1381049.09	532359.96
39296.32	1381049.87	532359.34
39282.56	1381050.66	532358.72
39279.09	1381051.45	532358.10
39258.72	1381053.02	532356.86
39246.69	1381053.80	532356.24
39237.17	1381054.59	532355.62

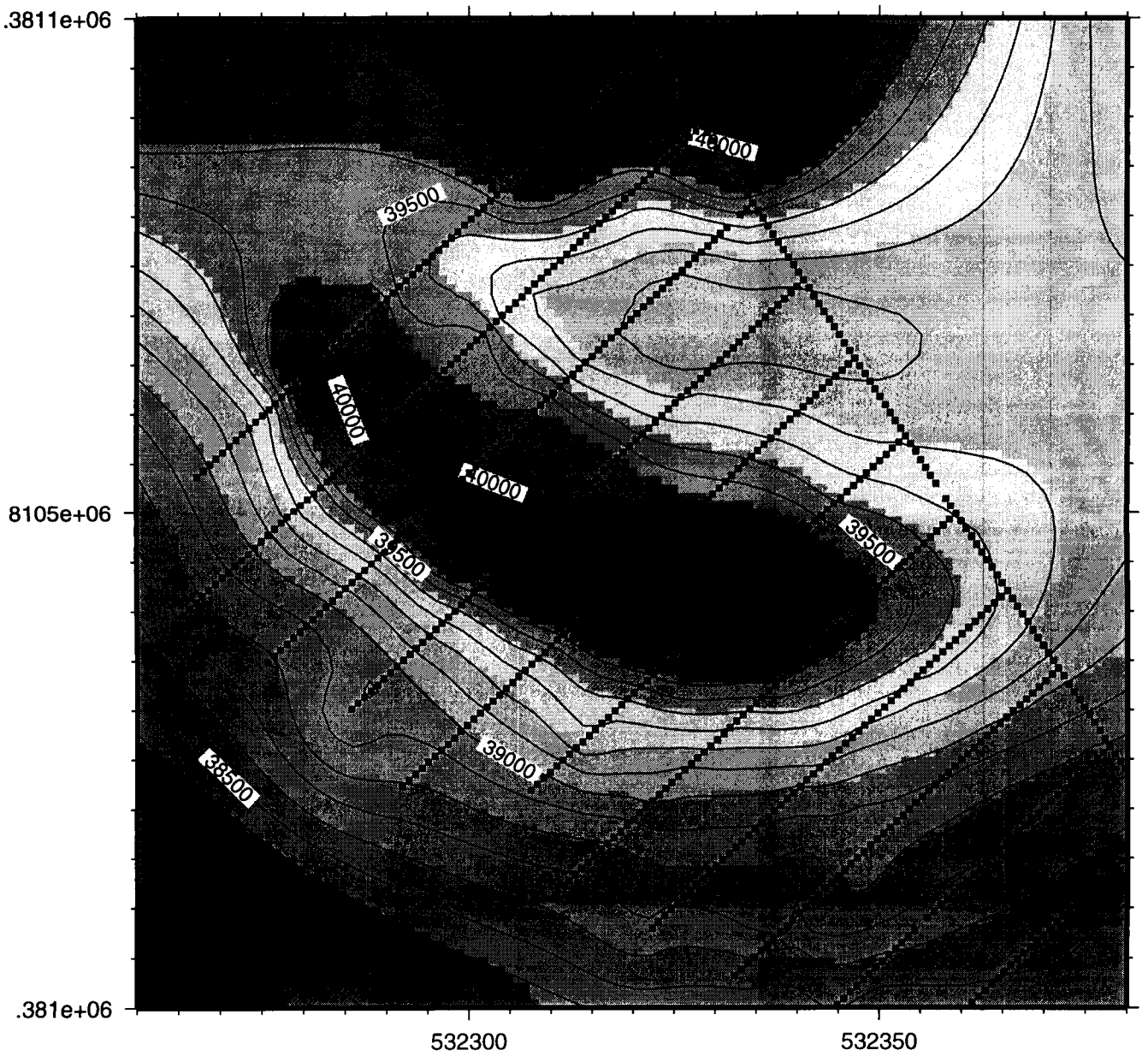
39223.15	1381055.37	532355.00
39206.24	1381056.16	532354.39
39193.73	1381056.94	532353.77
39184.16	1381057.73	532353.15
39164.67	1381058.51	532352.53
39151.19	1381059.30	532351.91
39140.58	1381060.08	532351.29
39128.83	1381060.87	532350.67
39119.08	1381061.66	532350.05
39110.30	1381062.44	532349.43
39100.36	1381063.23	532348.81
39090.82	1381064.01	532348.20
39086.30	1381064.80	532347.58
39081.09	1381065.58	532346.96
39072.86	1381066.37	532346.34
39069.78	1381067.15	532345.72
39068.92	1381067.94	532345.10
39069.52	1381068.72	532344.48
39070.79	1381069.51	532343.86
39074.28	1381070.30	532343.24
39082.70	1381071.08	532342.63
39091.65	1381071.87	532342.01
39104.25	1381072.65	532341.39
39121.49	1381073.44	532340.77
39147.10	1381074.22	532340.15
39176.60	1381075.01	532339.53
39207.47	1381075.79	532338.91
39242.75	1381076.58	532338.29
39281.84	1381077.36	532337.67
39333.34	1381078.15	532337.05
39385.50	1381078.94	532336.44
39441.03	1381079.72	532335.82
39493.05	1381080.51	532335.20
39542.83	1381081.29	532334.58
39612.20	1381082.08	532333.96
39660.51	1381082.86	532333.34
39709.47	1381083.65	532332.72
39814.20	1381084.43	532332.10
39881.78	1381085.22	532331.48
39935.64	1381086.00	532330.87
39984.94	1381086.79	532330.25
40043.67	1381087.57	532329.63
40063.39	1381088.36	532329.01
40093.18	1381089.15	532328.39
40089.71	1381089.93	532327.77
40106.43	1381090.72	532327.15
40090.24	1381091.50	532326.53
40060.35	1381092.29	532325.91
40044.05	1381093.07	532325.29
39993.32	1381093.86	532324.68
39929.54	1381094.64	532324.06
39751.12	1381095.43	532323.44
39709.27	1381096.21	532322.82
39665.13	1381097.00	532322.20
38403.05	1380993.70	532352.68
38405.32	1380994.32	532353.46
38404.67	1380994.94	532354.25
38409.58	1380995.56	532355.03
38418.65	1380996.17	532355.82
38429.76	1380996.79	532356.61
38437.25	1380997.41	532357.39
38449.43	1380998.03	532358.18
38460.11	1380998.65	532358.96
38466.28	1380999.27	532359.75

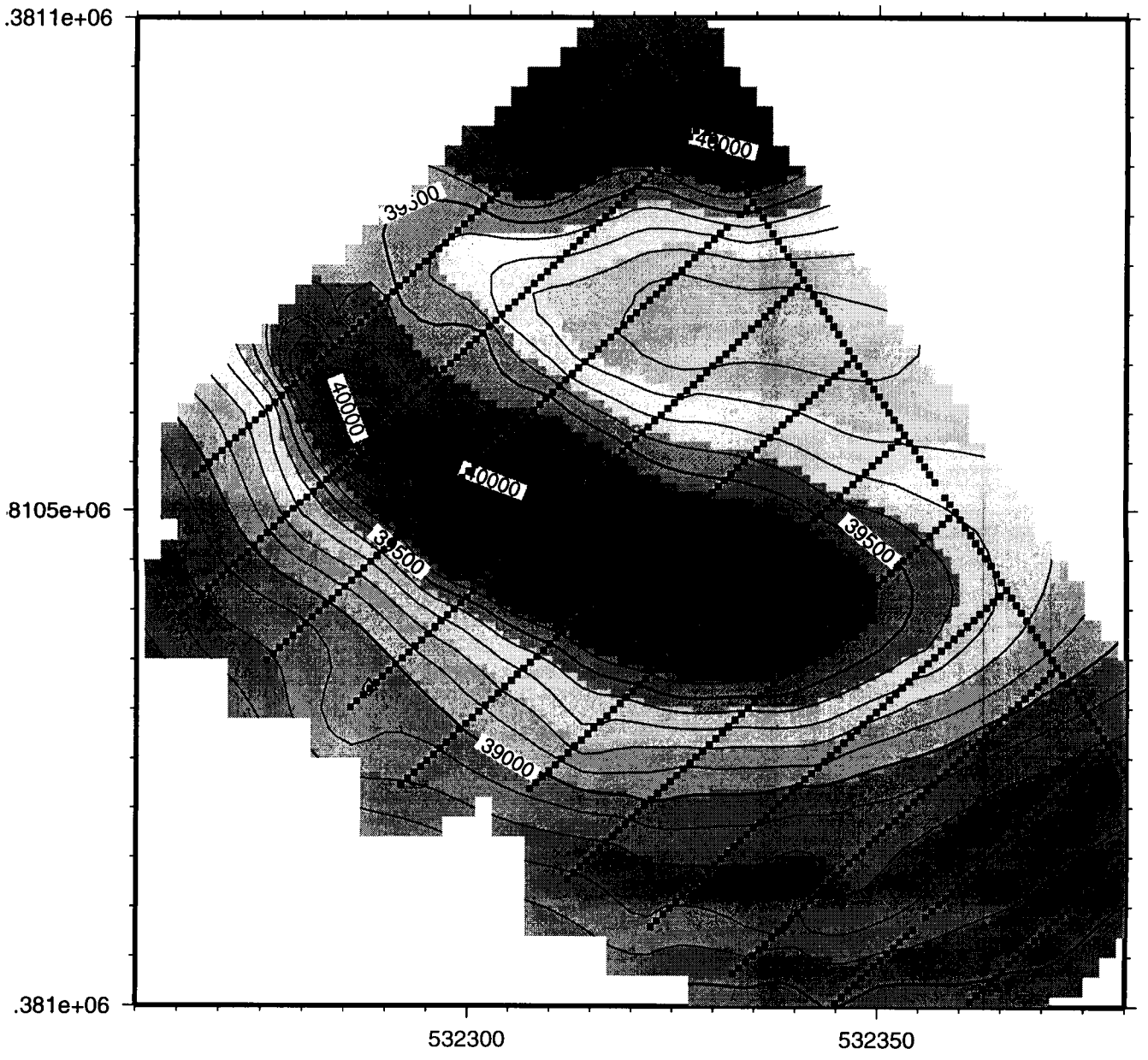
38470.62	1380999.89	532360.53
38478.40	1381000.51	532361.32
38479.26	1381001.13	532362.10
38487.70	1381001.75	532362.89
38495.45	1381002.36	532363.67
38504.19	1381002.98	532364.46
38507.91	1381003.60	532365.24
38509.89	1381004.22	532366.03
38519.92	1381004.84	532366.82
38532.85	1381005.46	532367.60
38536.70	1381006.08	532368.39
38546.47	1381006.70	532369.17
38550.03	1381007.32	532369.96
38549.01	1381007.93	532370.74
38545.54	1381008.55	532371.53
38536.49	1381009.17	532372.31
38538.18	1381009.79	532373.10
38539.10	1381010.41	532373.88
38540.30	1381011.03	532374.67
38540.14	1381011.65	532375.46
38544.39	1381012.27	532376.24
38549.00	1381012.89	532377.03
38549.71	1381013.51	532377.81
38554.78	1381014.12	532378.60
38552.92	1381014.74	532379.38
38556.62	1381015.36	532380.17
38556.28	1381015.98	532380.95
38556.66	1381016.60	532381.74
38556.08	1381017.22	532382.52
38560.58	1381017.84	532383.31
38567.20	1381018.46	532384.10
38782.44	1381026.31	532377.91
38768.80	1381025.69	532377.12
38759.21	1381025.07	532376.33
38764.24	1381024.45	532375.55
38766.86	1381023.84	532374.76
38762.26	1381023.22	532373.98
38746.99	1381022.60	532373.19
38735.81	1381021.98	532372.41
38739.23	1381021.36	532371.62
38734.81	1381020.74	532370.84
38719.57	1381020.12	532370.05
38712.81	1381019.50	532369.27
38709.15	1381018.88	532368.48
38716.41	1381018.27	532367.70
38708.43	1381017.65	532366.91
38699.25	1381017.03	532366.12
38699.33	1381016.41	532365.34
38698.78	1381015.79	532364.55
38690.21	1381015.17	532363.77
38689.73	1381014.55	532362.98
38692.07	1381013.93	532362.20
38688.96	1381013.31	532361.41
38680.93	1381012.69	532360.63
38674.68	1381012.08	532359.84
38673.99	1381011.46	532359.06
38672.10	1381010.84	532358.27
38662.12	1381010.22	532357.48
38658.35	1381009.60	532356.70
38657.67	1381008.98	532355.91
38646.16	1381007.74	532354.34
38633.61	1381007.12	532353.56
38625.06	1381006.50	532352.77
38611.42	1381005.89	532351.99

38594.44	1381005.27	532351.20
38578.85	1381004.65	532350.42
38553.32	1381004.03	532349.63
38550.83	1381003.41	532348.84
38539.10	1381002.79	532348.06
38522.18	1381002.17	532347.27
38518.48	1381001.55	532346.49
38516.17	1381000.93	532345.70
38509.60	1381000.32	532344.92
38503.71	1380999.70	532344.13
38493.59	1380999.08	532343.35
38478.78	1380998.46	532342.56
38460.41	1380997.84	532341.78
38456.12	1380997.22	532340.99
38451.19	1380996.60	532340.21
38448.87	1380995.98	532339.42
38438.66	1380995.36	532338.63


.3811e+06
8105e+06
.381e+06
532300
532350
40000
39500
40000
40000
39500
39500
39000
38500


.3811e+06
8105e+06
.381e+06
532300
532350
40000
39500
40000
40000
39500
39500
39000
39500


.3811e+06
8105e+06
.381e+06
532300
532350
44000
39500
40000
40000
39500
39500
39000
38500


.3811e+06
.8105e+06
.381e+06
532300
532350
40000
39500
40000
40000
39500
39500
39000

Scientific Notebook
20-1402-461
Inititals: CC

Chuck Connor
Oct 4,1999

The following pages contain codes written by C. Connor is java to manipulate data files. These codes include:

ColSwap - takes a strem of input data from a file in the form x,y,z and prints columns in the form z,y,x

File_handler3 - takes input file with three columns and outputs two columns with values in a specified range

Mergesort4 - sorts data be a specified column

Maggraph8 - plot a data file generated through the use of above codes

These codes were written to specifically handle magnetic data sets - but can be used for simialr data sets

```
import java.awt.*;
import java.applet.*;
import java.io.*;

import graph.*;
/*
**************************************************************************
**
**                      Applet maggraph8
**
**************************************************************************
**    Written by Chuck Connor Sept, 1999
**
**    This program gets an input data file and plots the results in
**     a java applet. The user can input the name of the data file
**     directly in the applet
**************************************************************************
*      The input data file must have the form

**  4052354.0, -286.9141
**  4052354.5, -286.8086
**  4052356.25, -289.375
**  4052357.0, -288.9492
**  4052357.0, -288.7266
**  4052357.75, -280.6797
**  4052357.75, -286.5937

*      where the first column is plotted as the "x" axis and the
*      second column is plotted as the "y" axis
**************************************************************************/

public class maggraph8 extends Applet {

      G2Dint graph         = new G2Dint();   // Graph class to do the plotting
      Axis xaxis;
      Axis yaxis;
      DataSet data;


      TextField datainput     = new TextField(20);      // field for inputting data
      Button plot          = new Button("Plot It!"); // Button to plot it.




      public void init() {
         Label title            = new Label("Data Graph",Label.CENTER);
         Panel panel            = new Panel();
         GridBagLayout gridbag  = new GridBagLayout();
         GridBagConstraints  c  = new GridBagConstraints();
         Font font              = new Font("Helvetica",Font.PLAIN,14);

         title.setFont(new Font("TimesRoman",Font.PLAIN,24));


         setLayout(new BorderLayout() );
         add("North",title);
         add("Center",panel);


         datainput.setText(getParameter("DATASET"));
```

```
        panel.setLayout(gridbag);

        Label datalabel   = new Label("Data Set");



        datalabel.setFont(font);


        datainput.setFont(font);
        datainput.setBackground(Color.lightGray);
        plot.setFont(font);
        plot.setBackground(Color.green);

        c.weightx = 1.0;
        c.weighty = 1.0;
        c.gridwidth = 4;
        c.gridwidth=GridBagConstraints.REMAINDER;
        c.fill  =  GridBagConstraints.BOTH;

        gridbag.setConstraints(graph,c);

        c.fill  =  GridBagConstraints.NONE;
        c.weightx=0.0;
        c.weighty=0.0;
        c.gridheight=1;

        c.gridwidth=1;
        c.anchor = GridBagConstraints.EAST;
        gridbag.setConstraints(datalabel,c);

        c.anchor = GridBagConstraints.CENTER;
        c.gridwidth=GridBagConstraints.RELATIVE;
        c.fill  =  GridBagConstraints.HORIZONTAL;
        gridbag.setConstraints(datainput,c);

        c.fill = GridBagConstraints.NONE;
        c.gridwidth=GridBagConstraints.REMAINDER;

        gridbag.setConstraints(plot,c);



        panel.add(graph);
        panel.add(datalabel);
        panel.add(datainput);
        panel.add(plot);



        xaxis = graph.createXAxis();
        xaxis.setTitleText("Distance (m)");
       xaxis.setTitleFont(new Font("Helvetica",Font.PLAIN,18));
       xaxis.setLabelFont(new Font("Helvetica",Font.PLAIN,14));


        yaxis = graph.createYAxis();
        yaxis.setTitleText("nT");
        yaxis.setTitleFont(new Font("Helvetica",Font.PLAIN,18));
        yaxis.setLabelFont(new Font("Helvetica",Font.PLAIN,14));


        data = new DataSet();
```

```
        xaxis.attachDataSet(data);
        yaxis.attachDataSet(data);
        graph.attachDataSet(data);


        graph.setDataBackground(new Color(255,200,175));
        graph.setBackground(new Color(200,150,100));

        plot();

      }



     void plot() {
String datafile;

       try {
            datafile = datainput.getText();
         } catch(Exception e) {
              this.showStatus("Error data file!");
              System.out.println("data file error "+e.getMessage());
              return;
         }


         double [] d = new double[10000];
       int token;
                int j=0;
         try {
         StreamTokenizer stok = new StreamTokenizer (
                new FileInputStream (datafile));

         // Treat a space as a normal character.
          stok.whitespaceChars(',', ',');

         // token is filled with a code indicating type of item
         // just read.


         while ( ( token = stok.nextToken() ) != stok.TT_EOF ) {

             d[j] = stok.nval - 4053000.0;
             token  = stok.nextToken();
             d[j+1] = stok.nval+ 300.0;
             j+=2;

         }

        } catch ( IOException e ) {
        System.err.println ( e );
        return;
    }

         System.out.println(j + "," + d[j]);
         data.deleteData();

         try {
               data.append(d,j/2);
         } catch(Exception e) {
             this.showStatus("Error while appending data!");
             System.out.println("Error while appending data!");
             return;
```

```
        }


        graph.repaint();
      }



    public boolean action(Event e, Object a) {

        if(e.target instanceof Button) {
            if( plot.equals(e.target) ) {
                 plot();
                 return true;
            }
        }


        return false;
      }



}
```

```
// TokenizerText.java
import java.io.*;

// ---------------------------------------------------------------------
//
//  Abstract:  Code implements example of StreamTokenizer class,
//             and reads columns of numbers separated by commas
//             or by white space. Code then outputs same data with the
//             columns swapped
//  Revised:  09 Sept 1999 --> new create, by:
//
//                             C. Connor / #20
//                             southwest research institute
//                             po drawer 28510
//                             san antonio tx 78228
//                             512-522-6649
//
//
//
//  Notes:  Example originally obtained from
//              R. Martin, #20, who obtained From source code in text: "JAVA by Example"
//          by Jackson and McClellan, (c) 1996, SunSoft Press,
//          Sun Microsystems and Prentice Hall,
//          ISBN 0-13-272295-X.
//
//
// ---------------------------------------------------------------------
//xx  yy  zz --->>> zz  yy  xx
// Input looks like:
//....+....1....+....2....+....3....+....4....+....5....+....6....+....7

//39039.05      1381034.17      532371.72
//39034.79      1381033.55      532370.93
//39036.00      1381032.93      532370.15
//39028.16      1381032.31      532369.36
//39034.43      1381031.69      532368.57
//39031.28      1381031.07      532367.79
//39023.83      1381030.45      532367.00
//39011.08      1381029.83      532366.22
//38997.12      1381029.21      532365.43
//38990.5.0     1381028.60      532364.65
//38976.53      1381027.98      532363.86



public class ColSwap {
  public static void main ( String[] args ) {
    if ( args.length < 1 ) {
      System.err.println ( "Usage: java TokenizerText <src> ");
      return;
    }
    try {
      StreamTokenizer stok = new StreamTokenizer (
        new FileInputStream (args[0]));

      // Treat a space as a normal character.
      stok.whitespaceChars(',', ',');
      int token;
      // token is filled with a code indicating type of item
      // just read.


 while ( ( token = stok.nextToken() ) != stok.TT_EOF ) {
```

```
        switch (token) {
          case stok.TT_NUMBER:
              // If a number is read, the value is placed in
              // the double nval.
              double zz = stok.nval;

              token  = stok.nextToken();
              double yy = stok.nval;

              token  = stok.nextToken();
              double xx = stok.nval;
              System.out.println ( xx  + "  " + yy + "  " + zz);
        }


        break;
 case stok.TT_WORD:
     // If a word is read, the value is placed in
     // the string sval.
     System.out.println ( "Word: " + stok.sval );
     break;
 default:
     break;
 }
    }
  } catch ( IOException e ) {
      System.err.println ( e );
      return;
  }
  }
}
```

```
// TokenizerText.java
import java.io.*;

/* This routine sorts two columns of numbers, indexing on the
 *  first column, sorts up to 1000 rows of doubles
 * written by Chuck Connor
 *                 new create - Sept. 1999
 * modified from the code:
 * Mergesort.java by Richard J. Davies
 * from 'Introductory Java for Scientists and Engineers'
 * chapter: 'Programming Practices'
 * section: 'Example - Mergesort'
 *
 * This program performs mergesort on an array.
 */
public class Mergesort4
{
  // The 'mergesort' method performs mergesort.
  // It calls itself recursively to sort the
  // subarrays.
  public static double[][] mergesort(double[][] data)
  {
    if (data.length == 1)
    {
      // If there's only one item then
      // it's already sorted.
      return data;
    }
    else
    {
      // Create two subarrays
      int halflen = data.length/2;

      double[][] done = new double[halflen][2];
      double[][] dtwo = new double[data.length - halflen][2];

      // Copy the data into them
      for (int i=0; i<halflen; i++) {
        done[i][0] = data[i][0];
        done[i][1] = data[i][1];

      }

      for (int i=halflen; i<data.length; i++) {
        dtwo[i-halflen][0] = data[i][0];
        dtwo[i-halflen][1] = data[i][1];

      }

      // Mergesort the subarrays
      done = mergesort(done);
      dtwo = mergesort(dtwo);

      // Create an array to return
      double[][] ans = new double[data.length][2];

      // Merge the subarrays into it
      int pone = 0;
      int ptwo = 0;
      int pans = 0;

      // Repeat until we've transferred all data.
      while (pans < data.length)
      {
```

```
        if (pone < done.length)
        {
          if (ptwo < dtwo.length)
          {
            // If there's data in both arrays,
            // then transfer the lower head
            // to the output list
            if (done[pone][0] < dtwo[ptwo][0])
            {
              ans[pans][0] = done[pone][0];
              ans[pans][1] = done[pone][1];

              pone++;
            }
            else
            {
              ans[pans][0] = dtwo[ptwo][0];
              ans[pans][1] = dtwo[ptwo][1];

              ptwo++;
            }
          }
          else
          {
            // If there's only data in the first
            // subarray then copy that.
            ans[pans][0] = done[pone][0];
            ans[pans][1] = done[pone][1];

            pone++;
          }
        }
        else
        {
          // If there's only data in the second
          // subarray then copy that.
          ans[pans][0] = dtwo[ptwo][0];
          ans[pans][1] = dtwo[ptwo][1];

          ptwo++;
        }

        pans++;
      }

      // return the sorted array.
      return ans;
    }
  }


  // The 'main' method actually
  // generates a random array and sorts it.
  public static void main(String[] args)  {
      int token;
      int i=0;
      double[][] inputdata = new double[5000][2];


      if ( args.length < 1 ) {
          System.err.println ( "Usage: java TokenizerText <src> ");
          return;
      }
```

```
      try {
          StreamTokenizer stok = new StreamTokenizer (new FileInputStream (args[0]));
          // Treat a space as a normal character.
          stok.whitespaceChars(',', ',');



          while ( ( token = stok.nextToken() ) != stok.TT_EOF ) {

         i ++;
         inputdata[i][0]=stok.nval;
         token  = stok.nextToken();
         inputdata[i][1] = stok.nval;


          }

      } catch ( IOException e ) {
          System.err.println ( e );
          return;
      }

      int  size = i-1;
      double[][] data = new double[size][2];

      // Fill it with elements.

      for (i=0; i<size; i++) {
          data[i][0] = inputdata[i][0];
          data[i][1] = inputdata[i][1];
      }
      // Sort it.
      data = mergesort(data);

      // Print the sorted data.
      for (i=0; i<size; i++)
          System.out.println(data[i][0] + ", " + data[i][1]);
  }
}
```

```
// TokenizerText.java
import java.io.*;

// ---------------------------------------------------------------------
//
//  Abstract:  Code implements example of StreamTokenizer class,
//             and reads columns of numbers separated by commas
//             or by white space. This code pulls out values by easting and northing
//
//  Revised:  09 Sept 1999 --> new create, by:
//
//                              C. Connor / #20
//                              southwest research institute
//                              po drawer 28510
//                              san antonio tx 78228
//                              512-522-6649
//
//
//
//  Notes:  Example originally obtained from
//              R. Martin, #20, who obtained From source code in text: "JAVA by Example"
//          by Jackson and McClellan, (c) 1996, SunSoft Press,
//          Sun Microsystems and Prentice Hall,
//          ISBN 0-13-272295-X.
//
//
// ---------------------------------------------------------------------
//
// Input looks like:
//....+....1....+....2....+....3....+....4....+....5....+....6....+....7
// 546272.375000 4055547.750000 -267.539100
// 546273.000000 4055557.500000 -286.394500
// 546273.125000 4055535.750000 -283.835900
// 545517.937500 4054927.500000 -311.289100
// 545515.312500 4054904.750000 -306.789100
// 545279.750000 4054955.500000 -294.136700
// 545272.437500 4054535.750000 -324.007800
// 545273.062500 4054541.250000 -328.050800
// 545515.937500 4054534.500000 -346.898400
// 544603.000000 4058306.000000 -296.510200
// 544601.625000 4058304.000000 -309.978900
// 544603.625000 4058308.500000 -270.475000
// 545281.875000 4054686.000000 -304.738300
// 545281.125000 4054697.250000 -323.078100
// 545515.687500 4054713.750000 -320.511700
// 545421.125000 4052345.750000 -289.668000
// 545414.312500 4052343.500000 -287.781200
// 545419.750000 4052345.750000 -291.406200
// 544550.812500 4055694.250000 -310.918000
// 544553.500000 4055694.250000 -311.339800
// 544555.562500 4055693.000000 -313.191400
// 545972.062500 4054448.750000 -341.761700

public class file_reader3 {
  public static void main ( String[] args ) {
    if ( args.length < 1 ) {
      System.err.println ( "Usage: java TokenizerText <src> ");
      return;
    }
    try {
      StreamTokenizer stok = new StreamTokenizer (
        new FileInputStream (args[0]));
```

```
      // Treat a space as a normal character.
      stok.whitespaceChars(',', ',');
      int token;
      // token is filled with a code indicating type of item
      // just read.


 while ( ( token = stok.nextToken() ) != stok.TT_EOF ) {
        switch (token) {
          case stok.TT_NUMBER:
            // If a number is read, the value is placed in
            // the double nval.
                     double xx = stok.nval;
              if ( (xx  > 545760.0 && xx  < 545825.0)) {
                  token  = stok.nextToken();
                               double yy = stok.nval;
                               if (yy > 4052000.0 && yy < 4056000.0) {
                                                    token  = stok.nextToken();
                                                    double nt = stok.nval;
                                   System.out.println ( yy + "," + nt);
                               }

              }
            break;
          case stok.TT_WORD:
            // If a word is read, the value is placed in
            // the string sval.
            System.out.println ( "Word: " + stok.sval );
            break;
          default:
            break;
        }
      }
    } catch ( IOException e ) {
        System.err.println ( e );
        return;
    }
  }
}
```

```
import java.io.*;
import java.math.*;

/**
 * This is an attempt to translate a program written by
 * Charles Connor in True Basic.
 **/
public class MagneticCube {
  public double a1, a2, b1, b2, h1, h2, theta, maginc, magdec, mag0inc, mag0dec, EI;

  /** a1 and a2 are the width (a1,a2) a1=300.0001, and a2=600.5001
   *  b1 and b2 are the length (b1,b2) b1=300.00001, and b2=600.50001
   *  h1 and h2 are the depth (h1,h2) h1=100, and h2=5000
   *  theta is the angle from north, in degrees theta=0
   *  maginc is the inclination of the Earth's field I=45
   *  magdec is the declination of the Earth's field magdec=0
   *  mag0inc is the inclination of the vector of magnetization mag0inc=45
   *  mag0dec is the declination of the vector of magnetization mag0dec=0
   *  EI is the intensity of magnetization EI=500
   **/

  public MagneticCube(double a1, double a2, double b1, double b2,
                      double h1, double h2, double theta, double maginc,
                      double magdec, double mag0inc, double mag0dec, double EI) {
    this.a1 = a1;
    this.a2 = a2;
    this.b1 = b1;
    this.b2 = b2;
    this.h1 = h1;
    this.h2 = h2;
    this.theta = theta;
    this.maginc = maginc;
    this.magdec = magdec;
    this.mag0inc = mag0inc;
    this.mag0dec = mag0dec;
    this.EI = EI;
  }


  // This method calculates the magnetic anomaly and returns...
  // ...the value of the total magnetic field
  public double anomaly2 (double x) {


  // Initialize some more variables
  double p, q, r;

  // Calculate the earth's inclination and declination wrt theta
  p = Math.cos(maginc)*Math.cos(magdec-theta);
  q = Math.cos(maginc)*Math.sin(magdec-theta);
  r = Math.sin(maginc);

  // Initialize some more variables
  double L, M, N;

  // Calculate the vector of magnetization's inclination and...
  //...declination wrt theta
  L = Math.cos(mag0inc)*Math.cos(mag0dec-theta);
  M = Math.cos(mag0inc)*Math.sin(mag0dec-theta);
  N = Math.sin(mag0inc);

  // Initialize some more variables
  double g1, g2, g3, g4, g5;
```

```
  // Calculates ????
  g1 = EI*(M*r + N*q);
  g2 = EI*(L*r + N*p);
  g3 = EI*(L*q + M*p);
  g4 = EI*(N*r - M*q);
  g5 = EI*(N*r - L*p);

  // Initialize more variables
  int kount;
 double y;

  /**
  * Calculates the magnetic anomaly and returns the value
  * of the total magnetic field.
  **/

  //Initialize some variables
  y = 0.0;

  double xp = x*Math.cos(theta) + y*Math.sin(theta);
  double yp = -x*Math.sin(theta) + y*Math.cos(theta);

  double ap1 = a1 - xp;
  double ap2 = a2 - xp;
  double bp1 = b1 - yp;
  double bp2 = b2 - yp;

  double r1 = Math.sqrt((ap1*ap1) + (bp1*bp1) + (h1*h1));
  double r2 = Math.sqrt((ap1*ap1) + (bp1*bp1) + (h2*h2));
  double r3 = Math.sqrt((ap2*ap2) + (bp1*bp1) + (h1*h1));
  double r4 = Math.sqrt((ap2*ap2) + (bp1*bp1) + (h2*h2));
  double r5 = Math.sqrt((ap1*ap1) + (bp2*bp2) + (h1*h1));
  double r6 = Math.sqrt((ap1*ap1) + (bp2*bp2) + (h2*h2));
  double r7 = Math.sqrt((ap2*ap2) + (bp2*bp2) + (h1*h1));
  double r8 = Math.sqrt((ap2*ap2) + (bp2*bp2) + (h2*h2));

  double f1 = (r2 + ap1)*(r3 + ap2)*(r5 + ap1)*(r8 + ap2)/
              ((r1 + ap1)*(r4 + ap2)*(r6 + ap1)*(r7 + ap2));
  double f2 = (r2 + bp1)*(r3 + bp1)*(r5 + bp2)*(r8 + bp2)/
              ((r1 + bp1)*(r4 + bp1)*(r6 + bp2)*(r7 + bp2));
  double f3 = (r2 + h2)*(r3 + h1)*(r5 + h1)*(r8 + h2)/
              ((r1 + h1)*(r4 + h2)*(r6 + h2)*(r7 + h1));
  double f4 = Math.atan(ap2*h2/(r8*bp2))-Math.atan(ap1*h2/(r6*bp2))-
              Math.atan(ap2*h2/(r4*bp1))+Math.atan(ap1*h2/(r2*bp1))-
              Math.atan(ap2*h1/(r7*bp2))+Math.atan(ap1*h1/(r5*bp2))+
              Math.atan(ap2*h1/(r3*bp1))-Math.atan(ap1*h1/(r1*bp1));
  double f5 = Math.atan(bp2*h2/(r8*ap2))-Math.atan(bp2*h2/(r6*ap1))-
              Math.atan(bp1*h2/(r4*ap2))+Math.atan(bp1*h2/(r2*ap1))-
              Math.atan(bp2*h1/(r7*ap2))+Math.atan(bp2*h1/(r5*ap1))+
              Math.atan(bp1*h1/(r3*ap2))-Math.atan(bp1*h1/(r1*ap1));

  /** The Math.log function returns a natural (base e) log.  Convert
     *  natural logs to base 10 logs with code like this:
     *  double nat_log = ...
     *
     *  double base10log = nat_log / Math.log(10.0);
     **/

  double T = g1*Math.log(f1) + g2*Math.log(f2) +
             g3*Math.log(f3) + g4*(f4) +
             g5*(f5);

  return T;
  }
```

```
}
```

Scientific Notebook
20-1402-461
Inititals: CC

Chuck Connor
Oct 4,1999

The following pages contain text written by C. Connor for the National Academy of Sciences review of the USGS volcanic hazard program

1 *2*

Information potentially subject to copyright protection was redacted from this location. The redacted material is from the reference information listed above.

PRIVILEGED DOCUMENT—DO NOT QUOTE OR CITE

Trip Report

Name of Meeting: UNAVCO Volcano Workshop 99

Location: Jackson, Wyoming

Dates: September 15-19

Persons Present: Chuck Connor and Peter La Femina (both at CNWRA), forty participants from various universities and the University Navstar Consortium (UNAVCO)

Purpose

Connor and LaFemina attended the University Navstar Consortium (UNAVCO) volcano geodesy workshop with the goals of becoming familiar with the current state-of-the-art in volcano geodesy, presenting recent CNWRA work on volcano deformation associated with small volume basaltic eruptions at Cerro Negro, and to participate in discussions about the Plate Boundary Observatory (PBO) proposal to the National Science Foundation (NSF). The meeting was an outstanding opportunity to learn about new techniques in geodesy that are applicable to work in the IA and SDS KTIs, and in the long term to monitoring of the proposed Yucca Mountain repository.

Summary of the Meeting

Numerous talks and posters were presented over three days. Highlights of these talks are briefly summarized in the following.

Roger Delinger (USGS) presented an overview of volcano deformation and current thinking about the dynamic links between surface deformation and magma source regions. On active volcanoes, complex deformation is produced in the near-field by fracturing and short period variations in tilt, and seismicity. These complexities were illustrated using deformation and related data sets from Monserrat and Mount St. Helens volcanoes. Delinger modeled thrust faulting at the base of lava domes using sandbox models that show the development of palm tree structures over intrusions. He also discussed progressive dilation inward, toward the intrusion, that mimics the development of imbricate fault sets observed on volcanoes at several scales. This progressively inward dilation was duplicated in a sandbox experiment. Delinger discussed the role of stick-slip along conduit walls in producing unsteady extrusion with constant magma supply. Periodic pressure changes can also explain unsteady extrusion. Barry Voight (Penn. State) has explained the same observations by cyclic degassing, rather than stick-slip. An important point made by Delinger is that elastic models of deformation always underestimate source depth. Delinger also showed data from the Leadville water injection experiment, showing that (1) strain was accommodated by dilation of faults oblique to the major axis of horizontal compression and (2)

the locus of seismicity migrated slowly outward from the inject well over time.

Alan Linde describe the use of borehole strainmeters at Long Valley caldera, California, and elsewhere. Borehole strainmeters amplify strain using hydraulics rather than electronics. Stability of the strainmeter in the borehole is achieved using an expanding grout. This amplification and stability allows measurement of aseismic strain at 10^{-12} (that's 10^{-3} Nanostrain/m). The instrument provides a real-time measurement of crustal strain. Linde believes that these instruments deployed at 500 m depth can detect magma movement at 30-35 km depth. These instruments would be extremely useful at Yucca Mountain for measuring strain due to thermal expansion of the mountain following emplacement of waste packages and could serve as a monitoring method for volcanic and seismic hazards. Ideally, a long baseline would be created before waste was ever placed in Yucca Mountain. The borehole strainmeters cost on the order of $100,000 and often require one year of stabilize.

Several talks were presented that summarize current, standard geodetic measurments at Pavlof, Hengill, and Kilauea volcanoes, and Long Valley, Taal, and Yellowstone calderas. A great deal of attention focused on the distribution of geodetic sites on these volcanoes, and the use of models, such as the Mogi model, to explain deformation in terms of magma depth. In particular, Kurt Feigel (Univeristy of the Globe, France) discussed the relationships between geodetic moment and seismic moment at Hengill volcano, Iceland. He found that the geodetic moment exceeds the seismic moment by at least one order of magnitude at this volcano. This finding is quite relevant to strain partitioning between volcanic, seismic, and aseismic strain in the Basin and Range. Feigel calculated the coulomb stress produced by a Mogi model of the deformation data at Hengill volcano. That map pattern of coulomb stress explained the distribution of earthquakes very well.

There were numerous talks on the use of interferometric satellite aperture radar (InSAR) as a method of monitoring volcano deformation. InSAR has successfully revealed patterns of deformation in calderas, such as Yellowstone, and at numerous volcanoes. These data are currently always modeled using a Mogi source, which partly reflects both the limited development of modeling techniques (i.e., a poor link with the geology of the volcanoes) and partly reflects the character of InSAR data. InSAR only yields information about vertical movements whereas other techniques (e.g., GPS) provide information about lateral and vertical movement. It is also clear that InSAR does not work at many volcanoes because of the poor satellite coverage, steepness of slopes, snow, rapid tree growth and similar issues. Nonetheless, it is clear from these talks the InSAR could provide a valuable perspective of deformation at Yucca Mountain and the surrounding area (some InSAR work has been done previously by Jim Savage, U.S. Geological Survey) if performed on a regular basis. However, the future of InSAR is highly uncertain. The satellite is only planned to be in operation through 2002 or 2003, and there are no plans to replace it.

An important aspect of monitoring deformation at volcanoes is the cost of monitoring technologies. Chuck Meertens (UNAVCO) discussed the development of a low-cost continuous GPS monitoring system. This system combines single-frequency GPS receivers, with differential GPS (DGPS) surveying techniques. The system developed by Meertens uses a Canadian Marconi Communications, Inc. All-Star, 12-channel GPS receiver, Micropulse, Inc. L1 GPS antenna,

Freewave spread spectrum radio/modems, and an antenna ground plane developed and manufactured at UNAVCO. The cost of the system is approximately $3,500. This is a major reduction in GPS geodetic survey cost, considering a geodetic grade Trimble antenna costs $7,000 alone. The Freewave radio/modems are an integral piece of this system, because of their ability for time-delay multiple access (TDMA) transmission and telemetry of GPS data. Essentially, TDMA allows remote GPS receivers to find the best path of transmission for their data back to a master station, which combines all data from the remote stations and then transmits the data back to an observatory where the data is stored. In addition, TDMA allows each remote station to act as a repeater. The L1 system is currently deployed at Long Valley caldera, Kilauea, and Taal volcano, Philippines. The system is also being used to monitor man-made structures, including bridges and dams.

Paul Segall presented an overview of models of volcano deformation data. His presentation of the forward models concentrated on the Mogi model, which is used to model deformation as a pressure point deforming an elastic half-space. Several variations on the Mogi model were presented. Segall demonstrated that Mogi models tend to underestimate the depths of source regions. This is important because InSAR data sets are nearly always modeled using Mogi models and nearly always show shallow sources. Segall also summarized his work using inverse modeling techniques (simulated annealing) to deduce the change in volume of a dike at Izu volcano, Japan. Segall pointed out that seismicity associated with the Izu dike intrusion stopped ten days before the deformation ceased. This lag in deformation is attributed to aseismic inflation of the dike. Segall was able to bound the parameter space (i.e., change in dike thickness, length, volume) using simulated annealing techniques.

Connor presented several examples of volcanic hazards assessments, with emphasis on the potential role of GPS networks in evaluating long-term hazards at nuclear facilities. Two nuclear facilities where long term hazard assessment would benefit from GPS strain measurement networks are the proposed Muria nuclear power plant, Indonesia, and the operating Yerevan power plant, Armenia. CNWRA staff have previously proposed several projects to the NRC international program office and the International Atomic Energy Agency to use these sites to demonstrate the role of geodetic GPS in hazard assessment at nuclear facilities. The proposed Muria power plant lies 20 km from the summit caldera of the Muria volcano complex on Java. Five nested caldera occur in the complex, that extend for over 45 km on a NE trend, also defined by a rift. Muria has not erupted since the beginning of the Holocene, when phreatic pit craters formed on the flanks of the volcano. Sedimentation rates are extremely high on the peninsula. Currently, no GPS network exists at Muria to monitor rates of deformation across this rift. The Yerevan power plant is located at the base of the Holocene Aragatz and Ararat volcanoes and within an active basaltic volcanic field. The area around the power plant has experienced uplift during the last 2 ka, accompanied by phreatic eruptions. Again, no GPS monitoring network is currently in place. As a third example, recent GPS surveys near Yucca Mountain, although controversial, illustrate the potential for introducing GPS strain measurements into quantitative volcanic hazard assessments.

La Femina summarized the recent work by a CNWRA team at Cerro Negro volcano, Nicaragua. A >2 km-long fracture set developed south of Cerro Negro volcano during and following seismic activity on August 4, 1999. The largest earthquakes, magnitude 4.9 and 4.6, preceded by ≈ 12 and 4 hrs, respectively, the onset of the August 5 - 7 small-volume basaltic eruption from new vents on the south flank of Cerro Negro. Our integrated geophysical surveys concentrated on an area ≈ 1 km south of Cerro Negro, where total dilation across the fracture set was up to 1 m over a distance of < 20 m, with little evidence of vertical or strike-slip displacement. Steam flow was visible from this fracture set in the days following the eruption. Surveys included ground magnetic, SP, radon, and temperature monitoring on profiles perpendicular to the fracture set. A linear, normally-magnetized 1200 nT anomaly was identified within the thermal area, which we have modeled as a 1 m wide, steeply dipping tabular body of magnetized rock (1-5 A/m), at a depth of 25-35 m beneath the surface. SP (160 mV), radon (> 60 pCi/l), and temperature (90 °C) anomalies across the fracture set all indicate forced convective flow of water vapor and gases through the fractures. One explanation for our observations is that lateral dike injection occurred from beneath Cerro Negro toward the south, following seismic activity along a preexisting fault. This interpretation is also consistent with the geology of Cerro Negro, where N-S to NW-SE trending structures are common, including the N-S trending Cerro La Mula cinder cones and phreatic pit craters north of Cerro Negro, and the 1968 Cristo Rey vent and fracture sets formed in 1955 in Cerro Las Pilas volcano, south of Cerro Negro. In 1994-1995, we identified soil radon anomalies on the south side of Cerro Negro, in the location of the new vents and the new fracture set, suggesting a permeable structure (fault / fracture) was already present at that time. Based on this model, the 1999 eruption and the pre-eruption seismicity were responses to crustal extension across the volcano alignment. In this case, the resulting strain was partitioned between seismic deformation and magma intrusion.

Policy statements

The UNAVCO workshop resulted in a series of position statements intended to reflect the consensus of the geodetic volcanology community. These statements, primarily addressed to funding agencies (NSF and NASA) are:

- Increase support for volcano monitoring and collaboration

-List high-risk and high-potential volcanoes; establish baseline geodetic data and crisis response plans at these volcanoes

- Implement real-time deformation monitoring and develop data accessibility

- Develop low-cost GPS systems

- Develop a USGS Volcanic Hazards Program geodetic data base

-Develop and distribute user friendly modeling software

-Encourage InSAR

Many of these themes are incorporated into a pre-proposal for the NSF Plate Boundary Observatory. This is an Earth Sciences project to be proposed as a NSF major research initiative, with the goal of securing approximately $100 million for instrument and facilities development. A fraction of this MRI would be devoted to volcanology. Tim Dixon (University of Miami), C. Connor, and S. McNutt (Alaska Volcano Observatory) wrote this pre-proposal at the workshop (attached). Tim Dixon will be presenting this pre-proposal (after revision) at an NSF MRI meeting in October.

Recommendations

Serious attention should be paid to the potential role of borehole strainmeters for monitoring Yucca Mountain. Data generated by these instruments would be extremely useful for monitoring deformation associated with normal performance of the repository. Such deformation would be produced by, for example, volumetric expansion associated with heating of the repository. These instruments would also provide exceptional insight into tectonic processes operating at Yucca Mountain, and may be very useful for improving volcanic and seismic hazard assessments (Connor et al., 1998). Alan Linde was very careful to point out that borehole strainmeters will provide the best results if a long baseline is developed before the onset of deformation. Therefore, borehole strainmeters should be installed at least several years prior to construction of the repository.

Excellent opportunities exist for integrating some of the strain partitioning work presented at the conference into CNWRA models of seismicity and deformation in the Yucca Mountain region. For example, many talks presented calculations of the geodetic moment (analogous to the seismic moment) and directly calculated coulomb stress from geodetic models. The calculated coulomb stress was then compared to seismic maps. CNWRA staff should consider folding these types of analyses directly into tectonic models and hazard assessments we provide the NRC.

CNWRA staff are in a good position to develop an integrated L1 GPS system, taking data from the receiver to a processed data set on the worldwide web. This seamless integration of technologies does not yet exist. If we were able to achieve such integration, UNAVCO and other organizations would be interested. It is recommended that CNWRA staff consider an internal research proposal to develop this system for use on NRC and related projects.

Action Items: None

Problems Encountered: None

References

Connor, C.B., J.A. Stamatakos, D.A. Ferrill, and B.E. Hill. 1998, Detecting strain in the Yucca Mountain area, Nevada. Science, 282:1007b.

Entries into SN #115E for the period October 14, 1999 to 8/18/00 have been made by C. Connor. [signature] B Conn 8/18/00

No original data have been removed
cc 8/18/00

SCIENTIFIC NOTEBOOK 115E

14 8/17

CHUCK CONNOR

PROJECT

IGNEOUS ACTIVITY
20-1402-461

SCIENTIFIC NOTEBOOK 115E

CHUCK CONNOR

PROJECT

IGNEOUS ACTIVITY
20-1402-461

Feb - July 2000

PARTS IN THIS NOTEBOOK:

Ash plume code development, testing , and summary
Summary of beowulf cluster development
Summary of development of parallel codes
Seismic hazard code development
Plans for physical analog models

Scientific Notebook 115E
20-1402-461
Inititals: CC

Chuck Connor
April 1, 2000

The following work was accomplished to support the airborne transport of radionuclides ISI.

Introduction

Planners and the communities they serve need to know what to expect from volcanic eruptions. Volcanologists can help planners develop a reasonable perspective of volcanic eruptions by providing quantitative information about volcanic hazards. The goal of this webpage, the accompanying articles, and java applets is to provide an overview of the tools used by CNWRA staff for estimating hazards associated with the dispersion and deposition of volcanic ash.

We emphasize that modeling volcanic phenomena, like deposition of volcanic ash, is complex. Experience in volcanology, fluid dynamics, and probabilistic hazard assessment is required. This webpage is not intended to act as a substitute for such experience. Rather, it is intended to provide a heuristic overview of the hazard assessment process. Users of this webpage are encouraged to experiment with the java applets in order to gain insight into how such probabilistic hazard assessments are done.

Organization of the Site

See the volcanic ash article for a detailed review of CNWRA methods in volcanic ash hazards assessment, including the mathematical basis and simplifying assumptions for these models.

Four java applets are provided to calculate volcanic ash accumulation, with the following functions:

Ash Point : Estimates the mass of ash expected to accumulate at a point (in gm/ sq. cm) for given eruption and meteorological conditions. This is by far the fastest applet to execute. It is a good idea to experiment with parameter variation using this applet before going to more involved calculations in other applets. See Ash Point Instructions for more details about running this applet.

Ash Dispersion : Estimates the ash accumulation along the major axis of dispersion of the volcanic eruption. This applet repeats the calculations done in Ash Point for numerous points along the ash plume axis in the downwind direction. See Ash Dispersion Instructions for more details about running this applet.

Chuck Connor
April 1, 2000

Ash Contour: Calculates an isopach map of ash accumulation (in gm/ sq. cm) for an area about the volcano. Such calculations can be extremely computationally intensive (take a long time on your PC). See Ash Contour Instructions for more details about running this applet.

Probability Graph: This applet produces a hazard curve for ash accumulation. Expected ash accumulation at a point is calculated using stochastic (random) sampling of a range of parameters. The results are plotted on a graph that shows the probability that ash accumulation at a point will exceed a given value, given the range of eruption characteristics and meteorological conditions specified in the parameter input fields. Note that all of the parameter distributions are assumed to be uniform random between the values specified. See Probability Instructions for more details about running this applet.

Each of these applets requires a number of input parameters. These are summarized in the Parameter Table.
The following constants are used:

CONSTANT
DESCRIPTION
VALUE
UNITS

gravity - gravity 980.0 - cm / s2
nua - dynamic viscosity of air 0.00018 - gm /cm -s
phiair - density of air 0.001293 - gm cc
c - eddy diffusivity in the atmosphere give as in cgs 400.0 - cm/s(5/2)

The following input variables are used:

VARIABLE
DESCRIPTION
UNITS

total_ash_mass - the total amount of material erupted -gm
ymin - the minimum particle size of tephra erupted - phi units (e.g., -5.0)
ymax - the maximum particle size of tephra erupted - phi units (e.g., 5.0)
dmean - the mean particle size of tephra erupted- phi units (e.g., -1.0)
dsigma - the standard deviation particle size of tephra erupted - phi units (e.g., 1.0)
phiash - the clast density of individual pyroclasts assumed to be independent of gransize - gm/cm3
dfshape - the particle shape factor, assumed to be independent of gransize - dimendionless
debeta - the empirical factor controlling plume geometry - dimendionless

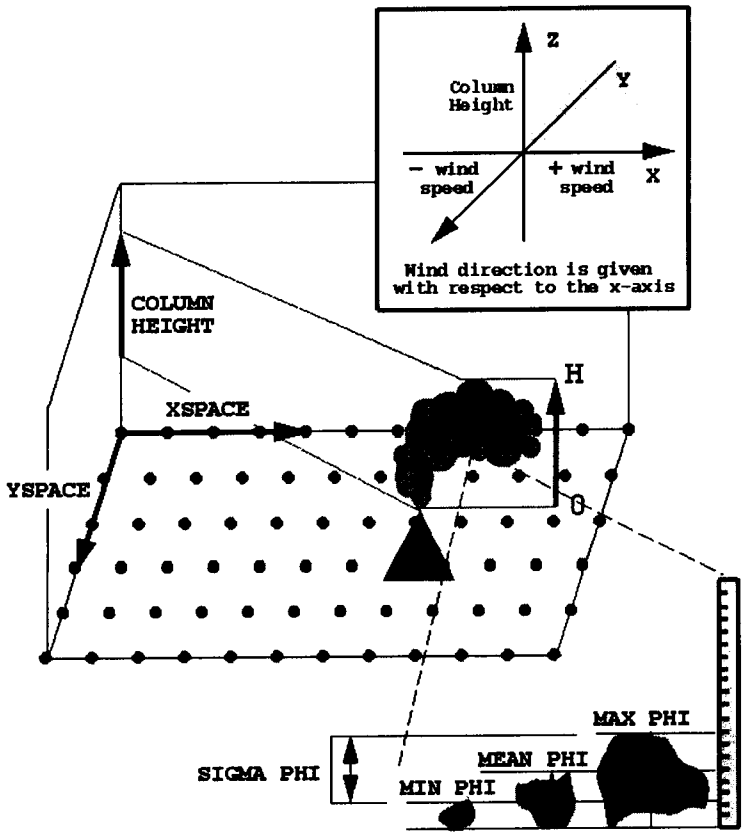
Chuck Connor
April 1, 2000

w0 - the eruption velocity - cm/s
h - the eruption column height - km
u - the average wind speed - cm/s
udir -the wind direction wrt the x-axis - +/- radians

A Graphic also diagrams some parameters.

Ash Point

The Ash Mass at a point is calculated by the Ash_Point applet. The webpage produces a numerical output.



Finally, overviews of CNWRA developments are provided for:

Server-side parallel computations using a Beowulf Cluster
Variable wind-field calculations
Useful scripts and code details

Contact Us for more information on this and related topics.

Estimation of Volcanic Hazards Related to Tephra Fallout

by

Charles B. Connor, Brittain E. Hill, Brandi Winfrey,
Nathan M. Franklin and Peter C. La Femina

Center For Nuclear Waste Regulatory Analyses

Southwest Research Institute

6220 Culebra Road, San Antonio, TX 78235-5166 USA

March 8, 2000

Abstract

The goal of probabilistic volcanic hazard assessment is to translate complex volcanological data and numerical models into practical hazard estimates for communities potentially affected by volcanic eruptions. Probabilistic volcanic hazard assessment quantifies volcanic hazards and illustrates uncertainties about the magnitude and consequences of volcanic activity. Planning based on probabilistic volcanic hazard assessment has the potential of mitigating the effects of volcanic eruptions when they occur. We develop an approach to estimate volcanic hazards related to tephra fallout, and illustrate this approach with a tephra fallout hazard assessment for the city of León, Nicaragua and the surrounding area. Tephra fallout from eruptions of Cerro Negro volcano has caused damage to property, adverse health effects, and disrupted life in this area. By summarizing the geologic and historical records of past eruptions of Cerro Negro on a probability tree, we show that the inhabitants of León can expect > 1 cm of tephra accumulation from approximately 30% of eruptions, and > 4cm tephra accumulation from approximately 9% of eruptions of Cerro Negro volcano. This historical record is augmented with simulations of tephra dispersion which estimate the likelihood of tephra accumulation given a range of eruption magnitudes and map the expected distribution of tephra over a broader region. A screening

distance value 0.5m is calculated using the tephra dispersion model. Without a fundamental change in the eruptive behavior of Cerro Negro, tephra accumulation in León is not expected to exceed this value.

Introduction

Planners and the communities they serve need to know what to expect from volcanic eruptions. Volcanologists can provide the information required to develop strategies for volcanic hazard mitigation and to weigh the relative costs of mitigation efforts. Information about volcanic hazards is best presented in the form of probabilistic hazard assessments, combining elements of the geologic record, numerical simulation of volcanic eruptions, and an accurate picture of the natural uncertainty in estimates of eruption magnitude and timing. Here, we outline steps for a probabilistic hazard assessment of one aspect of volcanic eruptions: tephra fallout. Other phenomena that impact communities near volcanoes, such as pyroclastic flows and lahars, may be treated in a similar fashion, but require separate hazard analyses.

Tephra fallout results from explosive volcanic activity. Tephra, colloquially referred to as volcanic ash, consists of pyroclasts produced during volcanic eruptions and accidental lithic fragments incorporated in the eruption. Tephra fallout results when tephra is carried aloft in a volcanic eruption column and subsequently deposited by sedimentation out of the volcanic plume, sometimes at great distances from the volcano (Figure 1) (Fisher and Schmincke, 1984; Sparks et al., 1997).

Large eruptions (VEI 4-6) (Newhall and Self, 1982; Simkin and Siebert, 1994), such as the May 18, 1980, eruption of Mount St. Helens, and the June 15, 1991, eruption of Mount Pinatubo, produce significant tephra accumulation at great distances from erupting volcanoes, resulting in major disruptions for society (e.g., Sarna-Wojcicki et al., 1981; Punongbayan et al., 1996; Koyaguchi, 1996). Even moderately-sized volcanic eruptions (VEI 3) can significantly affect areas >10km from the volcano due to tephra fallout (e.g., Hill et al., 1998). Tephra fallout causes building collapse, disruption of power and water supplies, damage to mechanical systems such as vehicle engines, and widespread damage to agricultural products, including livestock. Although the consequences of tephra fallout may be less severe than other eruption phenomena, vulnerability is often much greater due to the wide dispersal of tephra in volcanic plumes.

In some cases, the continual remobilization of tephra deposits by wind or surface-disturbing activities has resulted in respiratory ailments and related deleterious health effects (Baxter, 2000). In recent years, attention has also focused on the harm to aircraft caused by tephra, and the risks associated with such damage for airline passengers and cargo (Miller and Casadevall, 2000). Figure 1: Eruptions of Cerro Negro volcano, Nicaragua. The 1968 eruption (left), from USGS files; the 1995 eruption (top right); and damage in Leon resulting from the 1992 eruption (bottom right). Our purpose is to describe probabilistic methods for assessment of tephra fallout hazards. We focus on tephra accumulation within tens of kilometers of the volcano,

where tephra accumulation is most likely to cause damage. These methods rely on the recognition that tephra fallout is best treated as a conditional probability, derived independently of the current state of volcanic activity. Thus, hazard assessments can be prepared in advance of episodes of volcano unrest and mitigation strategies can be developed before volcanic crises occur. In this sense, long-term planning for tephra fallout is a practical goal. To achieve this goal, it is necessary to combine geologic observations of past patterns of tephra accumulation with results of numerical simulations of tephra fallout. These results must be presented in a manner that reflects both the natural variability in the physical processes resulting in tephra fallout, and the uncertainty related to our models and observations of these phenomena. Steps in tephra fallout hazard estimation are described in the following, with special reference to Cerro Negro volcano, Nicaragua, a small-volume basaltic cinder cone that experiences comparatively frequent eruptions accompanied by tephra fallout (Figures 2 and 3), and where we have implemented these steps previously (Hill et al., 1998).

Treating Tephra Fallout as a Conditional Probability

Conditional probabilities enable volcanologists to consider a complicated series of events that typically comprise a volcanic eruption discretely. In the case of tephra fallout, it is simpler to estimate expected tephra accumulation, given eruption conditions, than to combine this disparate information into a single hazard forecast. For example, the geologic record or numerical simulations can be used to infer P[tephra accumulation >1cm at (x,y) | VEI 2], the probability that tephra accumulation will exceed 1cm at a given location, (x,y) , given that a volcanic eruption of intensity VEI 2 has taken place. The long-term probability of an eruption occurring at all, or the probability that an eruption will follow a period of unrest, are treated as separate issues.

Figure 1 is photo showing this area (not included. CC 7/29/00

Chuck Connor
April 1, 2000

Figure 2: Isopach map for the 1995 eruption of Cerro Negro. Tephra thickness shown in cm.

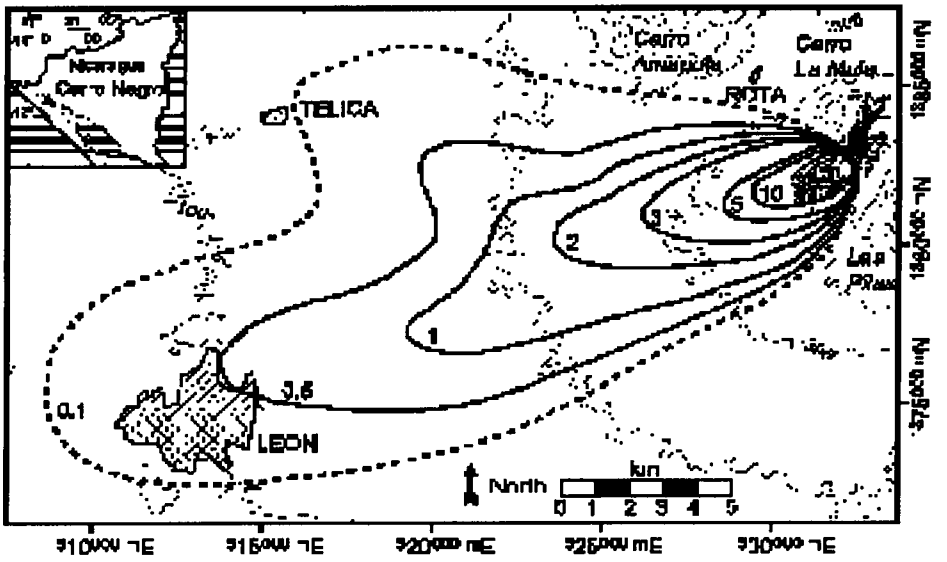
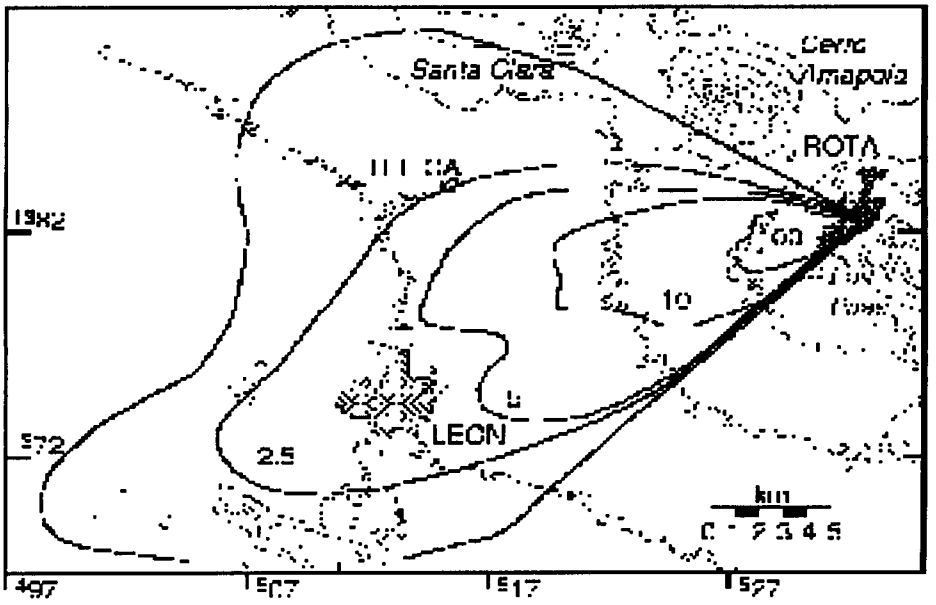
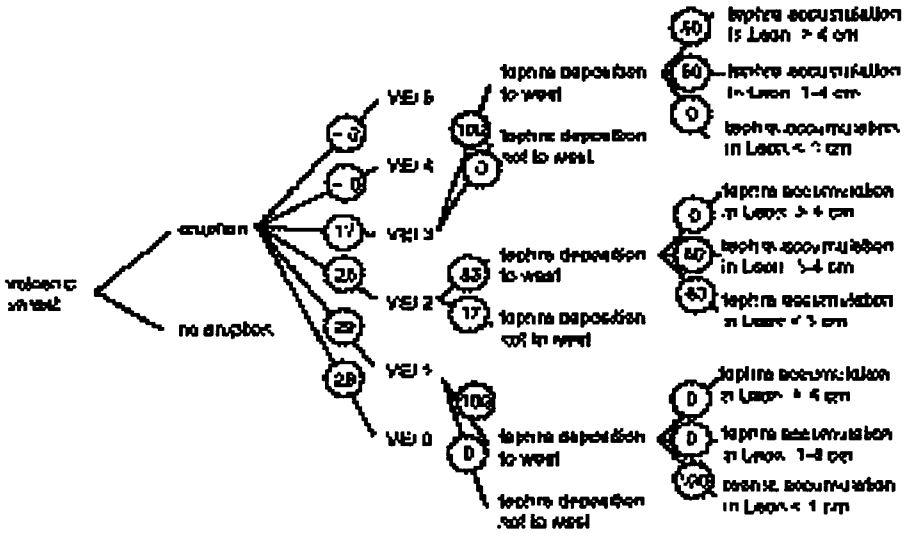


Figure 3: Isopach map for the 1992 eruption of Cerro Negro. Tephra thickness shown in cm.



Chuck Connor
April 1, 2000

Figure 4: Probability tree for tephra accumulation in Lean. Probabilities (in percent) based on the historical and geologic records of past Cerro Negro erruptions.



Application of conditional probabilities is important for two reasons. First, volcanoes often experience prolonged periods of unrest, during which time the probability of an eruption changes considerably. Hazard assessments of tephra fallout for a given eruption made before the volcano unrest began should not require significant modifications during episodes of unrest, when attention is focused on monitoring. Second, the geologic record often provides information about the magnitude of eruptions, for instance the thickness of tephra accumulated during past eruptions at a particular location, but comparatively little information about the timing and frequency of eruptions. Conditional probabilities are easily visualized using probability trees, which illustrate the observers ability to predict outcomes (Schafer, 1996). Often in volcanology, our ability to assign conditional probabilities to all possible outcomes is limited by lack of experience or lack of relevant information. In these circumstances, the trees may be only partially probabilized. We have constructed a probability tree for tephra fallout from Cerro Negro volcano (Figure 4) as an example of their application in volcanology. Cerro Negro has erupted 23 times since the volcano first formed in 1850 (McKnight, 1995; McKnight and Williams, 1997; Hill et al., 1998). Many of the early eruptions of Cerro Negro are poorly documented, but there is a reasonably complete record of tephra fall volumes since 1900. Since 1968, four eruptions have occurred at Cerro Negro (Table 1) that produced tephra fall volumes > 1 X 106m3, and three smaller eruptions have occurred. The most recent of these small eruptions was in August, 1999 (Hill et al., 1999; La Femina et al., 1999). Based on a steady-state model of cumulative volume of material erupted from Cerro Negro, a volcanic eruption is expected

before 2005 with 95% confidence (Hill et al., 1998; 1999). Many, but not all, of the eruptions of Cerro Negro have resulted in tephra deposition in the city of León, the second largest city in Nicaragua (population > 200,000), located 20km from Cerro Negro. In addition, an estimated 100,000 people live in the area surrounding Leon, with active agricultural communities located as close as 1 km from Cerro Negro. Tephra deposition in León has varied from trace amounts to 4cm in 1992 (Figure 3)(Connor et al., 1993). In the 1992 eruption, at least 2 people were killed and 146 injured through building collapse, over 12,000 people were evacuated, and $19M of crops and infrastructure destroyed (Organización Panamericana de la Salud, 2000). Rates of respiratory and intestinal diseases increased by factors of 4 to 6 in the month following the 1992 eruption (Malilay et al., 1997). The significantly smaller 1995 (Figure 2) eruption still displaced 1,200 people and destroyed $0.7M of crops and infrastructure (Organización Panamericana de la Salud, 2000). Future tephra eruptions from Cerro Negro clearly present a significant hazard to people in this area.

Conditional probabilities are assigned to the branches of the probability tree (Figure 4) based on this record of past volcanic activity (Table 1). The probability tree is incomplete because there is no probability assigned to the transition from volcanic unrest to volcanic eruption. There is simply inadequate data available to make this assessment at Cerro Negro. Given a volcanic eruption, there is a reasonably equal probability of VEI 0-2 and a slightly lower probability of VEI 3 eruptions. No eruptions of VEI > 3 have occurred at Cerro Negro. Nearly all of the explosive eruptions that have occurred (VEI 1-3), have resulted in deposition of tephra west of the volcano, toward the city of León. The tree is branched further into tephra accumulation < 1cm , 1-4cm , and > 4cm . This bifurcation reflects both the way past tephra accumulation has been reported in León (McKnight, 1995; Hill et al., 1998) and the severity of the consequences of the tephra accumulation.

Based on the geologic and historical records of volcanic eruptions at Cerro Negro, P[tephra accumulation > 4cm volcanic eruption] = 8.5% and P[tephra accumulation > 1cm volcanic eruption] = 29% (Figure 4). So, although the record is limited (a ubiquitous circumstance in volcanology), the probability tree provides a concise summary of the nature of past volcanic activity and its consequences for the city of León.

Modeling Tephra Fallout Numerically

The goal of modeling tephra fallout using numerical simulations is to provide a path from the geological and historical records of tephra accumulation to estimation of parameter distributions, and ultimately to probabilistic estimates of tephra fallout hazard. Why are these additional steps necessary? Numerical simulation of geologic phenomena is a complex undertaking and modeling tephra fallout is certainly no exception. It would be ideal if the geologic and historical records provided an adequate picture of the frequency distribution of tephra fallout, and if, as a consequence, we could have a great deal of confidence in hazard estimates solely based on this record. Unfortunately, this is not the case. The geologic and historical records are insufficient as

the sole basis for hazard assessment for several important reasons.

First, the geologic record is invariably sparse. In applying probabilistic techniques, we assume that there are parameters, such as mean and standard deviation, that describe the frequency distribution of tephra accumulation in a given area. Given a large number of events, we would expect the values of these parameters to converge toward a fixed limit. But with few events (e.g., < 20 well-documented eruptions of Cerro Negro), there is no reason to believe with confidence that the frequency distribution of outcomes (e.g., thickness of tephra in León) is well represented by the geologic record, or that the limiting values of the parameters that characterize the frequency distribution are known (e.g., von Mises, 1953). Probabilistic hazard assessments based on poorly known frequency distributions are highly uncertain.

Second, the geologic record is biased toward large events because these are more likely to be preserved. Smaller, generally more frequent, eruptions may leave no discernable; geologic record. Conversely, the historical record may also be biased. Because of its brevity, the historical record often poorly reflects the full range of activity a given volcano has experienced. At Cerro Negro, given its short history, it is unlikely that the full range of potential eruptions is captured in the range of eruptions that have already occurred.

The disparity between historical and geologic records is particularly evident at Colima volcano, Mexico, where there is nearly complete disjunction between the historical and the stratigraphic records. Dozens of explosive eruptions have been recorded in the last 400 years, but only a single scoria layer has been preserved in the geological record for that period (Luhr and Navarro, 1999). It is clear that at Colima volcano the preserved sequence includes just a fraction of the eruptive events, biasing the geologic record toward large eruptions. Nevertheless, the style of eruptive activity may have also changed over time. Modeling these data and estimating parameter distributions that best fit the data helps resolve this disparity. For example, if complex, multi-modal parameter distributions are needed to model the historical and geologic data sets together, nonstationary behavior in eruption style may be indicated. Alternatively, if comparatively simple parameter distributions model both historical and geologic data sets, then the brevity of the historical record may account for the disparity. This disjunction between geologic and historical data sets is a classic situation and numerical modeling provides insight into its origin and implications for probabilistic volcanic hazard assessment. Third, hazard estimates based on the geologic record are difficult to update, as more information about changing conditions becomes available. Although not the focus of this paper, if the goal is to update probabilistic volcanic hazard forecasts in near-real-time, a model of tephra fallout is required (Carey, 1996).

General Model Description

Therefore the overall benefit of numerical simulation lies in the promise of improved ability to quantify the expected variability in the volcanic system, and consequently better estimate hazard. Development of tephra fallout models have been recently reviewed by Carey (1996), Sparks et al.

(1997), and Rosi (1998). Here we illustrate the integration of tephra fallout models into a probabilistic volcanic hazard assessment using the model developed by Suzuki (1983), and subsequently modified and applied to volcanic eruptions by Armienti et al. (1988), Glaze and Self (1991), Jarzemba (1997), and Hill et al. (1998).

Suzuki's model is empirical, the erupting column is treated as a line-source reaching some maximum height governed by the energy and mass flow of the eruption. A linear decrease in the upward velocity of particles is assumed, resulting in segregation of tephra particles in the ascending column by settling velocity, which is a function of grain size, shape, and density. Tephra particles are removed from the column based on their settling velocity, the decreasing upward velocity of the column as a function of height, and a probability density function that attempts to capture some of the natural variation in the parameters governing particle diffusion out of the column. Dispersion of the tephra that diffuses out of the column is modeled assuming a uniform wind field and is governed by the diffusion-advection equation with vertical settling. Thus, this model relies on estimation of numerous parameters that describe the volcanic eruption and the atmosphere it penetrates. Although not as comprehensive in addressing the physics of the eruption column (c.f., Woods, 1988; 1995), the computational ease of Suzuki's (1983) approach makes it a worthwhile method for hazard assessment, especially in light of the practical difficulties inherent in characterizing the variability in eruption and meteorological parameters (e.g., GVN, 1999).

This model abstracts the thermo-fluid-dynamics of ash dispersion in the atmosphere using the following expression:

$$X(x,y) = \int_{\phi_{min}}^{\phi_{max}} \int_{0}^{H} \frac{5 f_z(z) f_\Phi(\phi) Q}{8\pi C'(t+t_s)^{5/2}} \exp\left[-\frac{5((x-ut)^2+y^2)}{8C'(t-t_s)^{5/2}}\right] dz\,d\phi \qquad (1)$$

where: X is the mass of tephra accumulated at geographic location (x,y) relative to the position of the volcanic vent (x increases in the downwind direction and y is orthogonal to this direction); fz(z) is a probability density function for diffusion of ash out of the eruption column, treated as a line-source extending vertically (z) from the vent to total column height, H; f() is a probability density function for grain size, ; Q is the total mass of material erupted; u is wind speed in the x-direction; t is the tephra particle fall time, and ts is the tephra diffusion time; and C is eddy diffusivity in the atmosphere.

In our implementation, we assume that tephra particle diameter is distributed normally in phi units ( ) (e.g., a grain diameter of 1 = -log2(0.5mm)):

$$f_\Phi(\phi) = \frac{1}{\sqrt{2\pi}\sigma} exp[-\frac{\phi - \mu^2}{2\sigma^2}] \qquad (2)$$

where u and sigma are the mean and standard deviation of particle diameter (Fisher and Schmincke, 1984). The tephra particle fall-time in the atmosphere is given by:

$$t = 0.752 \times 10^6 [1 \quad exp(\quad \frac{0.00625z}{v_0(\phi)})]^{0.928} \qquad (3)$$

where z is height and v0 is the settling velocity, which depends on particle diameter: v0() =

$$\frac{\rho_{ash} g \phi^2}{18\eta p_f^{-0.82} + \sqrt{81\eta^2 p_f^{-0.64} + 1.5\rho_{ash}\rho_{air} g\phi^3 \sqrt{1.07 - p_f}}} \qquad (4)$$

where rhoash is the density of tephra particles, here taken to be independent of grain size; g is gravitational acceleration; eta and rhoair are air viscosity and density, respectively; and pf is a particle shape factor:

$$p_f = \frac{p_b + p_c}{2p_a} \qquad (5)$$

where subscripts a, b, and c refer to the diameter of the particle along its principle axes and pa > pb > pc. The complexity of the expression for settling velocity stems from the complex relationship between drag on the particle and the particle Reynolds number:

$$Re = \frac{v_0(\phi)\phi}{v_{air}} \qquad (6)$$

where vair is the kinematic viscosity of air. At low Re , the settling velocity of particles obeys Stokes law. At higher Re, however, there is a transition to inertial flow and finally turbulent boundary flow (Bonadonna et al., 1998). Equation 4 is only one approximation to these conditions (Suzuki, 1983; Armienti et al., 1988; Sparks et al., 1997; Bonadonna et al., 1998).

In addition to settling, tephra particles diffuse in the atmosphere as they advect downwind. The diffusion time in equation 1 is given by Suzuki (1983) as:

$$t_s = \left[\frac{5z^2}{288C}\right]^{5/2} \qquad (7)$$

derived from the assumption that the standard deviation of the tephra plume width is one-third the height (c.f., Woods, 1988).

The probability density function fZ(z) describes the diffusion of tephra out of the erupting column and into the surrounding atmosphere where it is available for downwind transport:

$$f_z(z) = \frac{\beta w_0 Y(z) e^{-Y(z)}}{v_0(\phi) H (1 - (1 + Y_0) e^{-Y_0})} \qquad (8)$$

and:

$$Y(z) = \frac{\beta w(z)}{v_0(\phi)} \qquad (9)$$

$$Y_0 = \frac{\beta w_0}{v_0(\phi)} \qquad (10)$$

$$w(z) = w_0\left[1 - \frac{z}{H}\right]^{\lambda} \qquad (11)$$

where w0 is the initial eruption velocity at the vent. The parameter controls the shape of function fZ(z). Larger values of result in a greater proportion of tephra reaching high in the eruption column. The upward velocity of particles w(z) is assumed to decrease with height as a function of the parameter , wherefor most applications = 1.
Equations 9 - 11 differ slightly from those provided in Suzuki (1983) and are written to conserve mass in the eruption column. In practice, eruption column height, H, total eruption volume, V, and eruption duration, T, are the best known parameters for a given eruption. Walker et al. (1984) applied:

$$\frac{dV}{dt} = \left[\frac{H}{1.67}\right]^4 \qquad (12)$$

to estimate the mass eruption rate, where H is in kilometers and dV/dt is in m3 (dense rock equivalent). This application assumes that the eruption rate is constant over the duration of the eruption, i.e.,

$$V = \frac{dV}{dt}T \qquad (13)$$

These relations provide a check on input parameters used in the Suzuki (1983) model.

Tephra Fallout Hazard Curves for Cerro Negro Volcano

In practice, the model is calculated numerous times using a range of input parameters (e.g., total mass, column height, etc.) that reflect the historical range of activity at Cerro Negro. Each realization represents a possible combination of eruption style, magnitude, duration, and atmospheric conditions that control the amount and distribution of tephra fallout. Using this stochastic approach, the range of likely eruption styles and their consequences for tephra fallout hazard are evaluated.

Successful application of the model relies on well-chosen input parameters. Our experience is that model results are strongly controlled by eruption volume, column height, and eruption velocity. Eruption volumes were sampled from a log-uniform random distribution, with tephra volumes ranging from 5 × 105 to 1 × 108 m3 , based on estimates of the volumes of past eruptions. Column height was sampled from a uniform random distribution, U[2km , 8km]. The relationships between column height, mass flow, eruption duration, and total eruption volume (e.g., equations 12 - 13) were used to check these input parameters.

Eruptions with very long durations, >120 days, were eliminated from the sample set because long duration explosive activity at Cerro Negro has never occurred. Eruption velocity was sampled from a uniform random distribution U[50m/s , 100m/s], independent of mass flow. Wilson and Head (1981) suggested a relationship between eruption velocity and mass flow that, for Cerro Negro, yields eruption velocities much lower than the 75m/s to 100m/s \) velocities that have been observed (Connor et al., 1993; Hill et al., 1998) or the eruption velocities predicted for buoyant tephra columns at low mass flow rates, 1 × 6 kg/s (Woods and Bursik, 1991).

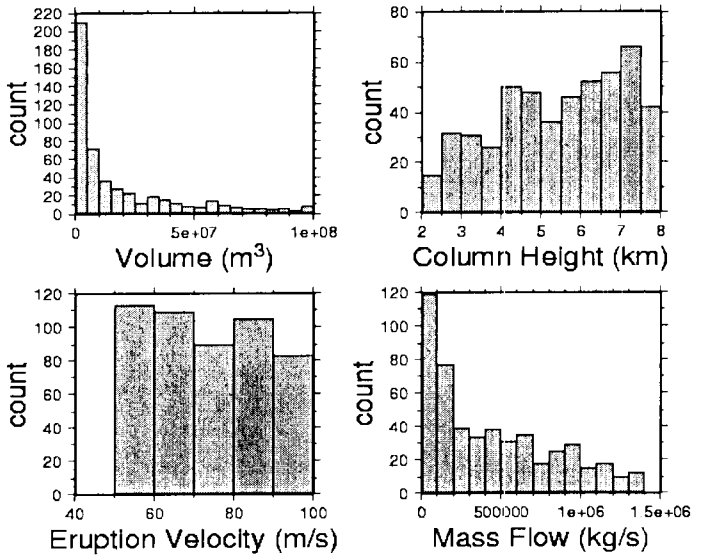
Wind speed and direction also have a strong influence on the tephra accumulation. At Cerro Negro, tradewinds are quite consistent. Average monthly wind direction is to the west or west southwest, and average wind speed on the ground is 4 to 6m/s , with maximum wind speeds of 15m/s measured during most months (National Climatic Data Center, 2000). Furthermore, with the exception of the 1914 eruption (McKnight, 1995), all significant tephra deposition has occurred west of Cerro Negro, with León lying near the major axis of dispersion for the 1968, 1971, 1992, and 1995 eruptions (Figures 2 and 3). Therefore wind speed was sampled from a U[5m/s , 15m/s] distribution and wind direction from a U[-5o , -35o] distribution, where wind direction is with respect to due west.

Grain size distribution was integrated from -5 to 5 , with mean grain size = -1 and standard deviation = 1 . Clast density was varied as U[0.9 , 1.2gm/cm3], equant grains were used (pf = 0.5), and = 0.5 . This value for forces a higher proportion of erupted pyroclasts to elevations close to the top of the eruption column, a choice consistent with field observations and thermo-fluid-dynamic models of volcanic plumes (Woods, 1995).

Stochastically sampled distributions for volume, column height, and eruption velocity are shown in Figure 5. Estimated mass flow given column height (equation 12) is also shown. For these parameter distributions, simulated tephra accumulation in León varies from <<0.1cm to 30cm in

the 500 realizations.

Figure 5: Histograms for several input parameters for the Suzuki model.



The results of the analysis are best illustrated using an exceedence probability plot, also known as a hazard curve. For this range of input parameters, 50% of eruptions result in tephra accumulation <0.2cm in León (Figure 6). Approximately 26% of eruptions result in tephra accumulation >1cm, and 11% of eruptions result in tephra accumulation >4cm, in reasonable agreement with the historical record. The numerical simulation suggests that P[tephra >10cm in León | volcanic eruption] = 5%. Probability of tephra accumulation >1cm and >4cm is contoured for the region about Cerro Negro volcano in Figures 7a and 7b. These maps illustrate the expected outcomes of eruptions at Cerro Negro for the population living closer to the volcano than León.

Figure 6: A hazard curve for tephra accumulation in Leon.

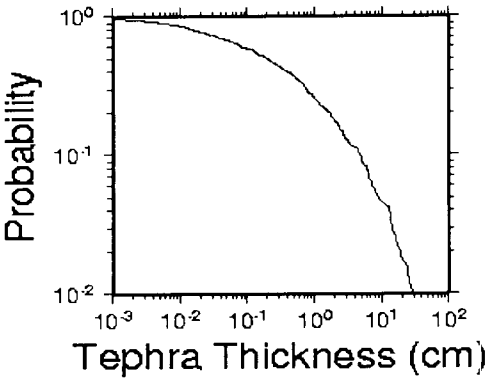
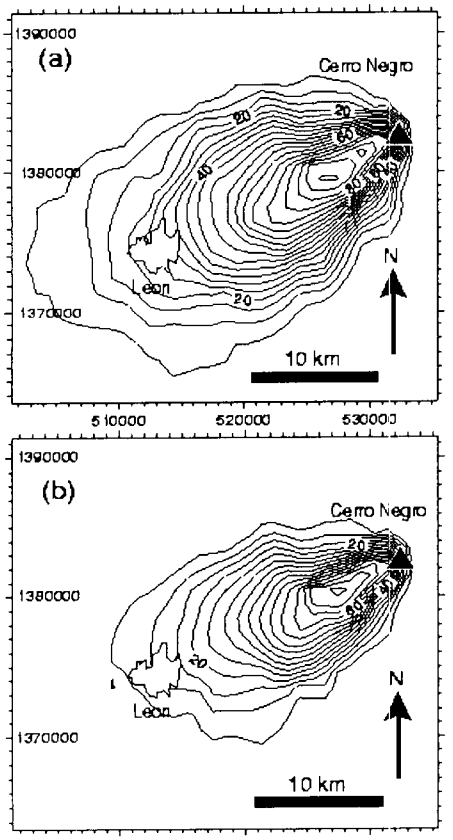


Figure 7: Probability maps for tephra accumulation from an eruption of Cerro Negro. P[tephra > 1 cm | volcanic eruption] (a) and P[tephra > 4 cm | volcanic eruption] (b) are contoured in percent.



Scientific Notebook 115E
20-1402-461
Inititals: CC

Chuck Connor
April 1, 2000

Extreme Events at Cerro Negro

In addition to the probable outcomes summarized in Figures 6 - 7, it is useful to produce worst-case scenarios of volcanic activity in order to communicate the potential magnitude of hazards. Worst case scenarios bound the consequences of volcanic activity, with the goal of conveying the limits of potential volcanic devastation based on reasonable or conservative assumptions. Volcanologists often hesitate to convey worst case scenarios because of the fear, often well-founded, that worst-case scenarios will be misinterpreted as "base case" or "expected" scenarios. Conversely, worst-case scenarios can be ridiculed as overly conservative or alarmist. Nevertheless, it is worthwhile for volcanologists and public officials to think freely about large magnitude events and their impact on risks to public health and safety.

Development of worst-case scenarios can be made palatable through the introduction of the concept of screening distance values (SDVs). Essentially, SDVs are area-based deterministic assessments of hazard using conservative assumptions. The concept of SDVs was developed in seismic risk assessment for sensitive facilities, such as nuclear power plants, that must be located in areas of very low geologic risk. These techniques have now been extended to volcanic hazards for nuclear facilities (International Atomic Energy Agency, 1997) and are readily adapted to general hazard mitigation efforts.

Accurate development of SDVs relies on basic geologic investigations at many volcanoes, not necessarily those most likely to erupt. For example, the recent (< 5,000 yr b.p.) history of Crater Lake suggests that eruptions in this magmatic system are highly unlikely in the coming decades. Nevertheless, the history of Crater Lake and ancestral Mt. Mazama provides a benchmark for the potential magnitudes of future silicic volcanic eruptions in the Cascade mountains. In constructing hazard assessments for other Cascade volcanic systems, such as South Sister, geologic insights from studies of the Mt. Mazama eruption can be considered through the use of SDVs. Similarly, the Katmai eruption of 1912 and the Mt. Pinatubo eruption of 1991 should not be considered expected events in the Cascades during the next several decades, but such eruptions are completely within the realm of possibility and can be considered in hazards assessments through the use of SDVs.

As an example, we constructed an SDV for tephra accumulation in León, based on a set of parameters outside the range of past activity, but nonetheless possible, given eruptions from basaltic cinder cones in general. This model assumes an 8km high eruption column, 100m/s) eruption velocity, and total volume of 1 × 108 m3. For comparison, the 71day eruption of Tolbachik volcano, Kamchatka, sustained 3-10km high eruption columns and erupted 9 × 108 m3 (0.42km3 DRE) of tephra (Doubik and Hill, 1999). It is also assumed that León lies on the major axis of dispersion and that the wind speed is 15m/s during the eruption. Such an eruption, larger than past eruptions at Cerro Negro, is at the approximate upper bound of VEI 3 activity. Given these parameters and based on the Suzuki model, the SDV for tephra accumulation in León is 47cm.

Discussion and Conclusions

The hazard models presented here are strictly empirical, rather than predictive. The historical and geologic record of events is used to temper and refine the numerical analysis, and to temper the interpretation of the results. Although short, the record of eruptions at Cerro Negro is more complete than many volcanoes. For many volcanoes, analogous volcanic eruptions at other volcanoes may be needed to augment the record and to completely reflect the natural variation in expected eruptions.

The Suzuki model does not capture the physics of volcanic eruption completely. Rather, this model simplifies the physics in several ways. For example, the wind field is considered to be uniform with height above the volcanic vent, and particle motion in the column is treated probabilistically rather than with analytical determinism. Used in a stochastic fashion and calibrated by independent observations, the model works reasonably well despite such limitations. This dynamic exists in much of volcanic hazard assessment. Other examples of the effective use of simplified models in volcanology include Iverson, et al. (1997) and Wadge et al. (1994).

When integrated with geologic data, as we have attempted here (Figures 2 and 3), such simplified models become efficient tools for volcanic hazards mitigation. This is different from attempting to forecast the outcome of a specific eruption, during which some variables, such as atmospheric conditions, are possibly well known. Although more dramatic to forecast the trajectory of an ash cloud as an eruption progresses, it is extremely useful to provide communities with a long-term forecast, prior to volcanic activity, so that informed decisions about building location, construction, and response can be formulated.

For Cerro Negro volcano, we conclude from analysis of the geologic data and numerical simulation that the probable tephra accumulation in León, given a volcanic eruption, of > 1cm is approximately 29%, > 4cm is approximately 9%, > 10cm is approximately 5%. For smaller, more frequent eruptions our analysis relies most heavily on observation. For larger, less frequent eruptions, our analysis relies more on the results of the numerical simulation. The SDV for tephra accumulation is León isapproximately 0.5m. In other words, given our current knowledge about this volcano, we do not envision any circumstances under which an eruption would result in more than approximately 0.5m of tephra deposition in León.

The main issues that emerge from application of this style of hazard assessment involve estimation of parameter distributions. For tephra accumulation, a clearer understanding of the links between magma properties and the parameters column height, eruption velocity, and total eruption volume has the potential of improving the hazard assessment. In time, the uniform random distributions used for parameters like eruption velocity might be replaced by more realistic distributions, or calculated directly from magma rheologic properties, as more about the physical basis of these links emerge. The importance of such links can only be appraised by progressing from the basic data and models through the hazard assessment. In this sense, results of probabilistic assessments can provide guidance about the volcanological research most likely to

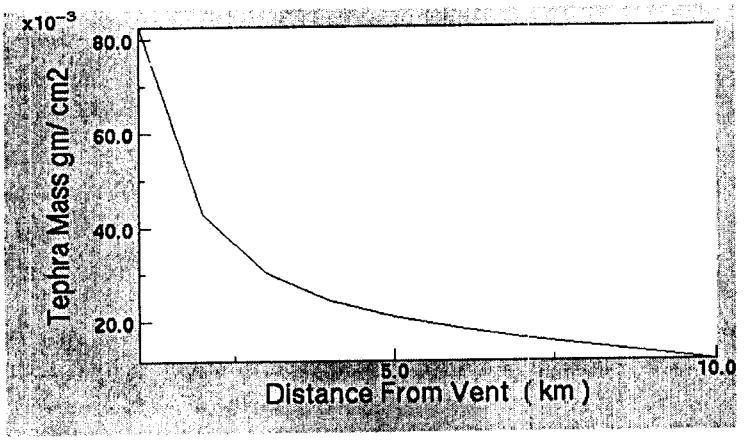
result in volcanic hazard reduction.

Ash Dispersion How To

The Ash Mass on the major axis of dispersion is calculated by the Ash_Dispersion applet. The first step is to enter the requested values for each field. Descriptions of each parameter and its corresponding units are given below for clarity. For some sample values, click on the button. total amount of material erupted in grams (e.g., 1.0e10). minimum particle size of tephra erupted in phi units (e.g., -5.0). maximum particle size of tephra erupted in phi units (e.g., 5.0). mean particle size of tephra erupted in phi units (e.g.,-1.0). standard deviation particle size of tephra erupted in phi units (e.g., 1.0). clast density of individual pyroclasts, assumed to be independent of grainsize in gm/cm3 (e.g., 0.8). particle shape factor, assumed to be independent of grainsize It is dimensionless (e.g., 0.5). empirical factor controlling plume geometry, dimensionless (e.g., 0.1). eruption velocity in cm/s (e.g., 10000.0). eruption column height in km (e.g., 10.0). average wind speed in cm/s (e.g., 1000.0). Minimum distance from the vent in km (e.g., 1). Do not enter decimal places. An integer is required here. Maximum distance from the vent in km (e.g., 10). This parameter corresponds to the eruption column height. Do not enter decimal places. An integer is required here.

To start the java applet, after you have filled in the parameters, click on the calculate button.

The Ash Dispersion java applet produces the following graphical output for the sample parameters.

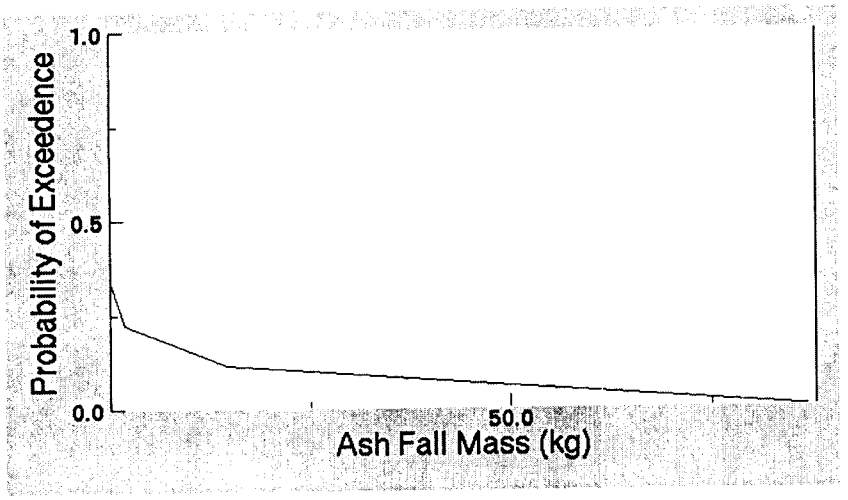


If you wish to change the parameters and create another graph, you can change each value individually, or click on the Clear the Fields button to clear all of the values and start over with a new form.

Ash Probability of Exceedence

The Ash Mass Probability of Exceedence is calculated by the prob_graph.java code. The first step is to enter the requested values for each field. Descriptions of each parameter and its corresponding units are given below for clarity. If you do not want to show a size variation for a particular parameter, enter the same number in both MIN and MAX fields. For some sample values, click on the button.
minimum and maximum total amount of material erupted in grams (e.g., MIN=1.0e10, MAX=1.0e15). minimum and maximum minimum particle size of tephra erupted in phi units (e.g., MIN=-5.0, MAX=-5.0). minimum and maximum maximum particle size of tephra erupted in phi units (e.g., MIN=5.0, MAX=5.0). minimum and maximum mean particle size of tephra erupted in phi units (e.g.,MIN=-1.0, MAX=-1.0). minimum and maximum standard deviation particle size of tephra erupted in phi units (e.g., MIN=1.0, MAX=1.0). minimum and maximum clast density of individual pyroclasts, assumed to be independent of grainsize in gm/cm3 (e.g., MIN=0.8, MAX=0.8). minimum and maximum particle shape factor, assumed to be independent of grainsize It is dimensionless (e.g., MIN=0.5, MAX=0.5). minimum and maximum empirical factor controlling plume geometry, dimensionless (e.g., MIN=0.5, MAX=0.5). minimum and maximum eruption velocity in cm/s (e.g., MIN=50000.0, MAX=100000.0). minimum and maximum eruption column height in km (e.g., MIN=4.0, MAX=10.0). minimum and maximum average wind speed in cm/s (e.g., MIN=0.0, MAX=2000.0). the X coordinate of the point being calculated(e.g., 1e5) the Y coordinate of the point being calculated(e.g., 5e5) For each simulation, a random value is chosen between each MIN and MAX value for each parameter listed above. The Number of Simulations basically determines how many random sample points will be taken and plotted on the curve. The larger the Number of Simulations, the more representative the graph will be. (e.g., 10) To start the java applet, after you have filled in the parameters, click on the calculate button.

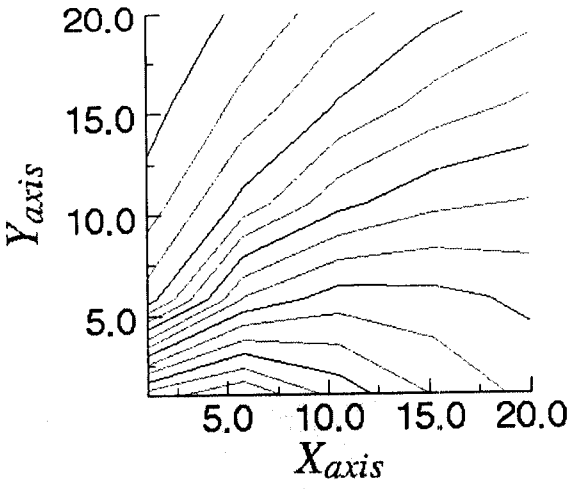
The Ash Dispersion java applet produces the following graphical output for the sample parameters.



If you wish to change the parameters and create another graph, you can change each value individually, or click on the Clear the Fields button to clear all of the values and start over with a new form.

Ash Contour

The Ash Mass Contour is calculated by the Ash_Contour code. The webpage produces a graphical output.



total amount of material erupted in grams (e.g., 1.0e10). minimum particle size of tephra erupted in phi units (e.g., -5.0). maximum particle size of tephra erupted in phi units (e.g.,5.0). mean particle size of tephra erupted in phi units (e.g.,-1.0). standard deviation particle size of tephra erupted in phi units (e.g., 1.0). clast density of individual pyroclasts, assumed to be independent of grainsize in gm/cm3 (e.g., 0.8). particle shape factor, assumed to be independent of grainsize It is dimensionless (e.g., 0.5). empirical factor controlling plume geometry, dimensionless (e.g., 0.5). eruption velocity in cm/s (e.g., 10000.0). eruption column height in km (e.g., 10.0). average wind speed in cm/s (e.g., 1000.0).

Program Name: ASH_PLUME
Class Name: Ash_Point
Date: 07/05/00
Release Version: 1.0
Client Name: USNRC
U. S. Nuclear Regulatory Commission
NRC Office of Nuclear Material Safety and Safeguard
Division of Waste Management
NRC Contract: NRC 20-01402-461
NRC Contact: Dr. John Trapp (301) 415-8063
CNWRA Contact: Dr. Charles Connor (210) 522-6649
Center For Nuclear Waste Regulatory Analyses
Southwest Research Institute
6220 Culebra Rd.
San Antonio, TX, 78238-5166, USA
cconnor@swri.edu

Documentation:
ASH_PLUME version 1.0 - Probabilistic Volcanic Hazard Assessment Methods for a Proposed High-Level Radioactive Waste Repository at Yucca Mountain, Nevada

CNWRA 2000-02

NUREG-Series Designator: N/A

DISCLAIMER

"This computer code / material was prepared as an account of work performed by the Center for Nuclear Waste Regulatory Analyses (CNWRA) for the Division of Waste Management of the Nuclear Regulatory Commission (NRC), an independent agency of the United States Government. Neither the developer(s) of the code nor any of their sponsors make any warranty, expressed or implied, or assume any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represent that its use would not infringe on privately-owned rights."

"In no event unless required by applicable law will the sponsors or those who have written or modified this code, be liable for damages, including any loss of profits, lost monies, or other special, incidental or consequential damages arising out of the use or inability to use the program (including but not limited to loss of data or data rendered inaccurate or losses sustained by third parties for failure of the program to operate with other programs), even if you have been advised of the possibility of such damages or for any claim by any other party."

by C. Connor and Brandi Winfrey

```
import java.applet.Applet;
import java.awt.Graphics;
import java.awt.Font;

public class Ash_Point extends Applet {

/**
 *  Subroutine CROSS computes the vector product of two vectors; i.e.,
 *
 *            (cx,cy,cz) = (ax,ay,az) X (bx,by,bz)
 **/

  Font f = new Font("TimesRoman", Font.BOLD, 20);
  String Total;
  String Dmin;
  String Dmax;
  String Dmean;
```

```
 String Dsigma;
 String Phiash;
 String Fshape;
 String beta;
 String w0;
 String height;
 String windvel;
 String xspace;
 String yspace;

public void init() {
   Total = new String(" ");
   Dmin = new String(" ");
   Dmax = new String(" ");
   Dmean = new String(" ");
   Dsigma = new String(" ");
   Phiash = new String(" ");
   Fshape = new String(" ");
   beta = new String(" ");
   w0 = new String(" ");
   height = new String(" ");
   windvel = new String(" ");
   xspace = new String(" ");
   yspace = new String(" ");
}

public void SetMessage(String MsgText1, String MsgText2, String MsgText3,
                String MsgText4, String MsgText5, String MsgText6,
                String MsgText7, String MsgText8, String MsgText9,
                String MsgText10, String MsgText11, String MsgText12,
                String MsgText13) {
   Total = MsgText1;
   Dmin = MsgText2;
   Dmax = MsgText3;
   Dmean = MsgText4;
   Dsigma = MsgText5;
   Phiash = MsgText6;
   Fshape = MsgText7;
   beta = MsgText8;
```

```
    w0 = MsgText9;
    height = MsgText10;
    windvel = MsgText11;
    xspace = MsgText12;
    yspace = MsgText13;
    repaint();
  }

  public void paint(Graphics g) {
    g.setFont( f );

    double newTotal = Double.valueOf( Total ).doubleValue();
    double newDmin = Double.valueOf( Dmin ).doubleValue();
    double newDmax = Double.valueOf( Dmax ).doubleValue();
    double newDmean = Double.valueOf( Dmean ).doubleValue();
    double newDsigma = Double.valueOf( Dsigma ).doubleValue();
    double newPhiash= Double.valueOf( Phiash ).doubleValue();
    double newFshape= Double.valueOf( Fshape ).doubleValue();
    double newbeta= Double.valueOf( beta ).doubleValue();
    double neww0= Double.valueOf( w0 ).doubleValue();
    double newheight= Double.valueOf( height ).doubleValue();
    double newwindvel= Double.valueOf( windvel ).doubleValue();
    double newxspace= Double.valueOf( xspace ).doubleValue();
    double newyspace= Double.valueOf( yspace ).doubleValue();

    Suzuki2 ash1 = new Suzuki2(newTotal, newDmin, newDmax, newDmean,
                        newDsigma, newbeta, neww0, newheight,
                        newwindvel, newPhiash, newFshape);
    double result;
    result = ash1.suzuki1( newxspace, newyspace );

    g.drawString( "The Result is: " + result, 15, 100 );
  }
}
```

****

Scientific Notebook 115E
20-1402-461
Inititals: CC

Chuck Connor
April 1, 2000

/**

****

Program Name: ASH_PLUME

Class Name: Ash_Dispersion

Date: 07/05/00

Release Version: 1.0

Client Name: USNRC

U. S. Nuclear Regulatory Commission
NRC Office of Nuclear Material Safety and Safeguard
Division of Waste Management

NRC Contract: NRC 20-01402-461

NRC Contact: Dr. John Trapp (301) 415-8063

CNWRA Contact: Dr. Charles Connor (210) 522-6649

Center For Nuclear Waste Regulatory Analyses
Southwest Research Institute
6220 Culebra Rd.
San Antonio, TX, 78238-5166, USA
cconnor@swri.edu

Documentation:
ASH_PLUME version 1.0 - Probabilistic Volcanic
Hazard Assessment Methods for a Proposed
High-Level Radioactive Waste Repository
at Yucca Mountain, Nevada

CNWRA 2000-02

NUREG-Series Designator: N/A

```
**************************************************************************

**************************************************************************
**************************************************************************
DISCLAIMER

**************************************************************************
**************************************************************************

        "This computer code / material was prepared as an account of work
performed by the Center for Nuclear Waste Regulatory Analyses (CNWRA)
for the Division of Waste Management of the Nuclear Regulatory
Commission (NRC), an independent agency of the United States Government.
Neither the developer(s) of the code nor any of their sponsors make any
warranty, expressed or implied, or assume any legal liability or
responsibility for the accuracy, completeness, or usefulness of any
information, apparatus, product, or process disclosed, or represent that
its use would not infringe on privately-owned rights."

        "In no event unless required by applicable law will the sponsors or
those who have written or modified this code, be liable for damages,
including any loss of profits, lost monies, or other special, incidental
or consequential damages arising out of the use or inability to use
the program (including but not limited to loss of data or data rendered
inaccurate or losses sustained by third parties for failure of the program
to operate with other programs), even if you have been advised of the
possibility of such damages or for any claim by any other party."

**************************************************************************
**************************************************************************
by C. Connor and Brandi Winfrey

**************************************************************************
**************************************************************************/
import java.applet.Applet;
import java.awt.Graphics;
import java.awt.Font;
import java.awt.*;
```

```
import graph.*;

public class Ash_Dispersion extends Applet {

/**
 * This program calculates the expected ash fall mass along
 * the major axis of dispersion for given input parameters.
 **/

  G2Dint graph      = new G2Dint();  // Graph class to do the plotting
  Axis xaxis;
  Axis yaxis;
  DataSet data;

  Font f = new Font("TimesRoman", Font.BOLD, 20);
  String Total;
  String Dmin;
  String Dmax;
  String Dmean;
  String Dsigma;
  String Phiash;
  String Fshape;
  String beta;
  String w0;
  String height;
  String windvel;
  String XMIN;
  String XMAX;

 public void init() {

//   Label title        = new Label("Ash Mass on the Major Axis of Dispersion",Label.CENTER);

    Panel panel        = new Panel();
    GridBagLayout gridbag  = new GridBagLayout();
    GridBagConstraints  c  = new GridBagConstraints();
    Font font          = new Font("Helvetica",Font.PLAIN,14);

//   title.setFont(new Font("TimesRoman",Font.PLAIN,24));
```

```
    setLayout(new BorderLayout() );
//    add("North",title);
    add("Center",panel);

    Total = new String(" ");
    Dmin = new String(" ");
    Dmax = new String(" ");
    Dmean = new String(" ");
    Dsigma = new String(" ");
    Phiash = new String(" ");
    Fshape = new String(" ");
    beta = new String(" ");
    w0 = new String(" ");
    height = new String(" ");
    windvel = new String(" ");
    XMIN = new String(" ");
    XMAX = new String(" ");

    panel.setLayout(gridbag);

    c.weightx = 1.0;
    c.weighty = 1.0;
    c.gridwidth = 4;
    c.gridwidth=GridBagConstraints.REMAINDER;
    c.fill  =  GridBagConstraints.BOTH;

    gridbag.setConstraints(graph,c);

    panel.add(graph);

    xaxis = graph.createXAxis();
    xaxis.setTitleText("Distance From Vent  (km)");
    xaxis.setTitleFont(new Font("Helvetica",Font.PLAIN,18));
    xaxis.setLabelFont(new Font("Helvetica",Font.PLAIN,14));

    yaxis = graph.createYAxis();
    yaxis.setTitleText("Tephra Mass gm/ cm2");
    yaxis.setTitleFont(new Font("Helvetica",Font.PLAIN,18));
```

```
    yaxis.setLabelFont(new Font("Helvetica",Font.PLAIN,14));

    graph.setDataBackground(new Color(255,200,175));
    graph.setBackground(new Color(200,150,100));

    data = new DataSet();

    xaxis.attachDataSet(data);
    yaxis.attachDataSet(data);
    graph.attachDataSet(data);
  }

  public void SetMessage(String MsgText1, String MsgText2, String MsgText3,
                         String MsgText4, String MsgText5, String MsgText6,
                         String MsgText7, String MsgText8, String MsgText9,
                         String MsgText10, String MsgText11, String MsgText12,
                         String MsgText13) {
    Total  = MsgText1;
    Dmin   = MsgText2;
    Dmax   = MsgText3;
    Dmean  = MsgText4;
    Dsigma = MsgText5;
    Phiash = MsgText6;
    Fshape = MsgText7;
    beta   = MsgText8;
    w0     = MsgText9;
    height = MsgText10;
    windvel = MsgText11;
    XMIN   = MsgText12;
    XMAX   = MsgText13;

    repaint();
    graph.repaint();
  }

  public void paint(Graphics g) {
    boolean error = false;
    g.setFont( f );
```

```
double newTotal = Double.valueOf( Total ).doubleValue();
double newDmin = Double.valueOf( Dmin ).doubleValue();
double newDmax = Double.valueOf( Dmax ).doubleValue();
double newDmean = Double.valueOf( Dmean ).doubleValue();
double newDsigma = Double.valueOf( Dsigma ).doubleValue();
double newPhiash= Double.valueOf( Phiash ).doubleValue();
double newFshape= Double.valueOf( Fshape ).doubleValue();
double newbeta= Double.valueOf( beta ).doubleValue();
double neww0= Double.valueOf( w0 ).doubleValue();
double newheight= Double.valueOf( height ).doubleValue();
double newwindvel= Double.valueOf( windvel ).doubleValue();
int newXMIN= Integer.valueOf( XMIN ).intValue();
int newXMAX= Integer.valueOf( XMAX ).intValue();

int j, kount;
double d[] = new double[50];

Suzuki2 ash1 = new Suzuki2(newTotal, newDmin, newDmax, newDmean,
                    newDsigma, newbeta, neww0, newheight,
                    newwindvel, newPhiash, newFshape);
System.out.println("total = " + newTotal);
j =0;
for (kount=newXMIN; kount <= newXMAX; kount++, j +=2) {
  try {
     this.showStatus("Calculating point: "+ kount);
     d[j] = (double)kount;
     d[j+1] = ash1.suzuki1(d[j] * 1.0e5, 0.0);

  } catch (Exception e) {error = true;}

}
data.deleteData();

try {
    data.append(d,j/2);
} catch(Exception e) {
   this.showStatus("Error while appending data!");
   System.out.println("Error while appending data!");
   return;
```

```
  }
 }
}
```

****

/**

****

Program Name: ASH_PLUME

Class Name: Ash_Contour

Date: 07/05/00

Release Version: 1.0

Client Name: USNRC

U. S. Nuclear Regulatory Commission
NRC Office of Nuclear Material Safety and Safeguard
Division of Waste Management

NRC Contract: NRC 20-01402-461

NRC Contact: Dr. John Trapp (301) 415-8063

CNWRA Contact: Dr. Charles Connor (210) 522-6649

Center For Nuclear Waste Regulatory Analyses
Southwest Research Institute
6220 Culebra Rd.
San Antonio, TX, 78238-5166, USA
cconnor@swri.edu

Documentation:
ASH_PLUME version 1.0 - Probabilistic Volcanic
Hazard Assessment Methods for a Proposed
High-Level Radioactive Waste Repository

Scientific Notebook 115E
20-1402-461
Initials: CC

Chuck Connor
April 1, 2000

at Yucca Mountain, Nevada

CNWRA 2000-02

NUREG-Series Designator: N/A

****

****
****
DISCLAIMER

****
****

"This computer code / material was prepared as an account of work performed by the Center for Nuclear Waste Regulatory Analyses (CNWRA) for the Division of Waste Management of the Nuclear Regulatory Commission (NRC), an independent agency of the United States Government. Neither the developer(s) of the code nor any of their sponsors make any warranty, expressed or implied, or assume any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represent that its use would not infringe on privately-owned rights."

"In no event unless required by applicable law will the sponsors or those who have written or modified this code, be liable for damages, including any loss of profits, lost monies, or other special, incidental or consequential damages arising out of the use or inability to use the program (including but not limited to loss of data or data rendered inaccurate or losses sustained by third parties for failure of the program to operate with other programs), even if you have been advised of the possibility of such damages or for any claim by any other party."

****
****
by C. Connor and Brandi Winfrey

****

```
**************************************************************************/
import java.awt.*;
import java.applet.*;
import java.net.URL;
import java.util.*;
import graph.*;

public class Ash_Contour extends Applet {

  LoadData dynamic;
  Contour graph;
//  Label title;
  Axis   xaxis;
  Axis   yaxis;

  Font f = new Font("TimesRoman", Font.BOLD, 20);
  String Total;
  String Dmin;
  String Dmax;
  String Dmean;
  String Dsigma;
  String Phiash;
  String Fshape;
  String beta;
  String w0;
  String height;
  String windvel;

  String nx;
  String ny;
  String xmin;
  String xmax;
  String ymin;
  String ymax;


  public void init() {

    Total = new String(" ");
```

```
    Dmin = new String(" ");
    Dmax = new String(" ");
    Dmean = new String(" ");
    Dsigma = new String(" ");
    Phiash = new String(" ");
    Fshape = new String(" ");
    beta = new String(" ");
    w0 = new String(" ");
    height = new String(" ");
    windvel = new String(" ");

    nx = new String(" ");
    ny = new String(" ");
    xmin = new String(" ");
    xmax = new String(" ");
    ymin = new String(" ");
    ymax = new String(" ");

/**
 *  Instantiate the Contour class and calculate the data
 **/
    graph = new Contour();
    graph.setDataBackground(new Color(0.933f,0.914f,0.749f));
    graph.setContourColor(new Color(0.180f,0.545f,0.341f));
    graph.setLabelledContourColor(new Color(0.5f,.0f,0.0f));

    graph.setLabelPrecision(2);
    graph.setLabelSignificance(2);

    this.showStatus("Creating Data to Contour!");
    graph.square      = true;
/**
 *  Build the title and place it at the top of the graph
 **/
    graph.setFont(new Font("TimesRoman",Font.PLAIN,25));
//    title = new Label("Contouring Example", Label.CENTER);
//    title.setFont(new Font("TimesRoman",Font.PLAIN,25));

    setLayout( new BorderLayout() );
```

```
//    add("North", title);
      add("Center", graph);
/**
 *  Instantiate the xaxis and attach the dataset.
 **/
    xaxis = graph.createXAxis();
    xaxis.setTitleText("X_axis");
    xaxis.setTitleColor(Color.magenta);
    xaxis.setTitleFont(new Font("TimesRoman",Font.ITALIC,25));
    xaxis.setLabelFont(new Font("Helvetica",Font.PLAIN,20));

/**
 *  Instantiate the yaxis and attach the dataset.
 **/
    yaxis = graph.createYAxis();
    yaxis.setTitleText("Y_axis");
    yaxis.setTitleColor(Color.magenta);
    yaxis.setTitleFont(new Font("TimesRoman",Font.ITALIC,25));
    yaxis.setLabelFont(new Font("Helvetica",Font.PLAIN,20));
  }

  public void SetMessage(String MsgText1, String MsgText2, String MsgText3,
                         String MsgText4, String MsgText5, String MsgText6,
                         String MsgText7, String MsgText8, String MsgText9,
                         String MsgText10, String MsgText11,
                         String MsgText12, String MsgText13,
                         String MsgText14, String MsgText15,
                         String MsgText16, String MsgText17) {
    Total  = MsgText1;
    Dmin   = MsgText2;
    Dmax   = MsgText3;
    Dmean  = MsgText4;
    Dsigma = MsgText5;
    Phiash = MsgText6;
    Fshape = MsgText7;
    beta   = MsgText8;
    w0     = MsgText9;
    height = MsgText10;
    windvel = MsgText11;
```

```
  nx   = MsgText12;
  xmin = MsgText13;
  xmax = MsgText14;
  ny   = MsgText15;
  ymin = MsgText16;
  ymax = MsgText17;

  createGrid(graph);
  graph.repaint();
}

public void createGrid(Contour graph) {
   int i,j;
   int count;
   int newnx = Integer.valueOf( nx ).intValue();
   int newny = Integer.valueOf( ny ).intValue();

   double d[] = new double[newnx * newny];
   double array[] = new double[2 * (d.length)];
   double x, y;

   double newTotal = Double.valueOf( Total ).doubleValue();
   double newDmin = Double.valueOf( Dmin ).doubleValue();
   double newDmax = Double.valueOf( Dmax ).doubleValue();
   double newDmean = Double.valueOf( Dmean ).doubleValue();
   double newDsigma = Double.valueOf( Dsigma ).doubleValue();
   double newPhiash= Double.valueOf( Phiash ).doubleValue();
   double newFshape= Double.valueOf( Fshape ).doubleValue();
   double newbeta= Double.valueOf( beta ).doubleValue();
   double neww0= Double.valueOf( w0 ).doubleValue();
   double newheight= Double.valueOf( height ).doubleValue();
   double newwindvel= Double.valueOf( windvel ).doubleValue();

   double newxmin= Double.valueOf( xmin ).doubleValue();
   double newxmax= Double.valueOf( xmax ).doubleValue();
   double newymin= Double.valueOf( ymin ).doubleValue();
   double newymax= Double.valueOf( ymax ).doubleValue();

   Suzuki3 ash1 = new Suzuki3(newTotal, newDmin, newDmax, newDmean,
```

```
                    newDsigma, newbeta, neww0, newheight,
                    newwindvel, newPhiash, newFshape);

    count = 0;

    for(j=0; j<newny; j++) {
      y = newymin + (newymax-newymin)/(double)(newny-1) * (double)j;

      for(i=0; i<newnx; i++) {
        x = newxmin + (newxmax-newxmin)/(double)(newnx-1) * (double)i;

          d[count] = Math.log(ash1.suzuki1(x*5.0e5, y*5.0e5))/Math.log(10.0);
          count++;
        }
      }

      System.arraycopy(d, 0, array, 0, newnx*newny);
      for(int l=0; l<array.length; l++) {
        System.out.println("array" + array[l]);
      }
//      for(int k=0; k<d.length; k++) {
//        array[count] = d[k];
//        count++;
//      }
//      System.out.println("array" + array[count]);

      graph.setGrid(array,newnx,newny);
      graph.setRange(newxmin,newxmax,newymin,newymax);
      graph.setLimitsToGrid(true);
      graph.setLabelLevels(3);
      graph.setNLevels(20);
    }
}

*****************************************************************************

/**

*****************************************************************************
```

Scientific Notebook 115E
20-1402-461
Initials: CC

Chuck Connor
April 1, 2000

Program Name: ASH_PLUME

Class Name: prob_graph

Date: 07/05/00

Release Version: 1.0

Client Name: USNRC

U. S. Nuclear Regulatory Commission
NRC Office of Nuclear Material Safety and Safeguard
Division of Waste Management

NRC Contract: NRC 20-01402-461

NRC Contact: Dr. John Trapp (301) 415-8063

CNWRA Contact: Dr. Charles Connor (210) 522-6649

Center For Nuclear Waste Regulatory Analyses
Southwest Research Institute
6220 Culebra Rd.
San Antonio, TX, 78238-5166, USA
cconnor@swri.edu

Documentation:
ASH_PLUME version 1.0 - Probabilistic Volcanic
Hazard Assessment Methods for a Proposed
High-Level Radioactive Waste Repository
at Yucca Mountain, Nevada

CNWRA 2000-02

NUREG-Series Designator: N/A

****

****

```
**************************************************************************
DISCLAIMER

**************************************************************************
**************************************************************************

        "This computer code / material was prepared as an account of work
performed by the Center for Nuclear Waste Regulatory Analyses (CNWRA)
for the Division of Waste Management of the Nuclear Regulatory
Commission (NRC), an independent agency of the United States Government.
Neither the developer(s) of the code nor any of their sponsors make any
warranty, expressed or implied, or assume any legal liability or
responsibility for the accuracy, completeness, or usefulness of any
information, apparatus, product, or process disclosed, or represent that
its use would not infringe on privately-owned rights."

        "In no event unless required by applicable law will the sponsors or
those who have written or modified this code, be liable for damages,
including any loss of profits, lost monies, or other special, incidental
or consequential damages arising out of the use or inability to use
the program (including but not limited to loss of data or data rendered
inaccurate or losses sustained by third parties for failure of the program
to operate with other programs), even if you have been advised of the
possibility of such damages or for any claim by any other party."

**************************************************************************
**************************************************************************
by C. Connor and Brandi Winfrey

**************************************************************************
**************************************************************************/
import java.applet.Applet;
import java.awt.Graphics;
import java.awt.Font;
import java.awt.*;
import java.math.*;

import graph.*;
```

```
public class prob_graph extends Applet {

/**
 * This program calculates the expected ash fall mass along
 * the major axis of dispersion for given input parameters.
 **/

  G2Dint graph       = new G2Dint();   // Graph class to do the plotting
  Axis xaxis;
  Axis yaxis;
  DataSet data;

  Font f = new Font("TimesRoman", Font.BOLD, 20);
  String Total_min;
  String Total_max;
  String Dmin_min;
  String Dmin_max;
  String Dmax_min;
  String Dmax_max;
  String Dmean_min;
  String Dmean_max;
  String Dsigma_min;
  String Dsigma_max;
  String Phiash_min;
  String Phiash_max;
  String Fshape_min;
  String Fshape_max;
  String beta_min;
  String beta_max;
  String w0_min;
  String w0_max;
  String height_min;
  String height_max;
  String windvel_min;
  String windvel_max;
  String XSPACE;
  String YSPACE;
  String NUM_SIMS;
```

```
public void init() {

  Panel panel          = new Panel();
  GridBagLayout gridbag  = new GridBagLayout();
  GridBagConstraints  c  = new GridBagConstraints();
  Font font            = new Font("Helvetica",Font.PLAIN,14);


  setLayout(new BorderLayout() );
  add("Center",panel);

  Total_min   = new String(" ");
  Total_max   = new String(" ");
  Dmin_min    = new String(" ");
  Dmin_max    = new String(" ");
  Dmax_min    = new String(" ");
  Dmax_max    = new String(" ");
  Dmean_min   = new String(" ");
  Dmean_max   = new String(" ");
  Dsigma_min  = new String(" ");
  Dsigma_max  = new String(" ");
  Phiash_min  = new String(" ");
  Phiash_max  = new String(" ");
  Fshape_min  = new String(" ");
  Fshape_max  = new String(" ");
  beta_min    = new String(" ");
  beta_max    = new String(" ");
  w0_min      = new String(" ");
  w0_max      = new String(" ");
  height_min  = new String(" ");
  height_max  = new String(" ");
  windvel_min = new String(" ");
  windvel_max = new String(" ");
  XSPACE      = new String(" ");
  YSPACE      = new String(" ");
  NUM_SIMS    = new String(" ");

  panel.setLayout(gridbag);
```

```
    c.weightx   = 1.0;
    c.weighty   = 1.0;
    c.gridwidth = 4;
    c.gridwidth = GridBagConstraints.REMAINDER;
    c.fill      = GridBagConstraints.BOTH;

    gridbag.setConstraints(graph,c);

    panel.add(graph);

    xaxis = graph.createXAxis();
    xaxis.setTitleText("Ash Fall Mass (kg)");
    xaxis.setTitleFont(new Font("Helvetica",Font.PLAIN,18));
    xaxis.setLabelFont(new Font("Helvetica",Font.PLAIN,14));

    yaxis = graph.createYAxis();
    yaxis.setTitleText("Probability of Exceedence");
    yaxis.setTitleFont(new Font("Helvetica",Font.PLAIN,18));
    yaxis.setLabelFont(new Font("Helvetica",Font.PLAIN,14));

    graph.setDataBackground(new Color(255,200,175));
    graph.setBackground(new Color(200,150,100));

    data = new DataSet();

    xaxis.attachDataSet(data);
    yaxis.attachDataSet(data);
    graph.attachDataSet(data);
}

public void SetMessage(String MsgText1,  String MsgText2,  String MsgText3,
                       String MsgText4,  String MsgText5,  String MsgText6,
                       String MsgText7,  String MsgText8,  String MsgText9,
                       String MsgText10, String MsgText11, String MsgText12,
                       String MsgText13, String MsgText14, String MsgText15,
                       String MsgText16, String MsgText17, String MsgText18,
                       String MsgText19, String MsgText20, String MsgText21,
                       String MsgText22, String MsgText23, String MsgText24,
                       String MsgText25) {
```

```
  Total_min  = MsgText1;
  Total_max  = MsgText2;
  Dmin_min   = MsgText3;
  Dmin_max   = MsgText4;
  Dmax_min   = MsgText5;
  Dmax_max   = MsgText6;
  Dmean_min  = MsgText7;
  Dmean_max  = MsgText8;
  Dsigma_min = MsgText9;
  Dsigma_max = MsgText10;
  Phiash_min = MsgText11;
  Phiash_max = MsgText12;
  Fshape_min = MsgText13;
  Fshape_max = MsgText14;
  beta_min   = MsgText15;
  beta_max   = MsgText16;
  w0_min     = MsgText17;
  w0_max     = MsgText18;
  height_min = MsgText19;
  height_max = MsgText20;
  windvel_min = MsgText21;
  windvel_max = MsgText22;
  XSPACE     = MsgText23;
  YSPACE     = MsgText24;
  NUM_SIMS   = MsgText25;

  repaint();
  graph.repaint();
}

public void paint(Graphics g) {
  boolean error = false;
  g.setFont( f );

  double newTotal_min  = Double.valueOf( Total_min ).doubleValue();
  double newDmin_min   = Double.valueOf( Dmin_min ).doubleValue();
  double newDmax_min   = Double.valueOf( Dmax_min ).doubleValue();
  double newDmean_min  = Double.valueOf( Dmean_min ).doubleValue();
  double newDsigma_min = Double.valueOf( Dsigma_min ).doubleValue();
```

```
  double newPhiash_min  = Double.valueOf( Phiash_min ).doubleValue();
  double newFshape_min  = Double.valueOf( Fshape_min ).doubleValue();
  double newbeta_min    = Double.valueOf( beta_min ).doubleValue();
  double neww0_min      = Double.valueOf( w0_min ).doubleValue();
  double newheight_min  = Double.valueOf( height_min ).doubleValue();
  double newwindvel_min = Double.valueOf( windvel_min ).doubleValue();

  double newTotal_max   = Double.valueOf( Total_max ).doubleValue();
  double newDmin_max    = Double.valueOf( Dmin_max ).doubleValue();
  double newDmax_max    = Double.valueOf( Dmax_max ).doubleValue();
  double newDmean_max   = Double.valueOf( Dmean_max ).doubleValue();
  double newDsigma_max  = Double.valueOf( Dsigma_max ).doubleValue();
  double newPhiash_max  = Double.valueOf( Phiash_max ).doubleValue();
  double newFshape_max  = Double.valueOf( Fshape_max ).doubleValue();
  double newbeta_max    = Double.valueOf( beta_max ).doubleValue();
  double neww0_max      = Double.valueOf( w0_max ).doubleValue();
  double newheight_max  = Double.valueOf( height_max ).doubleValue();
  double newwindvel_max = Double.valueOf( windvel_max ).doubleValue();

  double newXSPACE  = Double.valueOf( XSPACE ).doubleValue();
  double newYSPACE  = Double.valueOf( YSPACE ).doubleValue();
  int newNUM_SIMS  = Integer.valueOf( NUM_SIMS ).intValue();



// Do the math here ------------------------------------------------

  int i=0, j=0, kount=0;
  double out[] = new double[newNUM_SIMS];
  double d[] = new double[2 * out.length + 1];

  for (i=0; i < newNUM_SIMS; i++) {
    Rnum num = new Rnum();
    double newTotal  = num.Rand_num( newTotal_min,   newTotal_max);
    double newDmin   = num.Rand_num( newDmin_min,    newDmin_max);
    double newDmax   = num.Rand_num( newDmax_min,    newDmax_max);
    double newDmean  = num.Rand_num( newDmean_min,   newDmean_max);
    double newDsigma = num.Rand_num( newDsigma_min,  newDsigma_max);
    double newPhiash = num.Rand_num( newPhiash_min,  newPhiash_max);
```

```
    double newFshape  = num.Rand_num( newFshape_min,   newFshape_max);
    double newbeta    = num.Rand_num(   newbeta_min,     newbeta_max);
    double neww0      = num.Rand_num(     neww0_min,       neww0_max);
    double newheight  = num.Rand_num( newheight_min,   newheight_max);
    double newwindvel = num.Rand_num(newwindvel_min,  newwindvel_max);


    Suzuki2 ash1 = new Suzuki2(newTotal, newDmin, newDmax, newDmean,
                        newDsigma, newbeta, neww0, newheight,
                        newwindvel, newPhiash, newFshape);
    try {
       this.showStatus("Calculating point: "+ i);
       out[i] = ash1.suzuki1(newXSPACE, newYSPACE);
       System.out.println("" + newwindvel + " " + newheight + " " + neww0 + " " + out[i]);
    } catch (Exception e) {error = true;}
  }

  Sort order = new Sort();
  out = order.mergesort(out);

  for (kount=0; kount<newNUM_SIMS; kount++, j +=2) {
    try {
       this.showStatus("Calculating point: "+ kount);
       d[j] = out[kount];
       d[j+1] = (double)1 - ((double)kount /
             ((double)newNUM_SIMS - (double)1));
    } catch (Exception e) {error = true;}
  }
  data.deleteData();

  try {
      data.append(d,j/2);
  } catch(Exception e) {
     this.showStatus("Error while appending data!");
     System.out.println("Error while appending data!");
     return;
  }
 }
}
```

Chuck Connor
April 1, 2000

Probability Maps

The probability maps of 1cm and 4cm tephra accumulation (Figure 7a and Figure 7b) were created from a grid containg the mass of tephra deposited at each grid point. This grid was created by tephragrid.c. The grid was then used by the make_probmap script to produce the two figures. This script uses the Generic Mapping Tools, a public-domain data processing and display software package, to contour the data and then create the postscript of the two figures.

make_probmap

```
# /bin/bash
#
# make_probmap
#
# Purpose: Make a probabilty map of a utm grid of ash probabities using GMT (Generic
#       Mapping Tools - data processing and display software package).
#       The utm grid has a range from 437400 to 393400 meters on the
#       East-West axis and 1362500 to 1402500 meters on the North-South
#       axis.
#
#       -Nathan Franklin
#        March 6, 2000
#
# GMT progs:   surface, psbasemap, grdcontour, psxy, pstext
# Unix progs:  rm, gs


#surface
# does the interpolation with a minimum curvature algorithm and produces a grid
#
# -I1000    means the grid spacing is 1000 units (meters) in each direction
# -Gfilename indicates the output file
# -R        specifies the west,east,south, and north bounds of the map
# -V        verbose mode
surface  -I1000 gmt_prob -G1cmprob_surface -R501400/535400/1363500/1391500  -V


#psbasemap
# creates a base map
```

```
#
# -JX5.0iY4.12i     means the map will be 5 inches wide and 4.12 inches tall
# -Ba10000f1000/WSne draws the frame, 10000 m label, add 1000 m ticks,
#                   label on west and south sides only
# -X1i               means the origin of the plot is shifted by 1 inch on the x-axis
# -P                 select portrait plotting mode
# -K                 means more PostScript will be appended later
psbasemap -JX5.0iY4.12i -Ba10000f1000/WSne -R501400m/535400m/1363500m/1391500m
-X1i  -P -K -V   > 1cmprob_map.ps


#grdcontour
# draws the contours from the grid
#
# -JX5.0iY4.12i  means the map will be 5 inches wide and 4.12 inches tall
# -C5            means there is a 5  contour interval
# -A20f13        means the contours are annotated every 20 units  and have a font size
#                of 13
# -W1p           sets line width
#-L1/100         Limit range: do not draw contours for data values below
#                1 or above 100 units.
# -O             selects overlay plot mode
grdcontour 1cmprob_surface -JX5.0iY4.12i -C5 -A20f13 -L1/100 -W1p  -P -K -V -O  >>
1cmprob_map.ps


#psxy
# draws Leon using leon_outline.txt, which has xy points that connect to make an outline of the
city
#
#-G230      shading of the city (between 0 and 250)
#-W.25p     width of outline "line"

psxy leon_outline.txt -JX5.0iY4.12i  -R501400/535400/1363500/1391500   -O -G230 -K -W.25p
-V >> 1cmprob_map.ps


#psxy
# draws a triangle at Cerro Negro using one xy point from the file cerronegro.txt.
```

```
#
#-St.25i   draws a triangle at that point with a size of a circle with
#          .25 in radius.
#-G0     gives a shading between 0-250

psxy cerronegro.txt -JX5.0iY4.12i  -O -R501400/535400/1363500/1391500 -K
-W.4p/250/250/250 -St.4i -G0 -V >> 1cmprob_map.ps

#pstext
# labels Cerro Negro and Leon using a file, text.txt, that contains data in the following form:
#  x, y, size, angle, fontno, justify, text
#

pstext text.txt -O -R -JX  -V >> 1cmprob_map.ps


#remove the surface grid file
rm 1cmprob_surface

#display the image
gs 1cmprob_map.ps
```

The make_probmap script took the following steps to create either Figure 7a or Figure 7b from 1cm and 4cm probability grids:

1. The surface of the data is grided using surface function. Here, the file of data is called gmt_prob. This function grids the data minimum curvature algorithm and produces a grid of the surface called 1cmprob_surface.

```
surface  -I1000 gmt_prob -G1cmprob_surface -R501400/535400/1363500/1391500  -V
```

2. A post script base map for the figure is created with psbasemap. The map is given a range of from 437400 to 393400 meters on the East-West axis and 1362500 to 1402500 meters on the North-South axis. The postscript file created in this step allows for the appending of additonal postscript which will be created in the subsiquent steps.

```
psbasemap -JX5.0iY4.12i -Ba10000f1000/WSne -R501400m/535400m/1363500m/1391500m
-X1i  -P -K -V   > 1cmprob_map.ps
```

3.The grdcontour function draws the contours of the grid produced in the first step. The contours are appended to the base map.

grdcontour 1cmprob_surface -JX5.0iY4.12i -C5 -A20f13 -L1/100 -W1p -P -K -V -O >> 1cmprob_map.ps

4.The outline of the city of Leon is drawn with psxy. A file leon_outline.txt, which contains xy points that outline the city is used by the psxy function to draw the outline.

(First part of leon_outline.txt)

510731.062500 1374408.125000
511021.468750 1374862.500000
511328.375000 1374892.750000
511401.562500 1375354.750000
511596.093750 1375400.125000
511611.812500 1375195.750000
511813.718750 1375271.500000
511783.343750 1375385.000000
512059.968750 1375506.250000
512129.625000 1374885.250000
512540.968750 1375021.625000
512582.968750 1375831.875000
513106.281250 1376066.750000
513380.937500 1376725.625000
513680.562500 1376695.375000
513676.718750 1375695.750000
514059.656250 1375408.000000
514441.406250 1375453.500000
514448.218750 1375635.250000
514568.000000 1375642.750000
514559.531250 1375915.375000
514709.125000 1375960.875000
514823.281250 1375453.500000
514817.437500 1374999.125000
514722.718750 1374279.625000
514575.750000 1373514.875000
514320.812500 1373605.625000

514215.687500 1373681.375000
513968.187500 1373795.000000
513917.593750 1373287.625000

psxy leon_outline.txt -JX5.0iY4.12i -R501400/535400/1363500/1391500 -O -G230 -K -W.25p -V >> 1cmprob_map.ps

5.The psxy function is used again here to draw a triangle to mark the position of Cerro Negro. The cordinates of Cerro Negro are given in the cerronegro.txt file.

psxy cerronegro.txt -JX5.0iY4.12i -O -R501400/535400/1363500/1391500 -K -W.4p/250/250/250 -St.4i -G0 -V >> 1cmprob_map.ps

6.The figure is finally labeled with pstext. A file text.txt is used by the function and contains in addition to the text, the text's size justification, and position. After this post script is appended, the ps file of the figure is complete.

527500 1386700 18 0 21 TL Cerro Negro
511000 1372400 18 0 21 TL Leon

pstext text.txt -O -R -JX -V >> 1cmprob_map.ps

Make Grid

Figures 7a and 7b plot the probability of 1cm and 4cm tephra accumulation within a 35km area. These figures were created from a grid containg the mass of tephra deposited at each grid point. The original data used in (Figure 7a and Figure 7b)was obtained through tephragrid.c.

Program Name: ASH_PLUME

Class Name: tephragrid

Date: 07/05/00

Release Version: 1.0

Client Name: USNRC

U. S. Nuclear Regulatory Commission

Scientific Notebook 115E
20-1402-461
Inititals: CC

Chuck Connor
April 1, 2000

NRC Office of Nuclear Material Safety and Safeguard
Division of Waste Management

NRC Contract: NRC 20-01402-461

NRC Contact: Dr. John Trapp (301) 415-8063

CNWRA Contact: Dr. Charles Connor (210) 522-6649

Center For Nuclear Waste Regulatory Analyses
Southwest Research Institute
6220 Culebra Rd.
San Antonio, TX, 78238-5166, USA
cconnor@swri.edu

Documentation:
ASH_PLUME version 1.0 - Probabilistic Volcanic
Hazard Assessment Methods for a Proposed
High-Level Radioactive Waste Repository
at Yucca Mountain, Nevada

CNWRA 2000-02

NUREG-Series Designator: N/A

****

****
****
DISCLAIMER

****
****

"This computer code / material was prepared as an account of work performed by the Center for Nuclear Waste Regulatory Analyses (CNWRA) for the Division of Waste Management of the Nuclear Regulatory Commission (NRC), an independent agency of the United States Government. Neither the developer(s) of the code nor any of their sponsors make any

warranty, expressed or implied, or assume any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represent that its use would not infringe on privately-owned rights."

"In no event unless required by applicable law will the sponsors or those who have written or modified this code, be liable for damages, including any loss of profits, lost monies, or other special, incidental or consequential damages arising out of the use or inability to use the program (including but not limited to loss of data or data rendered inaccurate or losses sustained by third parties for failure of the program to operate with other programs), even if you have been advised of the possibility of such damages or for any claim by any other party."

****
****
by C. Connor, Nathan Frankln, and Brandi Winfrey

****
**********/

```
/* ---------------------------------------------------------------
 *
 * abstract:  estimation of tephra accumulation at a point
 *         using the method developed by Suzuki (1983)
 *         and integration using the  trapezoid rule
 *
 * revised:   18 Feb 2000 new create by:
 *
 *         Brandi Winfrey / #20
 *         Southwest Research Institute
 *         6220 Culebra Rd
 *         San Antonio TX 78238 USA
 *         e-mail: b_winfrey@hotmail.com
 *         210-522-4667 (voice)
 *         210-522-5155 (fax)
 *
 * revised:   Mar 1 2000
 *
 *         -Make a grid
```

```
*           Nathan Franklin
*
*
* Input looks like:
*
*f1,f2,f3,f4,f5,f6,f7,f8,f9,f10,f11,f12
*6.1E11, -5.0, 5.0, -1.3, 1.0, 1.2, 0.5, 0.5, 3152.7, 4.7,1425.0, 0.1
*
* where:
*  f1 = total_ash_mass: the total amount of material
*                  erupted in gm
*  f2 = ymin: the minimum particle size of tephra erupted
*           in phi units (e.g., -5.0)
*  f3 = ymax: the maximum  particle size of tephra erupted
*           in phi units (e.g., 5.0)
*  f4 = dmean: the mean particle size of tephra erupted
*           in phi units (e.g., -1.0)
*  f5 = dsigma: the standard deviation  particle size of tephra erupted
*            in phi units (e.g., 1.0)
*  f6 = phiash: the clast density of individual pyroclasts
*             assumed to be independent of gransize
*  f7 = dfshape: the particle shape factor,
*             assumed to be independent of gransize
*  f8 = dbeta: the empirical factor controlling plume geometry
*  f9 = w0: the eruption velocity in cm/s
*  f10 = h: the eruption column height in km
*  f11 = u: the average wind speed in cm/s
*  f12 = udir: the wind direction in +/- radians
*           wrt the x-axis
*
* ---------------------------------------------------------------
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <time.h>                    //to check program's time
```

```
#define PI 3.141592654

//nf: string length of a line from input data file
int ary_size=200;

// following are the 12 input variables
double total_ash_mass, ymin, ymax, dmean, dsigma, phiash;
double dfshape, dbeta, w0, h, u, udir;

//following are assumed const.
double c      = 400.0;       // eddy diffusivity in the atmosphere
double phiair =  0.001293;   // density of air
double nua    =  0.00018;    // dynamic viscosity of air
double gravity = 980.0;      // gravity


//nf: moved into main
//char variables[300];   /* -- store off the auto stack -- */

//define the following functions
double phi2cm( double xx );
double w( double zspace );
double p( double zspace, double xrho );
double ts( double zspace );
double t( double zspace, double xrho );
double v0( double xrho );
double frho( double xrho );
double f( double xspace, double yspace, double zspace, double xrho, int kount );


/* ---------------------- Main Starts Here ------------------------- */
main(argc, argv)
int argc;
char *argv[ ];
{
  FILE *infile, *outfile;

    //to evaluate program speed
    // clock_t start, end;
```

```
   // double elapsed;


int xsteps, ysteps;
int xxi, yyi, x_coord, y_coord;

//nf:  don't use T[] so this just keeps track of where one is in
//    the set of data.  ri =data_line#  * grid#.  (probably don't need)
int ri = 0; // used to increment T[]

//instead of storing data: just output it
//double T[ ary_size ];   // array in which to store the summation
                  // enter the size of the array on the
                  // command line

char *totalmass, *min,    *max,  *mean,     *sigma;
char *density,   *fshape, *beta, *velocity, *height;
char *windspeed, *winddir;
//nf: moved from outside main
char variables[ary_size];

double xspace, yspace, zspace, xrho, new_xspace, new_yspace;
double xsectionwidth, ysectionwidth;
double xstepwidth, ystepwidth;
double x0, y0, x1, xi, y1, yi,ans;
double total;
double xmax, xmin;

double xmax_grid=40e5;
double xmin_grid=-5e5;
double ymax_grid=20e5;
double ymin_grid=-20e5;
double spacing_grid=2.0e5;

//nf: record starting time
//start= clock();

//nf:  do the following over predifined grid
//      -using two for loops for x and y
```

```
//      distances from the vent.


// do the integration over the eruption
// column height in 300 steps
xsteps = 300;

// do the intergration over the particle
// size reange in 100 steps
ysteps = 100;

// initialize the summation for the
// contribution of each d phi and
// each dz to the total accumulation
// of tephra on the ground at the
// point xspace, yspace
total  = 0.0;



// standard usage io
if ( argc != 2 ) {
  fprintf ( stderr, "Usage: %s <file.in>\n", argv[0] );
  exit (0);
}
if ( ( infile = fopen (argv[1],"r") ) == NULL ) {
  fprintf ( stderr, "Unable to open %s for input.\n", argv[1]);
  exit(0);
}

//------------------GRID------------
// nf:  do the following over predifined grid
//      -using two for loops for x and y
//       distances from the vent.
xspace=xmin_grid;
while(xspace <= xmax_grid)
{  // traversing along xaxis
    yspace=ymin_grid;
    while(yspace <= ymax_grid)
```

```
{  // traversing along yaxis


// read the input file while the file still contains new data
if ( ( infile = fopen (argv[1],"r") ) == NULL ) {
   fprintf ( stderr, "Unable to open %s for input.\n", argv[1]);
    exit(0);
  }
while( fgets( variables, ary_size, infile) != NULL) {
 //nf:  switched "300" to
 // if (fgets( variables, ary_size, infile) == NULL ) {
  /*  get a line */
  //   fprintf ( stderr, "End of File for: %s.\n", argv[1] );
   //  close (infile);
  //}

    if ( strncmp(&variables[0],"#",1) != 0 ) {
      /* parse the line into seperate variables */
      totalmass = strtok ( variables, ",\r\n");
           min = strtok (      NULL, ",");
           max = strtok (      NULL, ",");
          mean = strtok (      NULL, ",");
         sigma = strtok (      NULL, ",");
       density = strtok (      NULL, ",");
        fshape = strtok (      NULL, ",");
          beta = strtok (      NULL, ",");
      velocity = strtok (      NULL, ",");
        height = strtok (      NULL, ",");
     windspeed = strtok (      NULL, ",");
       winddir = strtok (      NULL, ",");

      /* convert the variable strings to floats */
      total_ash_mass = atof( totalmass );
              ymin = atof( min       );
              ymax = atof( max       );
             dmean = atof( mean      );
            dsigma = atof( sigma     );
            phiash = atof( density   );
           dfshape = atof( fshape    );
```

```
            dbeta = atof( beta      );
              w0 = atof( velocity  );
               h = atof( height    );
               u = atof( windspeed );
            udir = atof( winddir   );
    }
/* do the calculations here ------------------------------------- */

    // first define the limits of integration
    // xmin is the elevation of the vent
    // xmax is the total hight of the eruption column
    // ymin and ymax are minimum and maximum
    // grainsizes, respectively

    xmin = 0.0;
    xmax = h;
    xsectionwidth = xmax - xmin;
    ysectionwidth = ymax - ymin;
    xstepwidth = xsectionwidth / xsteps;
    ystepwidth = ysectionwidth / ysteps;

    // for the calculation, define the coordinate
    // system wrt wind direction
    // here we are just assuming that
    // the wind is varying N or south of
    // a western direction, need to be genrealized
    new_xspace = xspace * cos(udir) + yspace * sin(udir);
    new_yspace = -xspace * sin(udir) + yspace * cos(udir);

    // do the integration
    for (yyi = 1; yyi < ysteps; yyi++) {

     // define the small slice d phi
     yi = (double)yyi;
     y0 = ymin + yi * ystepwidth;
     y1 = ymin + (yi+(double)1.0) * ystepwidth;

     for (xxi = 1; xxi < xsteps; xxi ++) {
```

```
            // define the small slice dz
            xi = (double)xxi;
            x0 = xmin + (xi) * xstepwidth;
            x1 = xmin + (xi+(double)1.0) * xstepwidth;

            // calculate the value of the ash accumulation  at
            // the corners  of the slice, and sum
            ans = f(new_xspace, new_yspace, x0, y0, ri) + f(new_xspace, new_yspace, x1, y0, ri) +
              f(new_xspace, new_yspace, x0, y1, ri) + f(new_xspace, new_yspace, x1, y1, ri);
            total += ans;
          }
        }

        // complete the integration by multiplying by dz and d phi
        // and dividing by 4, since 4 corners were used
        total *= xstepwidth * ystepwidth / 4.0;

        // print the result for the original coordinates
        // fprintf(outfile, "%f %f %f\n",xspace,yspace, total);

        // store the result in an array
       // T[ri] = total;
       //nf: printint to output instead of file
        printf("%f  %f  %f\n",xspace,yspace,total);
       // fprintf(outfile, "%f\n", T[ri]);
        ri++;
  }
   close(infile);
    yspace=yspace+spacing_grid;
    }//end of while(yspace<=ymax_grd)
    xspace=xspace+spacing_grid;
    //end=clock();
    //elapsed=((double) (end - start)) /CLOCKS_PER_SEC;
    //printf("\n# time elapsed %f secs\n",elapsed);


  }//end of while(xspace<=xmax_grd)
}
```

```
/* ----------------- New Function Starts Here ---------------------- */
/* function phi2cm converts the ash diameter from
   units of phi to cm
*/
double phi2cm(double xx) {
  double cms;

  cms = 0.1 * 1.0 / pow(2, xx);
  return cms;
}


/* ----------------- New Function Starts Here ---------------------- */
/* this function estimates the velocity
  as a function of height in the
  eruption column, given a
  linear decrease in velocity with
  height and an initial velocity
  of w0
*/

/* returns the upward particle velocity  */

double w(double zspace) {
  double particle_velocity;

  particle_velocity = w0 * (1.0 - zspace/h);

  // following only ocurrs as round-off error
 if (zspace > h) particle_velocity = 0.0;
  return particle_velocity;
}


/* ----------------- New Function Starts Here ---------------------- */
/* this function calculates the probability
  density for diffusion of ash out of the
  eruption column. Some variation from Suzuki
  implemented to conserve mass
```

```
  depends on beta, w0, w(z), v0(xrho)
  where w0 is the eruption velocity in cm/s
  w(z) in the upward particle velocity at height z (cm/s)
  v0(xrho) is the particle settling velocity (cm/s)
  h is the column height (km)

  integrating fz from 0 to h should give unity

*/

double p(double zspace, double xrho) {
  double y, y0;
  double prob_dens_func;
  double voo;
  double demon1, demon2;

  voo = v0(xrho);
  y  = dbeta * w(zspace) / voo;
  y0 = dbeta * w0 / voo;
  demon1 = dbeta * w0 * y * exp(-y);
  demon2 = voo*h * (1.0-(1.0+y0) * exp(-y0));
  prob_dens_func = demon1/demon2;

  return prob_dens_func;
}



/* ---------- New Function Starts Here ---------- */
/* function ts determines the particle diffusion time const
  in the atmosphere
  where zspace = height of the particle, converted here from
            km to cm, this should be wrt sea-level,
            zspace is given here in cm because
            of the units of eddy diffusivity
      c = eddy diffusivity in the atmosphere
        given as in cgs, e.g., 400 cm/s^(5/2)
```

```
  returns the particle diffusion time
*/

double ts (double zspace) {
  double particle_diffusion_time;
  double demon1, demon2;

  zspace *= 1.0e+5;
  demon1  = 5.0 * zspace * zspace;
  demon2  = 288.0 * c;
  particle_diffusion_time = pow(demon1/demon2, 2.0/5.0);

  return particle_diffusion_time;
}


/* ----------------- New Function Starts Here ---------------------- */
/* function t calculates the particle fall time
  in the atmposphere
  where: zspace = height in the atmosphere, here in km
               wrt to sealevel, because the const.
               -0.0625 is in units of 1/km
*/

double t (double zspace, double xrho) {
  double particle_fall_time;
  double demon1, demon2;

  demon1 = (1.0 - exp(-0.0625*zspace)) / v0(xrho);
  demon2 = pow(demon1, 0.926);
  particle_fall_time = 0.752e6 * demon2;
  return particle_fall_time;
}


/* ----------------- New Function Starts Here ---------------------- */
/* return a value for the particle terminal velocity
  at sea level, given the particle diameter, passed as x
  note that the shape factor does not depend on x
```

```
  note that the ash dens does not depend on x
*/

double v0 (double xrho) {

  double velocity0;
  double demon1, demon2, demon3;

  demon1 = 9.0 * nua * pow(dfshape, -0.32);
  demon2 = 81.0 * nua * nua * pow(dfshape, -0.64);
  demon3 = 1.5 * phiair * phiash * gravity * xrho * xrho * xrho *
        sqrt(1.07-dfshape);

  velocity0 = phiash * gravity * xrho * xrho /
          (demon1 + sqrt(demon2 + demon3));

  return velocity0;
}



/* ----------------- New Function Starts Here ---------------------- */
/* this function calculates the expected fraction of particles
  in a given grainsize class (xrho) assuming a normal
  distribution in phi units about the mean, dmean,
  with standard deviation dsigma.
*/

double frho(double xrho) {
  double func_rho;
  double demon1, demon3, demon2;

  demon1   = 1.0 / (2.506628 * dsigma);
  demon3   = xrho - dmean;
  demon2   = exp(-demon3*demon3 / (2.0*dsigma*dsigma));
  func_rho = demon1 * demon2;

  if (func_rho >= 0.0) {
     return func_rho;
```

```
  }
  else {
    printf("error in ash size distribution - method frho");
    return func_rho;
  }
}



/* ---------- New Function Starts Here ---------- */
/* function f calculates the epected mass of ash to accumulate in a
given location xspace, yspace (cm from the volcano) in gm / cm^3
*/

double f(double xspace, double yspace, double zspace, double xrho, int kount) {
  double ash_fall;
  double ashdiam;
  double t1, t2;
  double demon1, demon2, demon3, demon4;

  ashdiam = phi2cm(xrho);
  t1 = t(zspace, ashdiam);
  t2 = ts(zspace);
  demon1 =  5.0 * total_ash_mass * p(zspace,ashdiam) * frho(xrho);
  demon2 =  8.0 * PI * c * pow((t1+t2), 5.0/2.0);
  demon3 = -5.0 * ((xspace-u*t1) * (xspace-u*t1) + yspace * yspace);
  demon4 =  8.0 * c * pow(t1+t2, 5.0/2.0);

  ash_fall = demon1 / demon2 * exp(demon3/demon4);

  if (ash_fall >= 0.0) {

    return ash_fall;
  }
  else {
    printf("h: %f, zspace: %f, ashdiam: %f, kounter: %i\n",h, zspace,ashdiam,kount);
//    printf("p: %f, ahsdiam: %f, zspace: %f\n",p(zspace,ashdiam), ashdiam, zspace );
    return ash_fall;
    }
```

}

An alternate method for calculating the mass of tephra deposited at each point on the grid is the makegrid code which follows.

1. The makefile makegrid1 is a C-shell script which contains calls to three other codes that do the calculations. The command to initiate makegrid1 is:

   makegrid1

```
#!/usr/bin/csh

# ------------------------------------------------------------
#
#  abstract:  calculate suzuki 100 times for each of 100
#             points on a grid
#
#  revised:   16-Mar-2000, new create by:
#
#                Brandi Winfrey / #20
#                6220 culebra road
#                san antonio tx 78238 usa
#                E-mail: b_winfrey@hotmail.com
#                Phone: 210-522-4667
#
# ------------------------------------------------------------
#
# the following commands are to the c shell
#
echo Reading the file....
set xspace = 2000000
set yspace = 0
set row = 0
set col = 0
```

```
# the xspace value is associated with columns
while ( $col < 10 )
    echo 1st while....
    # the yspace value is associated with rows
    while ( $row < 10 )
        echo 2nd while....
        # call suzuki to get 100 pts for (Xspace, Yspace)
        echo step2....
        step2 data3.in junk 49 "$xspace" "$yspace"
        cat junk >> step2.out

        # convert from grams to centimeters AND
        # sort the columns in ascending order
        echo sort....
        awk -f sort_cols.awk junk > junk2
        cat junk2 >> sort_cols.out

        # calculate the percentage for point (Xspace, Yspace)
        # append the result to the end of junk3
        echo percent....
        awk -f percent.awk junk2 xx="$xspace" yy="$yspace" >> percent.out

        # increment variables
        @ yspace += 100000
        @ row++
    end
    set row = 0
    set yspace = 0
    @ xspace += 100000
    @ col++
end
```

2.The step2.c code is the main code. It actually performs the ashfall calculation. Step2.c creates a text file called step2.out that contains the ashfall mass, in grams, one calculation per input line. To run the code seperately from the makefile, type the following on the command line:

step2 <infile> <outfile> <number of lines in infile> <xspace> <yspace>

Scientific Notebook 115E
20-1402-461
Inititals: CC

Chuck Connor
April 1, 2000

```
/**

**************************************************************************
```

Program Name: ASH_PLUME

Class Name: step2

Date: 07/05/00

Release Version: 1.0

Client Name: USNRC

U. S. Nuclear Regulatory Commission
NRC Office of Nuclear Material Safety and Safeguard
Division of Waste Management

NRC Contract: NRC 02-97-009

NRC Contact: Dr. John Trapp (301) 415-8063

CNWRA Contact: Dr. Charles Connor (210) 522-6649

Center For Nuclear Waste Regulatory Analyses
Southwest Research Institute
6220 Culebra Rd.
San Antonio, TX, 78238-5166, USA
cconnor@swri.edu

Documentation:
ASH_PLUME version 1.0 - Probabilistic Volcanic
Hazard Assessment Methods for a Proposed
High-Level Radioactive Waste Repository
at Yucca Mountain, Nevada

CNWRA 2000-02

NUREG-Series Designator: N/A

Scientific Notebook 115E
20-1402-461
Inititals: CC

Chuck Connor
April 1, 2000

****

****
****
DISCLAIMER

****
****

"This computer code / material was prepared as an account of work performed by the Center for Nuclear Waste Regulatory Analyses (CNWRA) for the Division of Waste Management of the Nuclear Regulatory Commission (NRC), an independent agency of the United States Government. Neither the developer(s) of the code nor any of their sponsors make any warranty, expressed or implied, or assume any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represent that its use would not infringe on privately-owned rights."

"In no event unless required by applicable law will the sponsors or those who have written or modified this code, be liable for damages, including any loss of profits, lost monies, or other special, incidental or consequential damages arising out of the use or inability to use the program (including but not limited to loss of data or data rendered inaccurate or losses sustained by third parties for failure of the program to operate with other programs), even if you have been advised of the possibility of such damages or for any claim by any other party."

****
****
by C. Connor and Brandi Winfrey

****
**********/

```
/* ------------------------------------------------------------
 *
 * abstract:  estimation of tephra accumulation at a point
 *          using the method developed by Suzuki (1983)
 *          and integration using the  trapezoid rule
```

```
*
* revised:  18 Feb 2000 new create by:
*
*         Brandi Winfrey / #20
*         Southwest Research Institute
*         6220 Culebra Rd
*         San Antonio TX 78238 USA
*         e-mail: b_winfrey@hotmail.com
*         210-522-4667 (voice)
*         210-522-5155 (fax)
*
* Input looks like:
*
*f1,f2,f3,f4,f5,f6,f7,f8,f9,f10,f11,f12
*6.1E11, -5.0, 5.0, -1.3, 1.0, 1.2, 0.5, 0.5, 3152.7, 4.7,1425.0, 0.1
*
* where:
*  f1 = total_ash_mass: the total amount of material
*                  erupted in gm
*  f2 = ymin: the minimum particle size of tephra erupted
*           in phi units (e.g., -5.0)
*  f3 = ymax: the maximum  particle size of tephra erupted
*           in phi units (e.g., 5.0)
*  f4 = dmean: the mean particle size of tephra erupted
*           in phi units (e.g., -1.0)
*  f5 = dsigma: the standard deviation  particle size of tephra erupted
*           in phi units (e.g., 1.0)
*  f6 = phiash: the clast density of individual pyroclasts
*           assumed to be independent of gransize
*  f7 = dfshape: the particle shape factor,
*           assumed to be independent of gransize
*  f8 = dbeta: the empirical factor controlling plume geometry
*  f9 = w0: the eruption velocity in cm/s
*  f10 = h: the eruption column height in km
*  f11 = u: the average wind speed in cm/s
*  f12 = udir: the wind direction in +/- radians
*           wrt the x-axis
*
* ---------------------------------------------------------------------
```

Scientific Notebook 115E
20-1402-461
Inititals: CC

Chuck Connor
April 1, 2000

*/

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

#define PI 3.141592654

// following are the 12 input variables
double total_ash_mass, ymin, ymax, dmean, dsigma, phiash;
double dfshape, dbeta, w0, h, u, udir;

//following are assumed const.
double c      = 400.0;      // eddy diffusivity in the atmosphere
double phiair =  0.001293;  // density of air
double nua    =  0.00018;   // dynamic viscosity of air
double gravity = 980.0;     // gravity


char variables[300];   /* -- store off the auto stack -- */

//define the following functions
double phi2cm( double xx );
double w( double zspace );
double p( double zspace, double xrho );
double ts( double zspace );
double t( double zspace, double xrho );
double v0( double xrho );
double frho( double xrho );
double f( double xspace, double yspace, double zspace, double xrho, int kount );

/* ---------------------- Main Starts Here ------------------------- */
main(argc, argv)
int argc;
char *argv[ ];
{
  FILE *infile, *outfile;
```

```
int xsteps, ysteps;
int xi, yi, x_coord, y_coord;

int ary_size = atoi(argv[3]);
int ri = 0; // used to increment T[]
double T[ ary_size ];  // array in which to store the summation
                       // enter the size of the array on the
                       // command line
char *totalmass, *min,    *max,  *mean,    *sigma;
char *density,  *fshape,  *beta, *velocity, *height;
char *windspeed, *winddir;

double zspace, xrho, new_xspace, new_yspace;
double xsectionwidth, ysectionwidth;
double xstepwidth, ystepwidth;
double x0, y0, x1, y1, ans;
double total;
double xmax, xmin;

double xspace = atof(argv[4]);
double yspace = atof(argv[5]);
      fprintf ( stderr, "xspace: %f\n", xspace );
      fprintf ( stderr, "yspace: %f\n", yspace );

// do the integration over the eruption
// column height in 300 steps
xsteps = 300;

// do the intergration over the particle
// size reange in 100 steps
ysteps = 100;

// initialize the summation for the
// contribution of each d phi and
// each dz to the total accumulation
// of tephra on the ground at the
// point xspace, yspace
total  = 0.0;
```

```
//define the distance from the volcano
// in cm
//  xspace = 20.0e5;
//  yspace = 0.0e5;


// standard usage io
if ( argc != 6 ) {
  fprintf ( stderr, "Usage: %s <file.in> <file.out> <input file size> <xspace> <yspace>\n",
argv[0] );
  exit (0);
}
if ( ( infile = fopen (argv[1],"r") ) == NULL ) {
  fprintf ( stderr, "Unable to open %s for input.\n", argv[1]);
  exit(0);
}
else if ( (outfile = fopen (argv[2], "w")) == NULL) {
  fprintf ( stderr, "Unable to open %s for writing.\n", argv[2]);
}


// read the input file while the file still contains new data
do {
    if (fgets( variables, 300, infile) == NULL ) {
    /*  get a line */
     // fprintf ( stderr, "End of File for: %s.\n", argv[1] );
      close (infile);
      exit(0);
    }

      if ( strncmp(&variables[0],"#",1) != 0 ) {
        /* parse the line into seperate variables */
        totalmass = strtok ( variables, ",\r\n");
            min = strtok (    NULL, ",");
            max = strtok (    NULL, ",");
           mean = strtok (    NULL, ",");
          sigma = strtok (    NULL, ",");
        density = strtok (    NULL, ",");
         fshape = strtok (    NULL, ",");
```

```
            beta = strtok (     NULL, ",");
        velocity = strtok (     NULL, ",");
          height = strtok (     NULL, ",");
       windspeed = strtok (     NULL, ",");
         winddir = strtok (     NULL, ",");

       /* convert the variable strings to floats */
       total_ash_mass = atof( totalmass );
              ymin = atof( min      );
              ymax = atof( max      );
             dmean = atof( mean     );
            dsigma = atof( sigma    );
            phiash = atof( density  );
           dfshape = atof( fshape   );
             dbeta = atof( beta     );
               w0 = atof( velocity );
                h = atof( height   );
                u = atof( windspeed );
             udir = atof( winddir  );
    }
/* do the calculations here ------------------------------------- */

    // first define the limits of integration
    // xmin is the elevation of the vent
    // xmax is the total hight of the eruption column
    // ymin and ymax are minimum and maximum
    // grainsizes, respectively

    xmin = 0.0;
    xmax = h;
    xsectionwidth = xmax - xmin;
    ysectionwidth = ymax - ymin;
    xstepwidth = xsectionwidth / xsteps;
    ystepwidth = ysectionwidth / ysteps;

    // for the calculation, define the coordinate
    // system wrt wind direction
    // here we are just assuming that
    // the wind is varying N or south of
```

```
        // a western direction, need to be genrealized
        new_xspace = xspace * cos(udir) + yspace * sin(udir);
        new_yspace = -xspace * sin(udir) + yspace * cos(udir);

        // do the integration
        for (yi = 1; yi < ysteps; yi++) {

         // define the small slice d phi
         y0 = ymin + yi * ystepwidth;
         y1 = ymin + ( yi + (double)1.0 ) * ystepwidth;

         for (xi = 1; xi < xsteps; xi ++) {

          // define the small slice dz
          x0 = xmin + (xi) * xstepwidth;
          x1 = xmin + ( xi + (double)1.0 ) * xstepwidth;

           // calculate the value of the ash accumulation  at
          // the corners  of the slice, and sum
          ans = f(new_xspace, new_yspace, x0, y0, ri) + f(new_xspace, new_yspace, x1, y0, ri) +
            f(new_xspace, new_yspace, x0, y1, ri) + f(new_xspace, new_yspace, x1, y1, ri);
          total += ans;
         }
        }

        // complete the integration by multiplying by dz and d phi
        // and dividing by 4, since 4 corners were used
        total *= xstepwidth * ystepwidth / 4.0;

        // print the result for the original coordinates
        // fprintf(outfile, "%f %f %f\n",xspace,yspace, total);

        // store the result in an array
        T[ri] = total;
        fprintf(outfile, "%f\n", T[ri]);
        ri++;
  } while (1);
}
```

```
/* ----------------- New Function Starts Here -------------------- */
/* function phi2cm converts the ash diameter from
   units of phi to cm
*/
double phi2cm(double xx) {
  double cms;

  cms = 0.1 * 1.0 / pow(2, xx);
  return cms;
}


/* ----------------- New Function Starts Here -------------------- */
/* this function estimates the velocity
  as a function of height in the
  eruption column, given a
  linear decrease in velocity with
  height and an initial velocity
  of w0
*/

/* returns the upward particle velocity  */

double w(double zspace) {
  double particle_velocity;

  particle_velocity = w0 * (1.0 - zspace/h);

  // following only ocurrs as round-off error
  if (zspace > h) particle_velocity = 0.0;
  return particle_velocity;
}


/* ----------------- New Function Starts Here -------------------- */
/* this function calculates the probability
  density for diffusion of ash out of the
  eruption column. Some variation from Suzuki
  implemented to conserve mass
```

```
  depends on beta, w0, w(z), v0(xrho)
  where w0 is the eruption velocity in cm/s
  w(z) in the upward particle velocity at height z (cm/s)
  v0(xrho) is the particle settling velocity (cm/s)
  h is the column height (km)

  integrating fz from 0 to h should give unity

*/

double p(double zspace, double xrho) {
  double y, y0;
  double prob_dens_func;
  double voo;
  double demon1, demon2;

  voo = v0(xrho);
  y  = dbeta * w(zspace) / voo;
  y0 = dbeta * w0 / voo;
  demon1 = dbeta * w0 * y * exp(-y);
  demon2 = voo*h * (1.0-(1.0+y0) * exp(-y0));
  prob_dens_func = demon1/demon2;

  return prob_dens_func;
}



/* ----------------- New Function Starts Here ---------------------- */
/* function ts determines the particle diffusion time const
  in the atmosphere
  where zspace = height of the particle, converted here from
              km to cm, this should be wrt sea-level,
              zspace is given here in cm because
              of the units of eddy diffusivity
       c = eddy diffusivity in the atmosphere
          given as in cgs, e.g., 400 cm/s^(5/2)
```

```
  returns the particle diffusion time
*/

double ts (double zspace) {
  double particle_diffusion_time;
  double demon1, demon2;

  zspace *= 1.0e+5;
  demon1  = 5.0 * zspace * zspace;
  demon2  = 288.0 * c;
  particle_diffusion_time = pow(demon1/demon2, 2.0/5.0);

  return particle_diffusion_time;
}


/* ----------------- New Function Starts Here ---------------------- */
/* function t calculates the particle fall time
  in the atmposphere
  where: zspace = height in the atmosphere, here in km
               wrt to sealevel, because the const.
               -0.0625 is in units of 1/km
*/

double t (double zspace, double xrho) {
  double particle_fall_time;
  double demon1, demon2;

  demon1 = (1.0 - exp(-0.0625*zspace)) / v0(xrho);
  demon2 = pow(demon1, 0.926);
  particle_fall_time = 0.752e6 * demon2;
  return particle_fall_time;
}


/* ----------------- New Function Starts Here ---------------------- */
/* return a value for the particle terminal velocity
  at sea level, given the particle diameter, passed as x
  note that the shape factor does not depend on x
```

```
  note that the ash dens does not depend on x
*/

double v0 (double xrho) {

  double velocity0;
  double demon1, demon2, demon3;

  demon1 = 9.0 * nua * pow(dfshape, -0.32);
  demon2 = 81.0 * nua * nua * pow(dfshape, -0.64);
  demon3 = 1.5 * phiair * phiash * gravity * xrho * xrho * xrho *
       sqrt(1.07-dfshape);

  velocity0 = phiash * gravity * xrho * xrho /
         (demon1 + sqrt(demon2 + demon3));

  return velocity0;
}



/* ---------- New Function Starts Here ---------- */
/* this function calculates the expected fraction of particles
  in a given grainsize class (xrho) assuming a normal
  distribution in phi units about the mean, dmean,
  with standard deviation dsigma.
*/

double frho(double xrho) {
  double func_rho;
  double demon1, demon3, demon2;

  demon1  = 1.0 / (2.506628 * dsigma);
  demon3  = xrho - dmean;
  demon2  = exp(-demon3*demon3 / (2.0*dsigma*dsigma));
  func_rho = demon1 * demon2;

  if (func_rho >= 0.0) {
    return func_rho;
```

```
  }
  else {
    printf("error in ash size distribution - method frho");
    return func_rho;
  }
}


/* ---------- New Function Starts Here ---------- */
/* function f calculates the epected mass of ash to accumulate in a
given location xspace, yspace (cm from the volcano) in gm / cm^3
*/

double f(double xspace, double yspace, double zspace, double xrho, int kount) {
  double ash_fall;
  double ashdiam;
  double t1, t2;
  double demon1, demon2, demon3, demon4;

  ashdiam = phi2cm(xrho);
  t1 = t(zspace, ashdiam);
  t2 = ts(zspace);
  demon1 =  5.0 * total_ash_mass * p(zspace,ashdiam) * frho(xrho);
  demon2 =  8.0 * PI * c * pow((t1+t2), 5.0/2.0);
  demon3 = -5.0 * ((xspace-u*t1) * (xspace-u*t1) + yspace * yspace);
  demon4 =  8.0 * c * pow(t1+t2, 5.0/2.0);

  ash_fall = demon1 / demon2 * exp(demon3/demon4);

  if (ash_fall >= 0.0) {
    return ash_fall;
  }
  else {
    printf("demon1: %f, demon2: %f, kounter: %i\n",demon1, demon2, kount);
    return ash_fall;
    }
}
```

(First part of step2.out)

1.802803
2.582450
13.399405
23.884684
2.009549
0.778778
0.651823
15.360462
3.802757
14.782924
7.486856
0.708464
2.851208
1.331543
6.580137
0.616301
1.964228
4.791106
0.843005
6.330504
2.143994
0.880820

3. The next code in the sequence of events is the sort_cols.awk script which performs two operations on the step2.out data file. It sorts the data into ascending order, and converts the units from grams to centimeters. The output file from this script is called sort_cols.out. sort_cols.out contains the ashfall mass in centimeters. To run the script seperately from the makefile, type the following on the command line:

awk -f sort_cols.awk <infile> > <outfile>

```
# ------------------------------------------------------------
#
#  abstract:  sorts columns in ascending order
#
#  revised:   16-Mar-2000, new create by:
```

```
#
#               Brandi Winfrey / #20
#               6220 culebra road
#               san antonio tx 78238 usa
#               E-mail: b_winfrey@hotmail.com
#               Phone: 210-522-4667
#
# ---------------------------------------------------------------

# main
{

  # read all rows and assign values of $1 to array
  # named values as well as converts from grams
  # to centimeters
    values[NR] = $1 * .9

}

# sort numbers in ascending order
function sort (ARRAY, ELEMENTS,    temp, i, j) {
  for (i=2; i<=ELEMENTS; i++) {
    for(j=i; ARRAY[j-1] > ARRAY[j]; j--) {
      temp = ARRAY[j]
      ARRAY[j] = ARRAY[j-1]
      ARRAY[j-1] = temp
    }
  }
  return
}

END{

  # call sort function to sort elements
    sort(values, NR)
    for (j=1; j<=NR; j++) {
      printf ("%f\n", values[j])
    }
```

}

(First part of sort_cols.out)

0.028948
0.042951
0.057992
0.096459
0.146090
0.184261
0.269041
0.328742
0.394219
0.403404
0.466076
0.554528
0.554671
0.586641
0.637618
0.643726
0.679577
0.700900
0.758705
0.792738
1.198389
1.334009
1.509690
1.622523
1.635321

4.The last script in the chain is the percent.awk script. It calculates the percentages of ash mass greater than 1cm and greater than 4cm in thickness. Percent.awk reads in the sort_cols.out file and writes out to percent.out. percent.out contains the X,Y coordinates (xspace and yspace) and their respective percentages. To run the script seperately from the makefile, type the following on the command line:

awk -f percent.awk <infile> xx=<xspace> yy=<yspace> > > <outfile>

```
#
#  abstract:  sorts columns in ascending order
#
#  revised:   16-Mar-2000, new create by:
#
#             Brandi Winfrey / #20
#             6220 culebra road
#             san antonio tx 78238 usa
#             E-mail: b_winfrey@hotmail.com
#             Phone: 210-522-4667
#
# ---------------------------------------------------------------

# main
{

  # read all rows and assign values of $1 to array
  # named values as well as converts from grams
  # to centimeters
    values[NR] = $1 * .9

}

# sort numbers in ascending order
function sort (ARRAY, ELEMENTS,    temp, i, j) {
  for (i=2; i<=ELEMENTS; i++) {
    for(j=i; ARRAY[j-1] > ARRAY[j]; j--) {
      temp = ARRAY[j]
      ARRAY[j] = ARRAY[j-1]
      ARRAY[j-1] = temp
    }
  }
  return
}

END{

  # call sort function to sort elements
    sort(values, NR)
```

```
    for (j=1; j<=NR; j++) {
      printf ("%f\n", values[j])
    }


}
```

(First part of percent.out)

2e+06 0 60.4167% 22.9167%
2e+06 100000 60.4167% 22.9167%
2e+06 200000 60.4167% 22.9167%
2e+06 300000 60.4167% 22.9167%
2e+06 400000 60.4167% 22.9167%
2e+06 500000 60.4167% 22.9167%
2e+06 600000 60.4167% 22.9167%
2e+06 700000 60.4167% 22.9167%
2e+06 800000 58.3333% 22.9167%
2e+06 900000 58.3333% 20.8333%
2.1e+06 0 56.25% 18.75%
2.1e+06 100000 56.25% 18.75%
2.1e+06 200000 56.25% 18.75%
2.1e+06 300000 56.25% 18.75%
2.1e+06 400000 56.25% 18.75%
2.1e+06 500000 56.25% 18.75%
2.1e+06 600000 56.25% 18.75%
2.1e+06 700000 56.25% 18.75%
2.1e+06 800000 56.25% 18.75%
2.1e+06 900000 56.25% 18.75%
2.2e+06 0 50% 12.5%
2.2e+06 100000 50% 12.5%
2.2e+06 200000 50% 12.5%
2.2e+06 300000 50% 12.5%
2.2e+06 400000 50% 12.5%
2.2e+06 500000 50% 12.5%
2.2e+06 600000 50% 10.4167%
2.2e+06 700000 50% 10.4167%
2.2e+06 800000 50% 10.4167%
2.2e+06 900000 50% 10.4167%

2.3e+06 0 39.5833% 10.4167%

Note: Thepercent.awk script will not run independently of the makefile because the xspace and yspace variables are passed in from the makefile.

The percent2.awk script performs the same function as the percent.awk script with the exception of xspace and yspace. It can be run on the command line. The output file is percent2.out

awk -f percent.awk <infile> > > <outfile>

```
# ---------------------------------------------------------------
#
#  abstract:  calculates the percentage of inputs that contain ash
#             fall at values greater than 1cm and at values greater
#             than 4cm.
#
#  revised:   16-Mar-2000, new create by:
#
#                Brandi Winfrey / #20
#                6220 culebra road
#                san antonio tx 78238 usa
#                E-mail: b_winfrey@hotmail.com
#                Phone: 210-522-4667
#
# ---------------------------------------------------------------
$1 <= 1 {
  # read all rows and increments the number of rows equal to or
  # greater than one
    total1 = NR
}

$1 <= 4 {
  # read all rows and increments the number of rows equal to or
  # greater than four
    total2 = NR
}

# calculate percentage greater than or equal to 1
```

```
function percentage (TOTAL) {

  percent = (1 - (  TOTAL - 1 ) / NR) * 100
  return
}

END{

  # call function for > 1cm calculations
    percentage(total1)
      printf ("%g\% ", percent)

  # call function for > 4cm  calculations
    percentage(total2)
      printf ("%g\%\n", percent)

}
```

(First part of percent2.out)

```
60.4167% 22.9167%
60.4167% 22.9167%
60.4167% 22.9167%
60.4167% 22.9167%
60.4167% 22.9167%
60.4167% 22.9167%
60.4167% 22.9167%
60.4167% 22.9167%
58.3333% 22.9167%
58.3333% 20.8333%
56.25% 18.75%
56.25% 18.75%
56.25% 18.75%
56.25% 18.75%
56.25% 18.75%
56.25% 18.75%
56.25% 18.75%
56.25% 18.75%
56.25% 18.75%
```

56.25% 18.75%
50% 12.5%
50% 12.5%
50% 12.5%
50% 12.5%
50% 12.5%
50% 12.5%
50% 10.4167%
50% 10.4167%
50% 10.4167%
50% 10.4167%

****

```
#!/usr/bin/csh

# ------------------------------------------------------------
#
#  abstract:  calculate suzuki 100 times for each of 100
#             points on a grid
#
#  revised:   16-Mar-2000, new create by:
#
#                Brandi Winfrey / #20
#                6220 culebra road
#                san antonio tx 78238 usa
#                E-mail: b_winfrey@hotmail.com
#                Phone: 210-522-4667
#
# ------------------------------------------------------------
#
# the following commands are to the c shell
#
echo Reading the file....
set xspace = 2000000
set yspace = 0
set row = 0
set col = 0
```

```
# the xspace value is associated with columns
while ( $col < 10 )
    echo 1st while....
    # the yspace value is associated with rows
    while ( $row < 10 )
        echo 2nd while....
        # call suzuki to get 100 pts for (Xspace, Yspace)
        echo step2....
        step2 data3.in junk 49 "$xspace" "$yspace"
        cat junk >> step2.out

        # convert from grams to centimeters AND
        # sort the columns in ascending order
        echo sort....
        awk -f sort_cols.awk junk > junk2
        cat junk2 >> sort_cols.out

        # calculate the percentage for point (Xspace, Yspace)
        # append the result to the end of junk3
        echo percent....
        awk -f percent.awk junk2 xx="$xspace" yy="$yspace" >> percent.out

        # increment variables
        @ yspace += 100000
        @ row++
    end
    set row = 0
    set yspace = 0
    @ xspace += 100000
    @ col++
end

**********************************************************************************

# ------------------------------------------------------------
#
#  abstract:  sorts columns in ascending order
#
#  revised:   16-Mar-2000, new create by:
```

```
#
#               Brandi Winfrey / #20
#               6220 culebra road
#               san antonio tx 78238 usa
#               E-mail: b_winfrey@hotmail.com
#               Phone: 210-522-4667
#
# ---------------------------------------------------------------

# main
{

  # read all rows and assign values of $1 to array
  # named values as well as converts from grams
  # to centimeters
    values[NR] = $1 * .9

}

# sort numbers in ascending order
function sort (ARRAY, ELEMENTS,    temp, i, j) {
  for (i=2; i<=ELEMENTS; i++) {
    for(j=i; ARRAY[j-1] > ARRAY[j]; j--) {
      temp = ARRAY[j]
      ARRAY[j] = ARRAY[j-1]
      ARRAY[j-1] = temp
    }
  }
  return
}

END{

  # call sort function to sort elements
    sort(values, NR)
    for (j=1; j<=NR; j++) {
      printf ("%f\n", values[j])
    }
```

```
}

**************************************************************************

# ------------------------------------------------------------
#
#  abstract:  calculates the percentages of ash mass greater than
#             1cm and greater than 4cm for a given X,Y position.
#
#  revised:   16-Mar-2000, new create by:
#
#               Brandi Winfrey / #20
#               6220 culebra road
#               san antonio tx 78238 usa
#               E-mail: b_winfrey@hotmail.com
#               Phone: 210-522-4667
#
# ------------------------------------------------------------

$1 <= 1 {
  # read all rows and increments the number of rows equal to or
  # greater than one
    total1 = NR
}

$1 <= 4 {
  # read all rows and increments the number of rows equal to or
  # greater than four
    total2 = NR
}

# calculate percentage greater than or equal to 1
function percentage (TOTAL) {

  percent = (1 - ( TOTAL - 1 ) / NR) * 100
  return
}

END{
```

```
 # call function for > 1cm calculations
   percentage(total1)

 # print xspace and yspace - passed in as variables
 # print percentage
     printf ("%g ", xx)
     printf ("%g ", yy)
     printf ("%g\% ", percent)

 # call function for > 4cm  calculations
 # print percentage
   percentage(total2)
     printf ("%g\%\n", percent)

}

*******************************************************************************

# ------------------------------------------------------------
#
#  abstract:  calculates the percentage of inputs that contain ash
#             fall at values greater than 1cm and at values greater
#             than 4cm.
#
#  revised:   16-Mar-2000, new create by:
#
#               Brandi Winfrey / #20
#               6220 culebra road
#               san antonio tx 78238 usa
#               E-mail: b_winfrey@hotmail.com
#               Phone: 210-522-4667
#
# ------------------------------------------------------------
$1 <= 1 {
  # read all rows and increments the number of rows equal to or
  # greater than one
    total1 = NR
}
```

```
$1 <= 4 {
  # read all rows and increments the number of rows equal to or
  # greater than four
    total2 = NR
}

# calculate percentage greater than or equal to 1
function percentage (TOTAL) {

  percent = (1 - (  TOTAL - 1 ) / NR) * 100
  return
}

END{

  # call function for > 1cm calculations
    percentage(total1)
      printf ("%g\% ", percent)

  # call function for > 4cm  calculations
    percentage(total2)
      printf ("%g\%\n", percent)

}
```

****

```
#!/usr/bin/awk -f
#Nathan Franklin
#March 6, 2000
#
#Purpose:  To switch a grid with with origin at Cerro Negro to
#         a grid with UTM cordinates
#
#Cerro Negro UTM
#----------------------
#432400 East
#1382500 North
#
```

Scientific Notebook 115E
20-1402-461
Inititals: CC

Chuck Connor
April 1, 2000

```
#
#Input data looks like:
#----------------------
#    * x,y in cm from origin (cerro negro).  z is probability
#   ** positive x-axis is towards the west. (needs to be flipped)
#
#
#
# -------------------------------------------------------

BEGIN{
}
{
 X=532400 - $1/100;
 Y=1382500 + $2/100;
 printf("%f %f %f\n", X, Y, $3);


}
END {
}
```

****

```
#!/usr/bin/awk -f
# for every line, rotate the values by so many degrees

{
deg_rot = 20.0*3.14159/180.0;
oldxx = $1;
oldyy = $2;
thickness = $3;
newxx = oldxx * cos(deg_rot) + oldyy * sin(deg_rot);
newyy = -1 * oldxx * sin(deg_rot) + oldyy * cos(deg_rot);
print newxx,newyy,thickness

}
```

****

Scientific Notebook 115E
20-1402-461
Inititals: CC

Chuck Connor
April 1, 2000

```
import java.awt.*;
import java.util.*;

/* ----------------------------------------------------------------
 *
 * Abstract: code returns a random number equal to or between the
 *           input parameters Lo and Hi.
 *
 * Revised: 01 March 00 09
 * New create, by: Brandi Winfrey
 *
 * ----------------------------------------------------------------*/

public class Rnum {
  public static double Rand_num(double Lo, double Hi) {
    int i=0, j=0, k=0;
    double rand = 0;

    if (Lo == Hi) {
        rand = Lo;
        return rand;
    }

    if (Lo >= 1e10 || Hi > 1e10) {
      double kk = Math.log(Lo)/Math.log(10.0);
      double mm = Math.log(Hi)/Math.log(10.0);

      Random r = new Random();

      double number = (r.nextDouble() * (mm - kk) + kk);
      double ans = Math.pow(10, number);

      rand = ans;
      return rand;
    }

    else {
      Random r = new Random();
```

```
      double num = (Math.random() * (Hi - Lo)  +Lo);

      rand = num;
      return rand;
    }
  }
}
```

****

```
/* ---------------------------------------------------------------
 *
 * Abstract:  Code sorts an array
 *
 * Revised: 01 March 00 09--> new create, by:
 *
 * Notes:  Example originally obtained from
 *         source code in
 *         text: "  " by AUTHOR,
 *         (c) DATE, PRESS,
 *         PUBLISHER(S)
 *         ISBN _____.
 *
 * ---------------------------------------------------------------*/
public class Sort {
  public static double[] mergesort(double[] data) {
    if (data.length == 1) {
      return data;
    }
    else {
      int halflen = data.length/2;

      double[] done = new double[halflen];
      double[] dtwo = new double[data.length - halflen];

      for (int i=0; i<halflen; i++)
        done[i] = data[i];

      for (int i=halflen; i<data.length; i++)
```

```
        dtwo[i-halflen] = data[i];

      done = mergesort(done); // recursive
      dtwo = mergesort(dtwo); // recursive

      double[] ans = new double[data.length];

      int pone = 0;
      int ptwo = 0;
      int pans = 0;

      while (pans < data.length) {
        if (pone < done.length) {
          if (ptwo < dtwo.length) {
            if (done[pone] < dtwo[ptwo]) {
              ans[pans] = done[pone];
              pone++;
            }
            else {
               ans[pans] = dtwo[ptwo];
               ptwo++;
            }
          }
          else {
            ans[pans] = done[pone];
            pone++;
          }
        }
        else {
          ans[pans] = dtwo[ptwo];
          ptwo++;
        }
        pans++;
      }
      return ans;
    }
  }
}
```

Scientific Notebook 115E
20-1402-461
Inititals: CC

Chuck Connor
May 20, 2000

Ash plume modeling in support of the airborne radionuclide transport KTI is numerically expensive. In order to reduce this expense, we have developed a parallel processing capability for ash dispersion codes. The following describes details of this coding effort and the set up of the PC cluster on which these codes run.

Parallel Numerical modeling of Volcanic Hazards Related to Tephra Fallout

Prepared by

Nathan M. Franklin and Brandi L. Winfrey

Abstract

The decision to build a beowulf cluster is based on the large amount of time spent processing relatively simple calculations. By running a code in parallel, time becomes less of an issue. The suzukigrid code is the test case. It calculates the percentage of ash mass of a specified depth within a specified distance. The calculations involve integration and therefore, eat up a lot of time.

When parallelized, the suzuki code performs under the master-slave model, where the slave processes are spawned on each of the nodes (as needed). The nodes do not communicate with each other to perform calculation(s). The master simply allocates a task to each node which then computes and stores its results in a known location.

PVM is installed on only one machine. The others all have links to the directory PVM is located in. This leaves free more space on the nodes' hard drives, as well as reduces the amount of maintenance to only one copy on one machine. The nodes are all setup to auto-mount the PVM directory from the master upon reboot.

Because security is an issue, the beowulf is set up so that only one machine, the master, can be accessed from outside of the beowulf network. This is done by installing two ethernet cards in the master computer, one that communicates only with the beowulf network, and one that communicates with an outside network and the Internet. This effectively limits access to the cluster to individuals who have access to the master computer. Once logged onto the master, there is only one user account which has access to the nodes.

Scientific Notebook 115E
20-1402-461
Inititals: CC

Chuck Connor
May 20, 2000

Using class A (10.0.0.0) IP addresses for the nodes restricts access to the nodes from the "outside world" since private addresses aren't visible on the public Internet. Private IP addresses are sometimes referred to as Non-Routable IP. This is because the addresses are not routed to the Internet by an Internet Service Provider.

Hardware

10 nodes, each with
1 200 MHz Pentium Pro CPU
____K Cache
_____ RAM
_____ GB Hard Drive
_____ Ethernet cards
_____ Floppy Drive
_____ Video Board with _____ mb RAM
9 7' "Category 5" UTP cables with RJ45 ends to connect the nodes to the hub
1 25' "Category 5" UTP cables with RJ45 ends to connect the master to the hub
ADH-6160 16 port 10/100Mbps Dual Speed Hub
15" Monitor - for the nodes
20" Monitor - for the Master

Software

_____ Kernel
Linux distribution Suse 6.4.
Compiler gcc 2.7
PVM version 3.4

To see the configuration of each node click on the following links.
Masaya

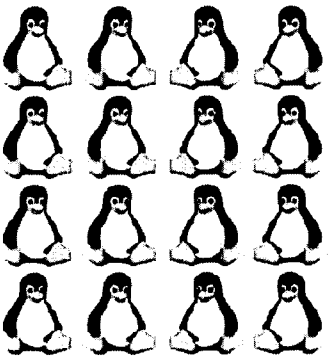
Beowulf.us

HOSTNAME: masaya

eth0:
LOCALHOST: masaya.geophysics.swri.edu
IP: 129.162.79.56
Ethernet card: 3Com Corporation

eth1:
LOCALHOST: b01.beowulf.us
IP: 10.0.0.1
Ethernet card: 3c905 100BaseTx [Boomerang]

Memory: _____ bytes - _____ kB
Disk: _____
CPU: 450MHz Pentium III
CDROM: _____



(Penguins)

Partition Values

Device	From	To	Blocks	Type	Mount
/dev/hda1	1	26	208813	83	Native /
/dev/hda2	27	65	313267	82	Swap swap
/dev/hda3	66	104	313267	83	Linux Native /var
/dev/hda4	105	784	5462100	83	Linux Native /usr

B04

Beowulf.us

HOSTNAME: B04

LOCALHOST: b04.beowulf.us
IP: 10.0.0.4
3Com 3c905

MX 2024
787 Cylinders
4,128,768 bytes per cylinder

Memory: 131073 kB
Disk: 3.2 Gig
CPU: Intel Pentium MMX 200
CDROM: .....

Partition Values

Device	From	To	Blocks	Type	Mount
/dev/hda1	1	102	411232	83	Linux Native /
/dev/hda2	103	170	274176	82	Swap swap
/dev/hda3	171	272	411264	83	Linux Native /var
/dev/hda4	273	787	2076480	83	Linux Native /usr

B07

Beowulf.us

HOSTNAME: B07

eth0:
LOCALHOST: b07.beowulf.us
IP: 10.0.0.7
3Com 3c900

767 Cylinders
4,128,768 bytes per cylinder

Memory: 131072 kB
Disk: WDC AC23200L
CPU: Intel Pentium MMX 200
CDROM: FX140S

Partition Values

Device	From	To	Blocks	Type	Mount
/dev/hda1	1	102	411232	83	Linux Native /
/dev/hda2	103	170	274176	82	Swap swap
/dev/hda3	171	272	411264	83	Linux Native /var
/dev/hda4	273	767	1995840	83	Linux Native /usr

B10

Beowulf.us

HOSTNAME: B10

LOCALHOST: b10.beowulf.us
IntelPro100 (Intel82557)
IP: 10.0.0.10

..... Cylinders
..... bytes per cylinder

Memory: 127560 kB
Disk: WDC AC33200L
CPU: Intel Pentium MMX 200
CDROM: IDE10602

Partition Values

Device	From	To	Blocks	Type	Mount
/dev/hda1	1	.....	.....	83	Linux Native /
/dev/hda2	.....	.....	.....	82	Swap swap
/dev/hda3	.....	.....	.....	83	Linux Native /var
/dev/hda4	.....	.....	.....	83	Linux Native /usr

B02

Beowulf.us

HOSTNAME: B02

LOCALHOST: b02.beowulf.us
IP: 10.0.0.2
Ethernet card: 3Com 3c905

Memory: 65536 bytes - 62672 kB
Disk: WDC AC23200L
CPU: Intel Pentium MMX 200
CDROM: FX162T

MX 2037
787 Cylinders

4,128,768 bytes per cylinder

Partition Values

Device	From	To	Blocks	Type	Mount
/dev/hda1	1	102	411232	83	Linux Native ./
/dev/hda2	103	170	274176	82	Swap swap
/dev/hda3	171	272	411264	83	Linux Native /var
/dev/hda4	273	787	2076480	83	Linux Native /usr

B05

Beowulf.us

HOSTNAME: B05

LOCALHOST: b05.beowulf.us
IP: 10.0.0.5
3Com 3c900

MX 2088
1027 Cylinders
8,225,280 bytes per cylinder

Memory: 131072 kB
Disk: 5.0 Gig
CPU: Intel Pentium MMX 200
CDROM: FX1405

Partition Values

Device	From	To	Blocks	Type	Mount
/dev/hda1	1	51	409626	83	Linux Native /
/dev/hda2	52	85	273105	82	Swap swap
/dev/hda3	86	136	409657	83	Linux Native /var
/dev/hda4	137	1027	7156957	83	Linux Native /usr

B08

Beowulf.us

HOSTNAME: B08

LOCALHOST: b08.beowulf.us
IP: 10.0.0.8
3Com 3c905

787 Cylinders
4,128,768 bytes per cylinder

Memory: 65536 kB
Disk: WDC AC23200L
CPU: Intel Pentium MMX 200
CDROM: ATAPI 8X Model CS-R380

Partition Values

Device	From	To	Blocks	Type	Mount
/dev/hda1	1	102	411232	83	Linux Native /
/dev/hda2	103	170	274176	82	Swap swap
/dev/hda3	171	272	411264	83	Linux Native /var
/dev/hda4	273	787	2076480	83	Linux Native /usr

B11

B03

Beowulf.us

HOSTNAME: B03

LOCALHOST: b03.beowulf.us
IP: 10.0.0.3
3Com 3c905

MX 2067
787 Cylinders
4,128,768 bytes per cylinder

Memory: 130048 kB Em - 131072 kB total
Disk: 3.2 Gig
CPU: Intel Pentium MMX 200
CDROM: .....

Partition Values

Device	From	To	Blocks	Type	Mount
/dev/hda1	1	102	411232	83	Linux Native /
/dev/hda2	103	170	274176	82	Swap swap
/dev/hda3	171	272	411264	83	Linux Native /var
/dev/hda4	273	787	2076480	83	Linux Native /usr

B06

Beowulf.us

HOSTNAME: B06

LOCALHOST: b06.beowulf.us
IP: 10.0.0.6
3Com 3c905

787 Cylinders
4,128,768 bytes per cylinder

Memory: 131072 kB
Disk: WDC AC23200L
CPU: Intel Pentium MMX 200
CDROM: FX162N

Partition Values

Device	From	To	Blocks	Type	Mount
/dev/hda1	1	102	411232	83	Linux Native /
/dev/hda2	103	170	274176	82	Swap swap
/dev/hda3	171	272	411264	83	Linux Native /var
/dev/hda4	273	787	2076480	83	Linux Native /usr

B09

Beowulf.us

HOSTNAME: B09

eth0:
LOCALHOST: b09.beowulf.us
IP: 10.0.0.9
3Com 3c905

787 Cylinders
4,128,768 bytes per cylinder

Memory: ..... kB
Disk: WDC AC33200L
CPU: Intel Pentium MMX 200
CDROM: FX162T

Partition Values

Device	From	To	Blocks	Type	Mount
/dev/hda1	1	102	411232	83	Linux Native /
/dev/hda2	103	170	274176	82	Swap swap
/dev/hda3	171	272	411264	83	Linux Native /var
/dev/hda4	273	787	2076480	83	Linux Native /usr

Setting up the Network

Start YaST (Yet another System Tool)and choose the following options in the order given:

1.System administration
2.Integrate hardware into system
you must tell your computer that it has a modem before you set up your network

1.System administration
2.network
3.

Most of the instructions needed for network installation can be found in chapter 5 Networking Linux in the SuSE Linux 6.4 book.

There needs to be name resolution between all machines on the network, the etc/hosts file must contain the names of all nodes in the cluster.

```
#
# hosts       This file describes a number of hostname-to-address
#             mappings for the TCP/IP subsystem.  It is mostly
#             used at boot time, when no name servers are running.
#             On small systems, this file can be used instead of a
#             "named" name server.
# Syntax:
#
# IP-Address  Full-Qualified-Hostname  Short-Hostname
#
# special IPv6 addresses
#beowulf cluster


10.0.0.1       b01.beowulf.us b01
10.0.0.10       b10.beowulf.us  b10
10.0.0.11       b11.beowulf.us  b11
10.0.0.2       b02.beowulf.us  b02
10.0.0.3       b03.beowulf.us  b03
10.0.0.4       b04.beowulf.us  b04
10.0.0.5       b05.beowulf.us  b05
```

10.0.0.6 b06.beowulf.us b06
10.0.0.7 b07.beowulf.us b07
10.0.0.8 b08.beowulf.us b08
10.0.0.9 b09.beowulf.us b09

In order to allow root logins from other machines, a switch must be set in the /etc/rc.config file.

ROOT_LOGIN_REMOTE="yes"

After making any changes to the rc.config SuSEconfig must be launched with the following command, in order to generate the necessary configuration files.

/sbin/SuSEconfig

Setting up NFS File Server

For information on how to set up NFS, look in /usr/doc/howto/en/ NFS-HOWTO

The following information pertains to the Server:

Give read/write permissions for directories on the server to the nodes in the /etc/exports file with the command:

<directory> <node>(<permissions>)

For example, to give read and write permissions in the /usr directory on masaya to b02 the command would look like this:

/usr b02(rw)

The /etc/exports file looks like this:

See exports(5) for a description.
This file contains a list of all directories exported to other computers.
It is used by rpc.nfsd and rpc.mountd.

/usr b02(rw)
/usr b03(rw)

/usr b04(rw)
/usr b05(rw)
/usr b06(rw)
/usr b07(rw)
/usr b08(rw)
/usr b09(rw)
/usr b10(rw)
/usr b11(rw)

/home b02(rw)
/home b03(rw)
/home b04(rw)
/home b05(rw)

After having made changes to the /etc/exports file, you must run the exportfs script in order to have the changes recognized. If your machine does not have exportfs (ours didn't) you should create it and store it in /usr/sbin/exportfs. Change the permissions to be executable and run exportfs as root whenever you make changes to the /etc/exports file.
The exportfs script looks like this:

```
#!/bin/sh
 killall -HUP /usr/sbin/rpc.mountd
 killall -HUP /usr/sbin/rpc.nfsd
 echo re-exported file systems
```

The RPC (Remote Procedure Calls) portmapper must be running. This server is started with the command:

/sbin/init.d/rpc

You can start it by hand now, but it will need to be started every time you boot your machine so you need to make/edit the rc scripts.

The following information pertains to the Nodes:

Mount user accounts from Masaya on the nodes with the command:

mount -t nfs <host>:<remote path> <local path>

For example, to mount the /usr/people directory on masaya onto the /usr/people directory of b02:

mount -t nfs b02:/usr/people /usr/people

To make this mount everytime the system reboots without having to do it manually, type the following line in the /etc/fstab file:

10.0.0.1:/usr/people /usr/people nfs

Parallel Virtual Machine

Our parallel LINUX cluster uses the PVM (Parallel Virtual Machine) System. The cluster consists of 10
nodes, each with a 200 MHz Pentium II CPU, and a 450MHz Pentium III for the server. While the nodes
are basically the same, they are not identical, therefore, the cluster is considered heterogeneous. The
compiler is gcc 2.7 and the PVM is version 3.4.

Setting up PVM:

PVM needs to be installed on each machine of the cluster. PVM is availabe at
www.netlib.org/pvm3/index.html. The PVM executables need to be installed on each machine, but
the PVM libraries only have to be on machines where PVM code development is being done.

After installing, add the following to your .bashrc:

```
#setting environment variable for PVM
PVM_ROOT=/usr/lib/pvm3
      export PVM_ROOT
source /usr/lib/pvm3/lib/.bashrc.stub
```

Compiling Code:

Using PVM:

To start pvm:

pvm

Commands in pvm:

ps	shows proccess information
conf	shows configuration
adds B02	adds B02 to the virtual machine
deletes B02	deletes B02 from the virtual machine
spawn -> application	spawns a pvm application
kill	terminates an pvm proccess
quit	quits pvm console but leaves PVM daemons and jobs running
halt	kills all PVM processes, the console and all PVM daemons

A host file should be created which contains a list of all the hosts and their configurations to be added
to the virtual machine:

B01 ep=$HOME
B02 ep=$HOME
B03 ep=$HOME
B04 ep=$HOME
B05 ep=$HOME

The host file can be given any name is started with the following command:

pvm cluster_hostfile

To start pvm on the server (masaya), one needs to be certain that it is started with the the name that
designates the ethernet connection to the cluster:

pvm -nB01 cluster_hostfile

(The "-n" command needs to be followed by the name of the machine in all caps)

PVM needs to have remote access to the other machines in the cluster. The pvm user needs to
create .rhosts file in the user's home directory which allows password free r-comman accces to a
specific user account. The rhost file should be readable only by the user!!! .rhosts:

b01 beowulfuser
B02 beowulfuser
B03 beowulfuser
B04 beowulfuser
B05 beowulfuser

Common Errors

B07 Can't start pvmd

PVM was unable to find the PVM executable on machine B07 or was unable to start PVM on B07. If the PVM executable is on B07, make sure that B07 has an .rhosts file that permits remote access by the master machine

Trouble spawning slaves. Aborting. Error codes are: TID 0 -7

Pvm was unable to find the slave file a host machine. When spawning slaves, pvmby default looks in $HOME/pvm3/bin/LINUX for the application. One can modify the host file to specify a different path to the executable. If the the application tasks on B07 are located in $HOME/pvm_code, the host file line for B07 should look like:

B07 ep=$HOME/pvm_code

It is also possible to receive this error when starting PVM on the server machine, but using the -n option without the server name in all caps.
Abnormal terminations of PVM may leave files in /tmp which can cause to PVM to fail when it restarts. One should delete all the /tmp/pvm*. files and attempt to restart. A list of common error codes can be found at www.mhpcc.edu/training/tutorials/html/pvm/error.msg.html
PVM Links

www.netlib.org/pvm3/book/pvm-book.html

Suzuki in Parallel

The grid code was written in C to be run on a parallel LINUX cluster using PVM (Parallel Virtual Machine) version 3.4. The purpose is to illustrate and attempt to make a master/slave parallel program to do a grid.

The PVM master grid code reads 100 lines of data from an input file and starts the slave processes according to a matrix with X and Y coordinates as its indices.

The master collects accumulaton data for a single point with a large set of input variables. This large set of input variables is split up with all the slaves

The slaves make the desired grid by knowing the x and y origins the grid-spacing, and the steps in x,y. The input to the slaves must be white-space delimited.

The following are 12 input variables for the slave code:

1.total_ash_mass
2.ymin
3.ymax
4.dmean
5.dsigma
6.phiash
7.dfshape
8.dbeta
9.w0
10.h
11.u
12.udir

The following are assumed constant:

eddy diffusivity in the atmosphere (c) = 400.0
density of air (phiair) = 0.001293
dynamic viscosity of air (nua) = 0.00018
gravity (gravity) = 980.0

Define the limits of integration:

xmin is the elevation of the vent
xmax is the total hight of the eruption column
ymin and ymax are minimum and maximum grainsizes, respectively

Do the integration over the eruption column height in 300 steps

xsteps = 300

Do the intergration over the particle size range in 100 steps

ysteps = 100

Define the coordinate system with respect to wind direction. Here we are just assuming that the wind is varying N or south of a western direction (need to be generalized).

The file looks like:

Statistics

---
tephra_master lasted 299 secs
8 hosts were used with the input file (data500)

---
tephra_master lasted 242 secs
10 hosts were used with the input file (data500)

Time to calulate ash accumulation at one point with 500 input sets:
---
9 nodes of Beowulf cluster and 450 Mhz PIII = 242 secs

8 nodes of Beowulf cluster (200Mhz Pentium II) = 299 secs

4 nodes of Beowulf cluster (200Mhz Pentium II) = 604 secs

200 Mhz Pentium II = 2399 secs

450 Mhz Pentium III = 859 secs

The following stats were obtained on 5/31/2000
---
tephra_master lasted 488 secs
5 hosts were used with the input file (data.in)

---
tephra_master lasted 303 secs
8 hosts were used with the input file (data.in)

---
tephra_master lasted 244 secs
10 hosts were used with the input file (data.in)

Source Code

The following are links to the source code for suzuki in parallel

Directions:
TEPHRA_README

Nathan Franklin
May 2, 2000

| Master and Slave Tephra Accumulation PVM Programs |
---

gridmaster.c and gridslave.c portions the grid for parallel processing, while tephra_master.c and tephra_slave.c split up the set of input variables for the parallel processing.

gridmaster.c grislave.c

Two programs writen in C with PVM libraries.
They compute (in parallel) the tephra accumulation over a grid.

gridmaster <variable file> <grid dimensions>

<variable file>= file of 12 input variables per line

<grid dimensions>= file of 5 numbers: x-origin, number of grid points along x-axis, y-origin, number of grid points along y, and the spacing between grid points

To compute the accumulation of the grid in parallel, gridmaster.c spawns gridslave.c on

each node of the cluster. Each spawned gridslave.c calculates the accumulation over its assigned portion of the grid and returns the results to gridmaster.c .

Splitting up portions of the grid to be computed by the slave proccesses is currently achieved by dividing up the grid along the x-axis. However, for this to work correctly, it is assumed that the number of grid points along the x-axes is divisible by number of spawned slaves. The code does not account for other cases in portioning the work.

tephra_master.c tephra_slave.c

Two programs writen in C with PVM libraries.
They compute (in parallel) the tephra accumulation over a grid.

gridmaster <variable file> <grid dimensions>

<variable file>= file of 12 input variables per line

<grid dimensions>= file of 5 numbers: x-origin, number of grid points along x-axis, y-origin, number of grid points along y, and the spacing between grid points

To compute the accumulation of the grid in parallel, tephra_master.c spawns tephra_slave.c on each node of the cluster. Each spawned gridslave.c calculates the accumulation over the entire portion of the grid. However, each slave only does so with a subset of the set of input variables and then returns the results to tephra_master.c .

Splitting up portions of the set of input variables is done by the slave proccesses. The set of input variables is divided into portions which each slave uses. A simple division of the set of variables by the number of slaves is done. However, for this division of work to be done correctly, it is assumed that the number of subset of varialbes is divisible by number of spawned slaves. The code does not account for non-divisable cases.

The input files:

data1 - file containing one data point

2.047825917000626E13 -5.0 5.0 -1.0 1.0 1.1413360493378952 0.5 0.5 3182.976378365838 5.024348390440153 1273.7058802138117 -0.0204251

data.in - file containing 500 data points

(First lines of data.in)
1.6398102618191145E13 -5.0 5.0 -1.0 1.0 1.142380536174999 0.5 0.5
6492.681046046187 5.5790885593758155 919.4482379200401 -0.2451391884920667
1.3571629492841992E12 -5.0 5.0 -1.0 1.0 1.0165999642558203 0.5 0.5
9719.182809293165 7.439070059360828 844.9732246066706 -0.30829295312536487
1.0038912253104176E13 -5.0 5.0 -1.0 1.0 1.0359910260705454 0.5 0.5
6893.301697250275 4.606298119101088 700.8543233869536 -0.26092898632449957
3.6765261454956086E13 -5.0 5.0 -1.0 1.0 1.163482439459926 0.5 0.5
5723.022457097762 7.242072474099914 506.12480815263086 -0.10943524777486675
2.4229593230443213E12 -5.0 5.0 -1.0 1.0 0.9692043338248323 0.5 0.5
7266.634413136851 5.602386925349782 1044.880738751789 -0.12249222101535764
4.968400249756788E13 -5.0 5.0 -1.0 1.0 1.1383387314998117 0.5 0.5
9324.989646414626 5.980779619299832 502.3682645557855 -0.36380422590463835
1.4005758486603021E13 -5.0 5.0 -1.0 1.0 1.143483271425962 0.5 0.5
6752.9260140948445 3.505259654274369 758.1271093272615 -0.307315604857167
3.71796823160394E13 -5.0 5.0 -1.0 1.0 1.0244083026025623 0.5 0.5 6948.721159485369
7.863964913469123 629.2134332184123 0.0639219161554872
4.763320057061053E12 -5.0 5.0 -1.0 1.0 1.1391687784394582 0.5 0.5
6924.445041841467 2.4711026507978606 722.0485918515963 0.015754978709200826
2.636468894957047E13 -5.0 5.0 -1.0 1.0 1.11408861881662 0.5 0.5 5023.019526162723
3.915754039804818 615.6006322945468 0.1429920373501288
6.09798687564312E12 -5.0 5.0 -1.0 1.0 1.027525852320106 0.5 0.5 7836.84496908234
5.119098619479689 596.6952143487503 -0.016047811102478094
2.940411388173128E13 -5.0 5.0 -1.0 1.0 1.0286296148364134 0.5 0.5
6370.198269202738 5.56643370738395 1124.7583604111737 -0.010656954207059637
1.2569591622611992E12 -5.0 5.0 -1.0 1.0 1.1735070704450343 0.5 0.5
7631.256212055163 4.349823828069781 979.0803155798883 -0.24730636421685795
2.243420293230339E13 -5.0 5.0 -1.0 1.0 1.054617053481302 0.5 0.5 9119.32354655998
4.108211189440069 512.6765707291343 -0.07263483989918093
1.0431233553838263E12 -5.0 5.0 -1.0 1.0 1.0017618995175965 0.5 0.5
7421.454156319634 2.841804215029601 919.912108578664 0.0820698277838131
6.310279426601699E13 -5.0 5.0 -1.0 1.0 1.0874799720355723 0.5 0.5
7289.158631194827 6.155136093903421 512.5712890897747 -0.36384022757517126
3.1280793414430034E12 -5.0 5.0 -1.0 1.0 1.147330296220498 0.5 0.5
9635.434442949754 4.110288390993726 714.0169561278876 -0.010300092351435293
1.7454063395529617E12 -5.0 5.0 -1.0 1.0 0.9161834440760109 0.5 0.5
7302.54567868232 5.126245560872986 663.7987080214925 -0.025486056381007465

5.772387927170236E13 -5.0 5.0 -1.0 1.0 0.916314189050631 0.5 0.5 8729.933945891371

dimensions

100000 1 0 1 100000

Script to create input data file :
Suzpara.java

```
import java.io.*;

public class Suzpara5 {


   public static final double total_min = 5.0e5;  // volume of deposit in m^3 (NOT DRE)
public static final double total_max = 1.0e8;

public static final double dmin_min = -5.0;
public static final double dmin_max = -5.0;

public static final double dmax_min = 5.0;
public static final double dmax_max = 5.0;

public static final double dmean_min = -1.0;
public static final double dmean_max = -1.0;

public static final double dsigma_min  =1.0;
public static final double dsigma_max  =1.0;

public static final double beta_min = 0.5;
public static final double beta_max = 0.5;

public static final double w0_min = 5000.0;
public static final double w0_max = 10000.0;

public static final double h_min = 2.0;
public static final double h_max = 8.0;

public static final double u_min = 500.0;
```

```
public static final double u_max = 1500.0;

public static final double udir_min = 10.0;
public static final double udir_max = -25.0;

public static final double phiash_min = 0.9;
public static final double phiash_max = 1.2;

public static final double fshape_min = 0.5;
public static final double fshape_max = 0.5;


public static double log10(double xx) {
      double logxx;
      logxx = Math.log(xx)/Math.log(10.0);
      return logxx;
   }

   public static void main (String[] argv)
   throws IOException, FileNotFoundException
{


      double t1_max, t1_min, t1_num, total_num;
      double dmin_num, dmax_num, dmean_num, dsigma_num;
      double beta_num, w0_num, u_num, udir_num, h_num;
      double phiash_num, fshape_num, total_ash_mass_num;
      double time_num;

      double qdot,log10rv,rv,w0,total_time;
      int n=0;

      //create the file output
PrintWriter f1 = new PrintWriter(
new FileOutputStream ("file1.out"), true);

PrintWriter f2 = new PrintWriter(
new FileOutputStream ("file2.out"), true);
```

```
PrintWriter f3 = new PrintWriter(
new FileOutputStream ("file3.out"), true);

PrintWriter f4 = new PrintWriter(
new FileOutputStream ("data.in"), true);

      do {
         t1_max = log10(total_max);
         t1_min = log10(total_min);
         t1_num = t1_min + Math.random()*(t1_max -t1_min);
         total_num = Math.pow(10.0,t1_num);


         dmin_num = dmin_min + Math.random() *(dmin_max - dmin_min);
         dmax_num = dmax_min + Math.random() *(dmax_max - dmax_min);
         dmean_num = dmean_min + Math.random() *(dmean_max - dmean_min);
         dsigma_num = dsigma_min + Math.random() *(dsigma_max - dsigma_min);
         beta_num = beta_min + Math.random() *(beta_max - beta_min);
         h_num = h_min + Math.random() *(h_max - h_min);
         w0_num = w0_min + Math.random() *(w0_max - w0_min);
         u_num = u_min + Math.random() *(u_max - u_min);

         udir_num = udir_min + Math.random() *(udir_max - udir_min);
         udir_num = udir_num*Math.PI/180.0;

         phiash_num = phiash_min + Math.random() *(phiash_max - phiash_min);
         fshape_num = fshape_min + Math.random() *(fshape_max - fshape_min);


         qdot = Math.pow(h_num/1.67, 4.0); // mass eruption rate in m^3/s DRE
         total_ash_mass_num = total_num * phiash_num* 1000.0 * 1000.0; // total mass erupted
in gm
         time_num = (total_num*phiash_num* 1000.0)/(qdot*2600.0*3600.0); //duration of
eruption in hrs



 if (total_num  > 1.0e6 && total_num < 1.0e8 && time_num < 120.0) {
```

```
f1.println(total_num + "    " + qdot*2600.0);
f2.println(h_num + "   " + w0_num/100.0);
f3.println( time_num);

f4.println(total_ash_mass_num +" "+dmin_num+" "+dmax_num+" "+
        dmean_num+" "+dsigma_num+" "+phiash_num+" "+fshape_num+" "+
        beta_num+" "+w0_num +" "+h_num+" "+u_num+" "+udir_num);



        n++;
 }
     }
     while (n<500);

   }


}
```

The source code:
gridmaster.c

```
/*
** Program name:   gridmaster.c
**
** Purpose:   To illustrate and attempt to make a master/slave parallel
**          program to do a grid
**
** Author:    Nathan Franklin
**
*/



#include <stdio.h>
#include <time.h>
```

```
#include <stdlib.h>
#include "pvm3.h"
#define SLAVENAME "gridslave"

int DataIn(double x[], char fn[]);

int main(argc,argv)
int argc;
char *argv[];
{
   int mytid;              /* my task id */
   int tids[32];                    /* slave task ids */
   int nproc;  //number of proccess
   int numt;
   int i,j;
   int who;
   int length_result;   //length of result (returned portion of grid)
   int msgtype;
   int nhost;  // number  of hosts
   int narch;
   int  n_grid_dimensions,n_variables;
   int time_length;    //length in micro seconds

   struct timeval start, stop;  //start and stop times


   double variables[15000];
   double grid_dimensions[5];
   double result[15000];
   struct pvmhostinfo *hostp;  //information of about each hosts:
                     // pvmd task id, name, arch, and relative
                     // speed

  // standard usage io
  if ( argc != 3 ) {
    fprintf ( stderr, "Usage: %s <variable file> <grid dimensions>\n", argv[0] );
    exit (0);
  }
```

```
   gettimeofday(&start, (struct timezone*)0);


   //enroll in pvm
   mytid = pvm_mytid();

   // Set number of slaves to start
   pvm_config( &nhost, &narch, &hostp );
   nproc = nhost;


   if( nproc > 32 ) nproc = 32 ;
      printf("Spawning %d worker tasks ... " , nproc);


// ----------------
// | Start Slaves |
// ----------------

   numt=pvm_spawn(SLAVENAME, (char**)0, 0, "", nproc, tids);
      //int numt = pvm_spawn( char *task, char **argv, int flag,
      //                      char *where, int ntask, int *tids )
      //Pointer to an array of arguments to the executable
      //nproc = number of copies of executable to be started
      //tids = array showing the tids of spawned executable
   if( numt < nproc ){
     printf("\n Trouble spawning slaves. Aborting. Error codes are:\n");
     for( i=numt ; i<nproc ; i++ ) {
       printf("TID %d %d\n",i,tids[i]);
     }
     for( i=0 ; i<numt ; i++ ){
       pvm_kill( tids[i] );
     }
     pvm_exit();
     exit(1);
   }
      printf("SUCCESSFUL\n");
```

```
// ----------------------------
// | Get variables from file |
// ----------------------------

    n_variables=DataIn(variables,argv[1]);
    n_grid_dimensions=DataIn(grid_dimensions,argv[2]);
    if(n_grid_dimensions!=5)
    {
      printf("Dimension File Error: check %s!",argv[2]);
      pvm_exit();
      exit(1);
    }
// -------------------------------------
// | Broadcast initial data to slaves |
// -------------------------------------

      pvm_initsend(PvmDataDefault);
      pvm_pkint(&nproc, 1, 1);
      pvm_pkint(tids, nproc, 1);
      pvm_pkint(&n_variables, 1, 1);
      pvm_pkdouble( &variables[0], n_variables, 1 );
      pvm_pkdouble( &grid_dimensions[0], 5, 1 );
      pvm_mcast(tids, nproc, 0);
//pvm_cast----Multicasts the data in the active message
//     buffer to a set of tasks.
//     info=pvm_cast(tids,ntask,msgtag)
//          ntask = number of tasks to be sent to
//          msgtag: int message tag to allow user's program
//                to distingues between different kinds of messages.
//          info = int -> value less than zero indicates an e(rror.



// -------------------------------------
// | Wait for results from the slaves |
// -------------------------------------

   msgtype = 50;  //going to listen to mesages with this type
```

```
   for( i=0 ; i<nproc ; i++ ){
     pvm_recv( -1, -1 );
     pvm_upkint( &who, 1, 1 );
     pvm_upkint( &length_result, 1, 1);
     pvm_upkdouble( &result[0], length_result, 1 );
     printf("Receiving from procces #%d: \n",who);

     for(j=0; j<length_result; j=j+3)
       printf( "%f %f %f\n", result[j],result[j+1],result[j+2]);

   }

// ----------------------------
// | End of program... quit   |
// ----------------------------
   pvm_exit();

   gettimeofday(&stop, (struct timezone*)0);
   time_length= stop.tv_sec - start.tv_sec;
   printf("%s lasted %d secs\n",argv[0],time_length);
   return 1;
}


int DataIn(double x[], char fn[])
{
  FILE *fp;
  int N=0, r;
  float temp;

  fp=fopen(fn, "r");
  if(fp==NULL)
  {
    printf("\nFile Error\n----------\n %s does not exist.\n",fn);
    pvm_exit();
    exit(1);
  }
  r=fscanf(fp, "%f", &temp);
  while(r!=EOF)
```

```
  {
    x[N]=temp;
    N++;
    r=fscanf(fp, "%f", &temp);
  }
  fclose(fp);
  return N;
}
```

gridslave.c

```
/*
** Program name:   gridslave.c
**
** Purpose:   To illustrate and attempt to make a master/slave parallel
**          program to do a grid
**
**
** Notes:  This program makes the desired grid by knowing the x and y origins
**        the grid-spacing, and the steps in x,y.
**
** Input:  The input data is seperated by white space (not commas).
**
**
**     Desired Additions!:
**                    -comment this and master program
**                    -precalcute any possible variables that might be
**                    in the double integrations....
**                -!!!!remove global variables.... all passing
**                 (last two probably not worth it)
**
** Author:    Nathan Franklin
**
*/



#include <stdio.h>
```

```
#include "pvm3.h"
#include <math.h>

//Variables-global
#define PI 3.141592654
#define SIZE_VARIABLES 15000   //length of variable array = 12 variables * n
                       //                    trials

#define NUMBER_OF_VARIABLES 12   //number of variables = 12 variables

#define SIZE_ARRAYGRID 100000  //length of grid array:  number of grid points *
                       //            number of runs * 3 (x,y,
                       //            accumulation)


// following are the 12 input variables
double total_ash_mass,ymin, ymax, dmean, dsigma, phiash;
double dfshape, dbeta, w0, h, u, udir;

//following are assumed const.
double c      = 400.0;       // eddy diffusivity in the atmosphere
double phiair =  0.001293;   // density of air
double nua    =  0.00018;    // dynamic viscosity of air
double gravity = 980.0;      // gravity


//grid():      Makes the grid
int grid(double data[], double input_variables[], int n_variables, double xorigin, double xsteps,
double yorigin, double ysteps, double spacing);

//tephra_accumulation():      Used in grid() to find tephra accumulation
double tephra_accumulation(double xpos, double ypos);

//functions used by tephra_accumulation to find answer:

double phi2cm( double xx );
double w( double zspace );
double p( double zspace, double xrho );
double ts( double zspace );
```

```
double t( double zspace, double xrho );
double v0( double xrho );
double frho( double xrho );
double f( double xspace, double yspace, double zspace, double xrho);



int main(      int argc,  char *argv[])
{


// -------------------------------
// |   Setting up variables     |
// -------------------------------


// PVM variables:
//----------------
   int mytid;      /* my task id */
   int tids[32];   /* task ids   */
      int me, i, nproc, master, msgtype;
   int length_data_array;
   int n_variables;


// Tephra program variables:
//---------------------------
   // Grid dimensions
   double x_origin;
   double x_steps;
   double y_origin;
   double y_steps;
   double spacing;

   double arraygrid[SIZE_ARRAYGRID];   //array of output grid
   double variables[SIZE_VARIABLES];   //array of input variables
   double grid_dimensions[5];          //array of grid dimensions
```

```
// --------------------------------
// | Receive data from the master |
// --------------------------------

   mytid = pvm_mytid();      // who am I?
   msgtype = 0;              // setting msgtag to 0. Master should
                             // be also broadcasting with 0.
   pvm_recv( -1, msgtype );
// If tid = -1 and msgtag is
//      defined by the user, then pvm_recv will accept  a  message
//      from any process which has a matching msgtag.

      pvm_upkint(&nproc, 1, 1);
      pvm_upkint(tids, nproc, 1);
      pvm_upkint( &n_variables, 1, 1);
      pvm_upkdouble( &variables[0], n_variables, 1 );
      pvm_upkdouble( &grid_dimensions[0], 5, 1 );


   /* Determine which slave I am (0 -- nproc-1) */
   for( i=0; i<nproc ; i++ )
     if( mytid == tids[i] ){ me = i; break; }




// --------------------------------
// | Do calculations with data    |
// --------------------------------

// Find my portion of the grid
//  <this 'can be/should be' changed -nmf

   spacing=grid_dimensions[4];
   x_steps=grid_dimensions[1]/nproc;
   x_origin=grid_dimensions[0]+((x_steps*me)*spacing);
   y_origin=grid_dimensions[2];
```

```
   y_steps=grid_dimensions[3];

//get grid -> arraygrid
   length_data_array = grid(arraygrid, variables, n_variables, x_origin, x_steps,y_origin, y_steps,
spacing);



// ---------------------------------
// |   Send results to master    |
// ---------------------------------
   pvm_initsend( PvmDataDefault );
   pvm_pkint( &me, 1, 1 );
   pvm_pkint( &length_data_array, 1, 1 );
   pvm_pkdouble( &arraygrid[0], length_data_array, 1 );
   msgtype = 50;
   master = pvm_parent();
   pvm_send( master, msgtype );


// ---------------------------------
// |   Program finished...exit   |
// ---------------------------------
   //pvm_initsend( PvmDataDefault );
   printf("end of program\n");
   pvm_exit();

   return 1; //
}
```

int grid(double data[], double input_variables[], int n_variables, double xorigin, double xsteps, double yorigin, double ysteps, double spacing)

```
{
  int i,j;
  int x_step_count,y_step_count,number_of_runs;
  int grid_xsteps;
  int grid_ysteps;

  double xpos, ypos;

  grid_xsteps=xsteps;
  grid_ysteps=ysteps;
  xpos=xorigin;
  ypos=yorigin;

  number_of_runs = n_variables / NUMBER_OF_VARIABLES; // # of variables
                                  // divided by # of variables
                                  // in a run
  i=0;
  x_step_count=0;


     while(x_step_count<grid_xsteps)
     {

          y_step_count=0;

          while(y_step_count<grid_ysteps)
          {
            //move to next point in grid
            xpos=xorigin + (x_step_count*spacing);
            ypos=yorigin + (y_step_count*spacing);

            //at this point find tephra accumulation
            //     for each set of input variables
            for(j=0;j<number_of_runs;j++)
            {
               total_ash_mass=input_variables[0+(j*12)];
               ymin   =input_variables[1+(j*12)];
               ymax   =input_variables[2+(j*12)];
               dmean  =input_variables[3+(j*12)];
```

```
                dsigma  =input_variables[4+(j*12)];
                phiash  =input_variables[5+(j*12)];
                dfshape =input_variables[6+(j*12)];
                dbeta   =input_variables[7+(j*12)];
                w0      =input_variables[8+(j*12)];
                h       =input_variables[9+(j*12)];
                u       =input_variables[10+(j*12)];
                udir    =input_variables[11+(j*12)];


                data[i]=xpos;
                i++;
                data[i]=ypos;
                i++;
                data[i]=tephra_accumulation(xpos, ypos);
                i++;

            }//end of for

            y_step_count++;
          }//end of 'y' while

          x_step_count++;
      }//end of 'x' while
  return i;
}


double tephra_accumulation(double xpos, double ypos)
{
      double xsectionwidth;
      double ysectionwidth;
      double xmin, xmax;
      double xstepwidth;
      double ystepwidth;
      double new_xpos, new_ypos;
      double x0, y0, x1, xi, y1, yi,ans;
      double total;
      int xsteps, ysteps, yyi, xxi;
```

```
// define the limits of integration
//--------------------------------
// xmin is the elevation of the vent
// xmax is the total hight of the eruption column
// ymin and ymax are minimum and maximum
// grainsizes, respectively

 // do the integration over the eruption
 // column height in 300 steps
 xsteps = 300;

 // do the intergration over the particle
 // size reange in 100 steps
 ysteps = 100;

 xmin= 0.0;
 xmax =h;
 xsectionwidth = xmax - xmin;
 ysectionwidth = ymax - ymin;
 xstepwidth = xsectionwidth / xsteps;
 ystepwidth = ysectionwidth / ysteps;


 // initialize the summation for the
 // contribution of each d phi and
 // each dz to the total accumulation
 // of tephra on the ground at the
 // point xspace, yspace
 total  = 0.0;


//define the coordinate system with
// respect to wind direction
//--------------------------------
// here we are just assuming that
// the wind is varying N or south of
// a western direction, need to be genrealized

 new_xpos = xpos * cos(udir) + ypos * sin(udir);
```

```
    new_ypos = -xpos*sin(udir) + ypos*cos(udir);

    //do the integration
    //-------------------
    for(yyi=1; yyi < ysteps; yyi++) {

        // define the small slice d phi

        yi = (double)yyi;
        y0 = ymin + yi * ystepwidth;
        y1 = ymin + (yi+(double)1.0) * ystepwidth;

        for(xxi=1; xxi < xsteps; xxi ++) {

          // define the small slice dz
          xi = (double)xxi;
          x0 = xmin + (xi) * xstepwidth;
          x1 = xmin + (xi+(double)1.0) * xstepwidth;

          // calculate the value of the ash accumulation  at
          // the corners  of the slice, and sum
          ans = f(new_xpos, new_ypos, x0, y0) + f(new_xpos, new_ypos, x1, y0) +
           f(new_xpos, new_ypos, x0, y1) + f(new_xpos, new_ypos, x1, y1);
          total += ans;
        }
    }

    // complete the integration by multiplying by dz and d phi
    // and dividing by 4, since 4 corners were used
    total *= xstepwidth * ystepwidth / 4.0;

    //all finished: return the answer
    return total;
}


/* ----------------- New Function Starts Here ---------------------- */
/* function phi2cm converts the ash diameter from
   units of phi to cm
```

```
*/

double phi2cm(double xx) {
  double cms;

  cms = 0.1 * 1.0 / pow(2, xx);
  return cms;
}

/* ---------- New Function Starts Here ---------- */
/* this function estimates the velocity
  as a function of height in the
  eruption column, given a
  linear decrease in velocity with
  height and an initial velocity
  of w0
*/

/* returns the upward particle velocity  */

double w(double zspace) {
  double particle_velocity;

  particle_velocity = w0 * (1.0 - zspace/h);

  // following only ocurrs as round-off error
 if (zspace > h) particle_velocity = 0.0;
  return particle_velocity;
}


/* ---------- New Function Starts Here ---------- */
/* this function calculates the probability
  density for diffusion of ash out of the
  eruption column. Some variation from Suzuki
  implemented to conserve mass

  depends on beta, w0, w(z), v0(xrho)
  where w0 is the eruption velocity in cm/s
```

```
  w(z) in the upward particle velocity at height z (cm/s)
  v0(xrho) is the particle settling velocity (cm/s)
  h is the column height (km)

  integrating fz from 0 to h should give unity

*/

double p(double zspace, double xrho) {
  double y, y0;
  double prob_dens_func;
  double voo;
  double demon1, demon2;
  voo = v0(xrho);
  y  = dbeta * w(zspace) / voo;
  y0 = dbeta * w0 / voo;
  demon1 = dbeta * w0 * y * exp(-y);
  demon2 = voo*h * (1.0-(1.0+y0) * exp(-y0));
  prob_dens_func = demon1/demon2;

  return prob_dens_func;
}



/* ---------------- New Function Starts Here ---------------------- */
/* function ts determines the particle diffusion time const
  in the atmosphere
  where zspace = height of the particle, converted here from
            km to cm, this should be wrt sea-level,
            zspace is given here in cm because
            of the units of eddy diffusivity
      c = eddy diffusivity in the atmosphere
        given as in cgs, e.g., 400 cm/s^(5/2)

  returns the particle diffusion time
*/

double ts (double zspace) {
```

```
  double particle_diffusion_time;
  double demon1, demon2;

  zspace *= 1.0e+5;
  demon1  = 5.0 * zspace * zspace;
  demon2  = 288.0 * c;
  particle_diffusion_time = pow(demon1/demon2, 2.0/5.0);

  return particle_diffusion_time;
}


/* ----------------- New Function Starts Here ---------------------- */
/* function t calculates the particle fall time
  in the atmposphere
  where: zspace = height in the atmosphere, here in km
              wrt to sealevel, because the const.
              -0.0625 is in units of 1/km
*/
double t (double zspace, double xrho) {
  double particle_fall_time;
  double demon1, demon2;

  demon1 = (1.0 - exp(-0.0625*zspace)) / v0(xrho);
  demon2 = pow(demon1, 0.926);
  particle_fall_time = 0.752e6 * demon2;
  return particle_fall_time;
}


/* ----------------- New Function Starts Here ---------------------- */
/* return a value for the particle terminal velocity
  at sea level, given the particle diameter, passed as x
  note that the shape factor does not depend on x
  note that the ash dens does not depend on x
*/

double v0 (double xrho) {
```

```
  double velocity0;
  double demon1, demon2, demon3;

  demon1 = 9.0 * nua * pow(dfshape, -0.32);
  demon2 = 81.0 * nua * nua * pow(dfshape, -0.64);
  demon3 = 1.5 * phiair * phiash * gravity * xrho*xrho*xrho * sqrt((1.07-dfshape));

  velocity0 = phiash * gravity * xrho * xrho /
          (demon1 + sqrt(demon2 + demon3));

  return velocity0;
}



/* ---------- New Function Starts Here ---------- */
/* this function calculates the expected fraction of particles
   in a given grainsize class (xrho) assuming a normal
   distribution in phi units about the mean, dmean,
   with standard deviation dsigma.
*/

double frho(double xrho) {
                                        double func_rho;
  double demon1, demon3, demon2;

  demon1  = 1.0 / (2.506628 * dsigma);
  demon3  = xrho - dmean;
  demon2  = exp(-demon3*demon3 / (2.0*dsigma*dsigma));
  func_rho = demon1 * demon2;

  if (func_rho >= 0.0) {
    return func_rho;
  }
  else {
    printf("error in ash size distribution - method frho");
    return func_rho;
  }
}
```

```
/* ---------- New Function Starts Here ---------- */
/* function f calculates the epected mass of ash to accumulate in a
given location xspace, yspace (cm from the volcano) in gm / cm^3
*/

double f(double xspace, double yspace, double zspace, double xrho) {
  double ash_fall;
  double ashdiam;
  double t1, t2;
  double demon1, demon2, demon3, demon4;

  ashdiam = phi2cm(xrho);
  t1 = t(zspace, ashdiam);
  t2 = ts(zspace);
  demon1 =  5.0 * total_ash_mass * p(zspace,ashdiam) * frho(xrho);
  demon2 =  8.0 * PI * c * pow((t1+t2), 5.0/2.0);
  demon3 = -5.0 * ((xspace-u*t1) * (xspace-u*t1) + yspace * yspace);
  demon4 =  8.0 * c * pow(t1+t2, 5.0/2.0);

  ash_fall = demon1 / demon2 * exp(demon3/demon4);

  if (ash_fall >= 0.0) {

    return ash_fall;
  }
  else {
    printf("h: %f, zspace: %f, ashdiam: %f\n",h, zspace,ashdiam);
//    printf("p: %f, ahsdiam: %f, zspace: %f\n",p(zspace,ashdiam), ashdiam);
    return ash_fall;
    }
}
```

tephra_master.c

```
/*
** Program name:   tephra_master.c
**
** Purpose:  To collect accumulaton data for a single point with a large set
**         of input variables.  This large set of input variables is split up
```

```
**        with all the slaves
**
** Author:    Nathan Franklin
**
*/



#include <stdio.h>
#include <stdlib.h>
#include "pvm3.h"
#define SLAVENAME "tephra_slave"

int DataIn(double x[], char fn[]);

int main(argc,argv)
int argc;
char *argv[];
{
   int mytid;                /* my task id */
   int tids[32];                        /* slave task ids */
   int nproc;  //number of proccess
   int numt;
   int i,j;
   int who;
   int length_result;   //length of result (returned portion of grid)
   int msgtype;
   int nhost;  // number  of hosts
   int narch;
   int  n_grid_dimensions,n_variables;


   double variables[130];
   double grid_dimensions[5];
   double result[1000];
   struct pvmhostinfo *hostp;  //information of about each hosts:
                        // pvmd task id, name, arch, and relative
                        // speed
```

```
  // standard usage io
  if ( argc != 3 ) {
    fprintf ( stderr, "Usage: %s <variable file> <grid dimensions>\n", argv[0] );
    exit (0);
  }



  //enroll in pvm
  mytid = pvm_mytid();

  // Set number of slaves to start
  pvm_config( &nhost, &narch, &hostp );
  nproc = nhost;


  if( nproc > 32 ) nproc = 32 ;
     printf("Spawning %d worker tasks ... " , nproc);


// ----------------
// | Start Slaves |
// ----------------

  numt=pvm_spawn(SLAVENAME, (char**)0, 0, "", nproc, tids);
     //int numt = pvm_spawn( char *task, char **argv, int flag,
     //                      char *where, int ntask, int *tids )
     //Pointer to an array of arguments to the executable
     //nproc = number of copies of executable to be started
     //tids = array showing the tids of spawned executable
  if( numt < nproc ){
    printf("\n Trouble spawning slaves. Aborting. Error codes are:\n");
    for( i=numt ; i<nproc ; i++ ) {
      printf("TID %d %d\n",i,tids[i]);
    }
    for( i=0 ; i<numt ; i++ ){
      pvm_kill( tids[i] );
    }
```

```
    pvm_exit();
    exit(1);
  }
    printf("SUCCESSFUL\n");


// ----------------------------
// | Get variables from file  |
// ----------------------------

   n_variables=DataIn(variables,argv[1]);
   n_grid_dimensions=DataIn(grid_dimensions,argv[2]);
   if(n_grid_dimensions!=5)
   {
     printf("Dimension File Error: check %s!",argv[2]);
     pvm_exit();
     exit(1);
   }
// --------------------------------------
// | Broadcast initial data to slaves  |
// --------------------------------------

     pvm_initsend(PvmDataDefault);
     pvm_pkint(&nproc, 1, 1);
     pvm_pkint(tids, nproc, 1);
     pvm_pkint(&n_variables, 1, 1);
     pvm_pkdouble( &variables[0], n_variables, 1 );
     pvm_pkdouble( &grid_dimensions[0], 5, 1 );
     pvm_mcast(tids, nproc, 0);
//pvm_cast----Multicasts  the  data  in the active message
//     buffer to a set of tasks.
//     info=pvm_cast(tids,ntask,msgtag)
//          ntask = number of tasks to be sent to
//          msgtag:  int message tag to allow user's program
//                to distingues between different kinds of messages.
//          info =  int -> value less than zero indicates an error.
```

```
// ------------------------------------
// | Wait for results from the slaves |
// ------------------------------------

   msgtype = 50;  //going to listen to mesages with this type

   for( i=0 ; i<nproc ; i++ ){
     pvm_recv( -1, -1 );
     pvm_upkint( &who, 1, 1 );
     pvm_upkint( &length_result, 1, 1);
     pvm_upkdouble( &result[0], length_result, 1 );
     printf("Receiving from procces #%d: \n",who);

     for(j=0; j<length_result; j=j+3)
       printf( "%f %f %f\n", result[j],result[j+1],result[j+2]);

   }

// ----------------------------
// | End of program... quit   |
// ----------------------------
   pvm_exit();
   return 1;
}


int DataIn(double x[], char fn[])
{
  FILE *fp;
  int N=0, r;
  float temp;

  fp=fopen(fn, "r");
  if(fp==NULL)
  {
    printf("\nFile Error\n----------\n %s does not exist.\n",fn);
    pvm_exit();
    exit(1);
  }
```

```
  r=fscanf(fp, "%f", &temp);
  while(r!=EOF)
  {
    x[N]=temp;
    N++;
    r=fscanf(fp, "%f", &temp);
  }
  fclose(fp);
  return N;
}
```

tephra_slave.c

```
/*
** Program name:  gridslave.c
**
** Purpose:  To illustrate and attempt to make a master/slave parallel
**          program to do a grid
**
**
** Notes:  This program makes the desired grid by knowing the x and y origins
**        the grid-spacing, and the steps in x,y.
**
** Input:  The input data is seperated by white space (not commas).
**
**
**     Desired Additions!:
**                    -comment this and master program
**                    -precalcute any possible variables that might be
**                    in the double integrations....
**                -!!!!remove global variables.... all passing
**                  (last two probably not worth it)
**
** Author:   Nathan Franklin
**
*/
```

```
#include <stdio.h>
#include "pvm3.h"
#include <math.h>

//Variables-global
#define PI 3.141592654
#define SIZE_VARIABLES 16000  //length of variable array = 12 variables * n
                      //                    trials

#define NUMBER_OF_VARIABLES 12  //number of variables = 12 variables

#define SIZE_ARRAYGRID 100000 //length of grid array:  number of grid points *
                      //           number of runs * 3 (x,y,
                      //           accumulation)


// following are the 12 input variables
double total_ash_mass,ymin, ymax, dmean, dsigma, phiash;
double dfshape, dbeta, w0, h, u, udir;

//following are assumed const.
double c      = 400.0;       // eddy diffusivity in the atmosphere
double phiair =  0.001293;   // density of air
double nua    =  0.00018;    // dynamic viscosity of air
double gravity = 980.0;      // gravity


//grid():      Makes the grid
int grid(double data[], double input_variables[], int n_variables, double xorigin, double xsteps,
double yorigin, double ysteps, double spacing,int n_slaves, int n_me);

//tephra_accumulation():      Used in grid() to find tephra accumulation
double tephra_accumulation(double xpos, double ypos);

//functions used by tephra_accumulation to find answer:

double phi2cm( double xx );
double w( double zspace );
double p( double zspace, double xrho );
```

```
double ts( double zspace );
double t( double zspace, double xrho );
double v0( double xrho );
double frho( double xrho );
double f( double xspace, double yspace, double zspace, double xrho);



int main(      int argc,  char *argv[])
{


// -------------------------------
// |   Setting up variables    |
// -------------------------------


// PVM variables:
//----------------
   int mytid;      /* my task id */
   int tids[32];   /* task ids   */
      int me, i, nproc, master, msgtype;
   int length_data_array;
   int n_variables;


// Tephra program variables:
//---------------------------
   // Grid dimensions
   double x_origin;
   double x_steps;
   double y_origin;
   double y_steps;
   double spacing;

   double arraygrid[SIZE_ARRAYGRID];  //array of output grid
   double variables[SIZE_VARIABLES];  //array of input variables
   double grid_dimensions[5];         //array of grid dimensions
```

```
// --------------------------------
// | Receive data from the master |
// --------------------------------

   mytid = pvm_mytid();      // who am I?
   msgtype = 0;              // setting msgtag to 0. Master should
                             // be also broadcasting with 0.
   pvm_recv( -1, msgtype );
// If tid = -1 and msgtag is
//      defined by the user, then pvm_recv will accept  a  message
//      from any process which has a matching msgtag.

      pvm_upkint(&nproc, 1, 1);
      pvm_upkint(tids, nproc, 1);
      pvm_upkint( &n_variables, 1, 1);
      pvm_upkdouble( &variables[0], n_variables, 1 );
      pvm_upkdouble( &grid_dimensions[0], 5, 1 );


   /* Determine which slave I am (0 -- nproc-1) */
   for( i=0; i<nproc ; i++ )
     if( mytid == tids[i] ){ me = i; break; }




// --------------------------------
// | Do calculations with data    |
// --------------------------------

// Find my portion of the grid
//    Parallel work is split up in grid function

   spacing=grid_dimensions[4];
   x_steps=grid_dimensions[1];
   x_origin=grid_dimensions[0];
```

```
   y_origin=grid_dimensions[2];
   y_steps=grid_dimensions[3];

//get grid -> arraygrid
   length_data_array = grid(arraygrid, variables, n_variables, x_origin, x_steps,y_origin, y_steps,
spacing,nproc,me);



// ---------------------------------
// |   Send results to master     |
// ---------------------------------
   pvm_initsend( PvmDataDefault );
   pvm_pkint( &me, 1, 1 );
   pvm_pkint( &length_data_array, 1, 1 );
   pvm_pkdouble( &arraygrid[0], length_data_array, 1 );
   msgtype = 50;
   master = pvm_parent();
   pvm_send( master, msgtype );


// ---------------------------------
// |   Program finished...exit    |
// ---------------------------------
   //pvm_initsend( PvmDataDefault );
   printf("end of program\n");
   pvm_exit();

   return 1; //
}
```

int grid(double data[], double input_variables[], int n_variables, double xorigin, double xsteps,

```
double yorigin, double ysteps, double spacing, int n_slaves, int n_me)
{
  int i,j;
  int x_step_count,y_step_count,number_of_runs;
  int grid_xsteps;
  int grid_ysteps;


  double xpos, ypos;

  grid_xsteps=xsteps +0;
  grid_ysteps=ysteps;
  xpos=xorigin;
  ypos=yorigin;

// ------------------------------------------------
// | Parallel:  Take my portion of variable set   |
// ------------------------------------------------

  number_of_runs = n_variables/n_slaves / NUMBER_OF_VARIABLES; // # of variables
                                  // divided by # of variables
                                  // in a run
  i=0;
  x_step_count=0;


     while(x_step_count<grid_xsteps)
     {

          y_step_count=0;

          while(y_step_count<grid_ysteps)
          {
            //move to next point in grid
            xpos=xorigin + (x_step_count*spacing);
            ypos=yorigin + (y_step_count*spacing);

            //at this point find tephra accumulation
            //     for each set of input variables
```

```
            for(j=0;j<number_of_runs;j++)
            {
               total_ash_mass=input_variables[0+((j+n_me)*12)];
               ymin   =input_variables[1+((j+n_me)*12)];
               ymax   =input_variables[2+((j+n_me)*12)];
               dmean  =input_variables[3+((j+n_me)*12)];
               dsigma =input_variables[4+((j+n_me)*12)];
               phiash =input_variables[5+((j+n_me)*12)];
               dfshape=input_variables[6+((j+n_me)*12)];
               dbeta  =input_variables[7+((j+n_me)*12)];
               w0     =input_variables[8+((j+n_me)*12)];
               h      =input_variables[9+((j+n_me)*12)];
               u      =input_variables[10+((j+n_me)*12)];
               udir   =input_variables[11+((j+n_me)*12)];


               data[i]=xpos;
               i++;
               data[i]=ypos;
               i++;
               data[i]=tephra_accumulation(xpos, ypos);
               i++;

            }//end of for

            y_step_count++;
          }//end of 'y' while

         x_step_count++;
      }//end of 'x' while
  return i;
}


double tephra_accumulation(double xpos, double ypos)
{
      double xsectionwidth;
      double ysectionwidth;
      double xmin, xmax;
```

```
double xstepwidth;
double ystepwidth;
double new_xpos, new_ypos;
double x0, y0, x1, xi, y1, yi,ans;
double total;
int xsteps, ysteps, yyi, xxi;

// define the limits of integration
//--------------------------------
// xmin is the elevation of the vent
// xmax is the total hight of the eruption column
// ymin and ymax are minimum and maximum
// grainsizes, respectively

 // do the integration over the eruption
 // column height in 300 steps
 xsteps = 300;

 // do the intergration over the particle
 // size reange in 100 steps
 ysteps = 100;

 xmin= 0.0;
 xmax =h;
 xsectionwidth = xmax - xmin;
 ysectionwidth = ymax - ymin;
 xstepwidth = xsectionwidth / xsteps;
 ystepwidth = ysectionwidth / ysteps;


 // initialize the summation for the
 // contribution of each d phi and
 // each dz to the total accumulation
 // of tephra on the ground at the
 // point xspace, yspace
 total  = 0.0;


//define the coordinate system with
```

```
// respect to wind direction
//--------------------------------
// here we are just assuming that
// the wind is varying N or south of
// a western direction, need to be genrealized

new_xpos = xpos * cos(udir) + ypos * sin(udir);
new_ypos = -xpos*sin(udir) + ypos*cos(udir);

//do the integration
//-------------------
for(yyi=1; yyi < ysteps; yyi++) {

    // define the small slice d phi

    yi = (double)yyi;
    y0 = ymin + yi * ystepwidth;
    y1 = ymin + (yi+(double)1.0) * ystepwidth;

    for(xxi=1; xxi < xsteps; xxi ++) {

      // define the small slice dz
      xi = (double)xxi;
      x0 = xmin + (xi) * xstepwidth;
      x1 = xmin + (xi+(double)1.0) * xstepwidth;

      // calculate the value of the ash accumulation  at
      // the corners  of the slice, and sum
      ans = f(new_xpos, new_ypos, x0, y0) + f(new_xpos, new_ypos, x1, y0) +
       f(new_xpos, new_ypos, x0, y1) + f(new_xpos, new_ypos, x1, y1);
      total += ans;
    }
}

// complete the integration by multiplying by dz and d phi
// and dividing by 4, since 4 corners were used
total *= xstepwidth * ystepwidth / 4.0;

//all finished: return the answer
```

```
    return total;
}


/* ---------- New Function Starts Here ---------- */
/* function phi2cm converts the ash diameter from
   units of phi to cm
*/

double phi2cm(double xx) {
  double cms;

  cms = 0.1 * 1.0 / pow(2, xx);
  return cms;
}

/* ---------- New Function Starts Here ---------- */
/* this function estimates the velocity
  as a function of height in the
  eruption column, given a
  linear decrease in velocity with
  height and an initial velocity
  of w0
*/

/* returns the upward particle velocity  */

double w(double zspace) {
  double particle_velocity;

  particle_velocity = w0 * (1.0 - zspace/h);

  // following only ocurrs as round-off error
 if (zspace > h) particle_velocity = 0.0;
  return particle_velocity;
}


/* ---------- New Function Starts Here ---------- */
```

```
/* this function calculates the probability
   density for diffusion of ash out of the
   eruption column. Some variation from Suzuki
   implemented to conserve mass

   depends on beta, w0, w(z), v0(xrho)
   where w0 is the eruption velocity in cm/s
   w(z) in the upward particle velocity at height z (cm/s)
   v0(xrho) is the particle settling velocity (cm/s)
   h is the column height (km)

   integrating fz from 0 to h should give unity

*/

double p(double zspace, double xrho) {
   double y, y0;
   double prob_dens_func;
   double voo;
   double demon1, demon2;
   voo = v0(xrho);
   y  = dbeta * w(zspace) / voo;
   y0 = dbeta * w0 / voo;
   demon1 = dbeta * w0 * y * exp(-y);
   demon2 = voo*h * (1.0-(1.0+y0) * exp(-y0));
   prob_dens_func = demon1/demon2;

   return prob_dens_func;
}



/* ---------- New Function Starts Here ---------- */
/* function ts determines the particle diffusion time const
   in the atmosphere
   where zspace = height of the particle, converted here from
            km to cm, this should be wrt sea-level,
            zspace is given here in cm because
            of the units of eddy diffusivity
```

```
      c = eddy diffusivity in the atmosphere
        given as in cgs, e.g., 400 cm/s^(5/2)

  returns the particle diffusion time
*/

double ts (double zspace) {
  double particle_diffusion_time;
  double demon1, demon2;

  zspace *= 1.0e+5;
  demon1  = 5.0 * zspace * zspace;
  demon2  = 288.0 * c;
  particle_diffusion_time = pow(demon1/demon2, 2.0/5.0);

  return particle_diffusion_time;
}


/* ----------------- New Function Starts Here ---------------------- */
/* function t calculates the particle fall time
  in the atmposphere
  where: zspace = height in the atmosphere, here in km
            wrt to sealevel, because the const.
            -0.0625 is in units of 1/km
*/
double t (double zspace, double xrho) {
  double particle_fall_time;
  double demon1, demon2;

  demon1 = (1.0 - exp(-0.0625*zspace)) / v0(xrho);
  demon2 = pow(demon1, 0.926);
  particle_fall_time = 0.752e6 * demon2;
  return particle_fall_time;
}


/* ----------------- New Function Starts Here ---------------------- */
/* return a value for the particle terminal velocity
```

```
  at sea level, given the particle diameter, passed as x
  note that the shape factor does not depend on x
  note that the ash dens does not depend on x
*/

double v0 (double xrho) {

  double velocity0;
  double demon1, demon2, demon3;

  demon1 = 9.0 * nua * pow(dfshape, -0.32);
  demon2 = 81.0 * nua * nua * pow(dfshape, -0.64);
  demon3 = 1.5 * phiair * phiash * gravity * xrho*xrho*xrho * sqrt((1.07-dfshape));

  velocity0 = phiash * gravity * xrho * xrho /
          (demon1 + sqrt(demon2 + demon3));

  return velocity0;
}



/* ----------------- New Function Starts Here ---------------------- */
/* this function calculates the expected fraction of particles
  in a given grainsize class (xrho) assuming a normal
  distribution in phi units about the mean, dmean,
  with standard deviation dsigma.
*/

double frho(double xrho) {
                                        double func_rho;
  double demon1, demon3, demon2;

  demon1   = 1.0 / (2.506628 * dsigma);
  demon3   = xrho - dmean;
  demon2   = exp(-demon3*demon3 / (2.0*dsigma*dsigma));
  func_rho = demon1 * demon2;

  if (func_rho >= 0.0) {
```

```
    return func_rho;
  }
  else {
    printf("error in ash size distribution - method frho");
    return func_rho;
  }
}
/* ----------------- New Function Starts Here ---------------------- */
/* function f calculates the epected mass of ash to accumulate in a
given location xspace, yspace (cm from the volcano) in gm / cm^3
*/

double f(double xspace, double yspace, double zspace, double xrho) {
  double ash_fall;
  double ashdiam;
  double t1, t2;
  double demon1, demon2, demon3, demon4;

  ashdiam = phi2cm(xrho);
  t1 = t(zspace, ashdiam);
  t2 = ts(zspace);
  demon1 =  5.0 * total_ash_mass * p(zspace,ashdiam) * frho(xrho);
  demon2 =  8.0 * PI * c * pow((t1+t2), 5.0/2.0);
  demon3 = -5.0 * ((xspace-u*t1) * (xspace-u*t1) + yspace * yspace);
  demon4 =  8.0 * c * pow(t1+t2, 5.0/2.0);

  ash_fall = demon1 / demon2 * exp(demon3/demon4);

  if (ash_fall >= 0.0) {

    return ash_fall;
  }
  else {
    printf("h: %f, zspace: %f, ashdiam: %f\n",h, zspace,ashdiam);
//    printf("p: %f, ahsdiam: %f, zspace: %f\n",p(zspace,ashdiam), ashdiam);
    return ash_fall;
    }
}
```

The resulting files:
Parallel results - for ten data points

```
Spawning 5 worker tasks ... SUCCESSFUL
Receiving from procces #3:
1000000.000000 0.000000 14.226482
1000000.000000 0.000000 22.533389
Receiving from procces #4:
1000000.000000 0.000000 22.533389
1000000.000000 0.000000 9.106846
Receiving from procces #1:
1000000.000000 0.000000 34.389264
1000000.000000 0.000000 12.952275
Receiving from procces #2:
1000000.000000 0.000000 12.952275
1000000.000000 0.000000 14.226482
Receiving from procces #0:
1000000.000000 0.000000 25.665705
1000000.000000 0.000000 34.389264
----------------------------------------
 tephra_master lasted 11 secs
5 hosts were used with the input file (data10)
----------------------------------------
```

Non-parallel results - for ten data points

```
1000000.000000  0.000000  25.665707
1000000.000000  0.000000  34.401527
1000000.000000  0.000000  12.971079
1000000.000000  0.000000  14.232705
1000000.000000  0.000000  22.537761
1000000.000000  0.000000  9.118118
1000000.000000  0.000000  7.981919
1000000.000000  0.000000  27.484349
1000000.000000  0.000000  11.955715
1000000.000000  0.000000  26.733252

# time elapsed 15.650000 secs/n
```

Parallel results - for ten data points

Spawning 5 worker tasks ... SUCCESSFUL
Receiving from procces #3:
1000000.000000 0.000000 14.226482
1000000.000000 0.000000 22.533389
Receiving from procces #4:
1000000.000000 0.000000 22.533389
1000000.000000 0.000000 9.106846
Receiving from procces #1:
1000000.000000 0.000000 34.389264
1000000.000000 0.000000 12.952275
Receiving from procces #2:
1000000.000000 0.000000 12.952275
1000000.000000 0.000000 14.226482
Receiving from procces #0:
1000000.000000 0.000000 25.665705
1000000.000000 0.000000 34.389264
---
tephra_master lasted 11 secs
5 hosts were used with the input file (data10)

Non-parallel results - for ten data points

1000000.000000 0.000000 25.665707
1000000.000000 0.000000 34.401527
1000000.000000 0.000000 12.971079
1000000.000000 0.000000 14.232705
1000000.000000 0.000000 22.537761
1000000.000000 0.000000 9.118118
1000000.000000 0.000000 7.981919
1000000.000000 0.000000 27.484349
1000000.000000 0.000000 11.955715
1000000.000000 0.000000 26.733252

time elapsed 15.650000 secs/n

Problems

1. If more than one user tries to use pvm from the same directory, it will only work for one of the users
2. Why is it that we CAN'T add the nodes from b01, but we CAN add b01 from the nodes?
3. If you start pvm on masaya with the following command:

    pvm -nb01

    It starts on 10.0.0.1 and you can add all the nodes from that network (but not the blue network). However, any program one runs will crash and will give an error code stating it couldn't start the executables on all the client machines. The only successful way to run pvm programs is to telnet to another machine on the beowulf. Start pvm with all the nodes (including b01) and then run your code from that machine.
    #3 is DONE
4. Unable to add b01 from any of the other cluster machines. The rsh command won't work from the machines.

    rsh b01 ls

    It gives the error message:

    PBINDPROC_DOMAIN: Domain not bound

    I guess we'll have to fix whatever is causing this error and not allowing us remote shell access.
5. When I am logged into a node (b02, for example), and then rlogin to the same node as root (in another window) and try to cd to ~beowulfuser, nothing is listed in the directory, even though there is something there.

    B02 login: root
    Password:
    B02:~ # cd ~beowulfuser
    B02:/usr/people/beowulfuser # ll
    total 0
    B02:/usr/people/beowulfuser #

    Or, if I telnet...

```
B02 login: root
Password:
B02:~ # cd ~beowulfuser
B02:/usr/people/beowulfuser # ll
total 0
B02:/usr/people/beowulfuser # ls
B02:/usr/people/beowulfuser #
```

But, if I rlogin as beowulfuser... everything is there...

```
masaya:~ # rlogin -l beowulfuser b02
Password:
beowulfuser@B02:~ > ll
total 392
-rw-r--r--  1 beowulfu users       92 May 30 17:31 10cluster
-rw-r--r--  1 beowulfu users    16862 May 31 08:54 data.out
```

6.I can't tell which nodes are actually doing the work. The code only says "process #0". In order to tell if all of the nodes are being used, I suggest placing code similar to the following in the tephra_slave code. This will get the hostname of each node that does the work.

```
#include "pvm3.h"

main()
{
    int ptid;
    char buf[100];

    ptid = pvm_parent();

    gethostname(buf, 64);

    pvm_initsend(PvmDataDefault);
    pvm_pkstr(buf);
    pvm_send(ptid, 1);

    pvm_exit();
```

```
        exit(0);
    }
```

#6 is DONE

7.

8.

```
From lauraconnor@netscape.net Wed May 17 11:16:37 2000
Date: 17 May 00 09:11:55 CDT
From: Laura Connor <lauraconnor@netscape.net>
To: nfrankli@colima.geophysics.swri.edu
Subject: Re: [Beowulf cluster]

Nathan, mmmmmmmmmmm, not quite sure what to tell you. Sometimes
it seems like this network stuff is a little bit of black magic.
When you are running Yast to configure network things, two files
get adjusted: /etc/re.config  and  /etc/route.conf
(assuming both network cards are recognized on bootup)

In order for the network stuff to work right, these two files need
have the right numbers. Once you get it right, it SHOULD always
boot up right.

Try running netscape to see if the other network card is working
right.

My guess is that something is not configured right with the
route.conf file.

If you cannot get it working, maybe you can send me the output from
the  route  command and the /etc/route.conf file. I can compare with
my configuration. And see if something is missing.

We reinstalled with YAST2 and both cards were recognized, but
we could only configure one. I had to go into Yast when the
install was complete to fix the numbers for the second card.
You have eth0 and eth1 and you have to make sure that the
networks have the correct IP addresses. I think Yast numbers
the networks based the the order it checks its drivers. Sometimes
you just need to switch the cables on the cards. I also changed
the option to use USER level nfs rather than KERNEL level nfs.
I had read that there had been some problems with using KERNEL level
nfs and I could not get it working (I was using SuSE 6.3, but
USER nfs worked very easily). To use nfs you have to edit the
/etc/exports  file.  For netscape you need to add the
GATEWAY = 129.162.79.1, and it needs to know nameserver IP's
and search domains (swri.edu and geophysics.swri.edu). These
are all stored in the /etc/rc.config file.

Well, this is just some info. I just talked to CHuck and hear
that things are working again. If it works make copies of the
```

configuration files and note which card is eth0 and eth1 and
what settings go with each card. You can also add your nodes
(their names and IP addresses) to the /etc/hosts file so
that you can just ping and telnet using their names. You do
not need to reboot in order for this to work. Just edit the file.
You can also add your other blue network nodes.

Good luck.

Laura

<nfrankli@colima.geophysics.swri.edu> wrote:
Mrs. Connor,

Thanks alot for your help on the Beowulf cluster. Chuck said you worked
with the Kernel in getting both ethernet cards up. I was wondering what
you had to change and configure differently to accomplish this.

I also just lost connection to the nodes in the cluster. I can
ping the 10.0.0.01 card but none of the nodes. This just happened after
I rebooted Masaya (ooops..). Is their a configuration file that I need to
run to get them back up or can you suggest any other ideas to try or
explore in order to regain connection with the cluster. (the routing table
looks correct)

Thanks again for your time and help!

Nathan

****

```
#
# Some people don't want SuSEconfig to modify the system. With this
# entry you can disable SuSEconfig completely.
# Please don't contact our support if you have trouble configuring your
# system after having disabled SuSEconfig.  (yes/no)
#
ENABLE_SUSECONFIG=yes

#
# If you say yes here, you will have some control over the system even
# if the system crashes for example during kernel debugging.
# Please consult /usr/src/linux/Documentation/sysrq.txt
# for further information.
#
ENABLE_SYSRQ=no

#
# This variable will overrule all LC-variables!!
# Again, ROOT_USES_LANG has to be set to YES in order
# to get any effect for the superuser.
#
RC_LC_ALL=""
```

```
#
# This defines the locale in which messages of programs and
# libraries with i18n-support should appear if a translated
# message catalog for the library or the program is installed.
# It also defines yes/no answers which are defined by the locale.
#
RC_LC_MESSAGES=""

#
# This defines the locale for character handling and classification.
# The locale defined here is used by the libc in functions which
# are used to qualify if this character is an charcater which may
# be used in an text string, if the character is e.g. lowercase
# and it defnes upper/lowercase-mapping of foreign characters
#
RC_LC_CTYPE=""

#
# This defines the locale for sorting strings and characters.
# The locale defined here is used by the libc in functions which
# are used to qualify if a character is befor or beyond an other
# character in the alphabet. Note: sort(1) doesn't use these
# functions, but other application such as databases may use it.
#
# To keep bash and possibly other apps from misbehavour because
# of mixed upper/lowercase sorting with locales, you should keep
# this at POSIX and just set it for the apps that need it:
#
RC_LC_COLLATE="POSIX"

#
# This defines the locale for date and time output formats.
# i.e.: 06/09/1999 vs. 09.06.1999
#
RC_LC_TIME=""

#
# This defines the locale for formatting and reading numbers.
# i.e.: 1,234.56 vs. 1.234,56
#
RC_LC_NUMERIC=""

#
# This defines the locale for formatting and reading money values.
#
RC_LC_MONETARY=""

#
# This defines if the user "root" should use the locale settings
# which are defined here.
#
ROOT_USES_LANG="no"

#
# SuSEconfig can mail reports (created by YaST or included in packages)
```

```
# to you.  Here you can set the address.  If you don't want reports to
# be send, then simply set it to "".
#
MAIL_REPORTS_TO="root"

#
# There are two levels of mailing.  If you set MAIL_LEVEL it to "warn"
# you only get the important mails.  If you set it to "all", you get
# logs also.
#
MAIL_LEVEL="warn"

#
# Which device is the modem ? (e.g. "/dev/ttyS1")
#
MODEM=""

#
# for some fonts the console has to be initialized with CONSOLE_MAGIC.
# CONSOLE_MAGIC can be empty or have the values "(B", ")B", "(K" or ")K".
#
CONSOLE_MAGIC=""


#
# keyboard repeat rate (2.0 - 30.0)
# keyboard delay time in ms (250, 500, 750, 1000)
# (If you want "kbdrate" to be executed, you have to set both of them.)
#
KBD_RATE=""
KBD_DELAY=""

#
# NumLock on? ("yes" or "no")
KBD_NUMLOCK="no"

#
# CapsLock on? ("yes" or "no")
KBD_CAPSLOCK="no"

#
# tty's for NumLock and CapsLock
# example: "tty1 tty2"
# "" for all tty's
#
KBD_TTY="tty1 tty2 tty3 tty4 tty5 tty6"

#
# start loopback networking? ("yes" or "no")
# (this will be needed for all rpc services)
#
START_LOOPBACK="yes"

#
# contains all indices of active PCMCIA network devices
```

```
#
NETCONFIG_PCMCIA=""
IPADDR_2=""
IPADDR_3=""
NETDEV_2=""
NETDEV_3=""
IFCONFIG_2=""
IFCONFIG_3=""

#
# setup dummy network device for IPADDR_0? this is useful for non permanent
# network connections (e.g. SLIP, PPP). Some software needs a connection
# to FQHOSTNAME (e.g. plp). (yes, no)
SETUPDUMMYDEV="no"

#
# Do you want the "dynamic IP patch" to be enabled at bootup? (yes/no)
#
IP_DYNIP=no

#
# Enable syn flood protection (see
/usr/src/linux/Documentation/Configure.help)
# (yes/no)
#
IP_TCP_SYNCOOKIES=yes

#
# runtime-configurable parameter: forward IP packets.
# Is this host a router? (yes/no)
#
IP_FORWARD=no

#
# if SORT_PASSWD_BY_UID is set to yes, SuSEconfig sorts your /etc/passwd
# and /etc/group by uid/gid.
#
SORT_PASSWD_BY_UID=no

#
# Used for News-Postings.
#
ORGANIZATION=""

#
# News server.
#
NNTPSERVER="news"

#
# space separated list of irc servers
#
IRCSERVER=""

#
```

```
# have mail daemon on SMTP port? ("yes" or "no")
# needed, if you receive email from other hosts via tcp/ip
# not needed, if you have a uucp-only host or only out-going email.
# If set to "yes", sendmail will be started as daemon.
# As uucp site, you can get along with "SMTP=no", if you make
# a "sendmail -q" call after each poll. (As rmail is queuing the mail only
# and not delivering it...)
#
SMTP="yes"

#
# some programs (e.g. lynx, arena and wget) support proxies, if set in
# environment.  SuSEconfig can add this environment variables to
# /etc/SuSEconfig/* (sourced by /etc/profile etc.) - See
# http://www.suse.de/Support/sdb_e/lynx_proxy.html for more details.
# Example: HTTP_PROXY="http://proxy.provider.de:3128/"
HTTP_PROXY=""

#
# Example: FTP_PROXY="http://proxy.provider.de:3128/"
#
FTP_PROXY=""

#
# Example: GOPHER_PROXY="http://proxy.provider.de:3128/"
#
GOPHER_PROXY=""

#
# Example: NO_PROXY="www.me.de, do.main, localhost"
#
NO_PROXY="localhost"


#
# start kernel daemon? ("yes" or "no")
#
START_KERNELD="yes"

#
# start cron daemon? ("yes" or "no")
# should be left unchanged to the default "yes" entry
#
CRON="yes"

#
# the kernel nfs-server supports multiple server threads
#
USE_KERNEL_NFSD_NUMBER="4"

#
# translates userid and goupid between server and client
# ("yes" or "no"). Needs to be started on NFS clients for
# certain special user-id mappings.
#
```

```
NFS_SERVER_UGID="yes"


#
# should imported NFS be reexported? ("yes" or "no")
#
REEXPORT_NFS="yes"

#
# when shutting down routing, all net connection can be closed (not useful
# in all cases).  If CLOSE_CONNECTIONS is set to "true" /sbin/init.d/route
# scans /proc to search for network connections and sends a term signal
# to the processes.
#
CLOSE_CONNECTIONS="false"

#
# start pcnfsd (for PCNFS clients; needs activated portmapper -
# see man pcnfsd) (yes/no)
#
START_PCNFSD=no

#
# start bwnfsd (pcnfs related) (yes/no)
#
START_BWNFSD=no

#
# pcnfsd and bwnfsd need spool directory for lpd. Set it here.
#
PCNFSD_LPSPOOL=/var/spool/lpd

#
# start rwhod? NOTE: rwhod broadcasts regularly, so dial
# on demand connections (ISDN and/or diald) might be established
# (yes/no)
#
START_RWHOD=no

#
# start routed (for dynamic routing - see man routed) (yes/no)
# ATTENTION:  starting routed causes net traffic every 30 seconds.
# If your host is connected to internet via dial-up it makes absolutely
# no sense to activate it.
#
START_ROUTED=no

#
# start the named (package bind)? You have to configure the named first,
# before you can start it (man named).
#
START_NAMED=no

#
# should updatedb (for locate) be started by cron.daily ("yes" or "no")
```

```
#
RUN_UPDATEDB=yes

#
# should mandb and whatis be recreated by cron.daily ("yes" or "no")
#
REINIT_MANDB=yes

#
# updatedb has a parameter "--localuser". it runs the find as this user.
# some people think, its a security hole to run it as root (because you
# can get information about directories you can not read normally). Some
# think its useful to hold all files in the database. If you want full
# information in locate db, set RUN_UPDATEDB_AS=root. If you want security
# use RUN_UPDATEDB_AS=nobody.
#
RUN_UPDATEDB_AS=nobody

#
# uptdatedb normally only scans local harddisks, but can include net paths
# in database as well. If you specify directories here, they will be scanned.
#
UPDATEDB_NETPATHS=""

#
# uptdatedb can skip directories for its database. The following parameter
# says which.
#
UPDATEDB_PRUNEPATHS="/S.u.S.E. /mnt /cdrom /tmp /usr/tmp /var/tmp /var/spool
/proc"

#
# search net paths as ? (e.g. nobody)
#
UPDATEDB_NETUSER=""

#
# old corefiles? should they be deleted ("yes" or "no")
# if set to no, cron.daily will tell you, if it finds old core files.
# please note, that this feature needs RUN_UPDATEDB to be set to "yes".
#
DELETE_OLD_CORE=no

#
# how old are 'old' core files? (in days)
#
MAX_DAYS_FOR_CORE=7

#
# should old preformated man pages be deleted (/var/catman)
# (yes/no)
#
DELETE_OLD_CATMAN=yes

#
```

```
# How old are OLD preformated man pages for you? (days)
#
CATMAN_ATIME=7


#
# we have a small script to generate usr/info/dir file. This needs perl..
# ("yes" or "no")
#
CREATE_INFO_DIR="yes"

#
# SuSEconfig can call chkstat to check permissions and ownerships for
# files and directories (using /etc/permissions).
# Setting to "set" will correct it, "warn" produces warnings, if
# something strange is found. Disable this feature with "no".
#
CHECK_PERMISSIONS=set

#
# SuSE Linux contains two different configurations for
# chkstat. The differences can be found in /etc/permissions.secure
# and /etc/permissions.easy. If you create your own configuration
# (e.g. permissions.foo), you can enter the extension here as well.
#
# (easy/secure local foo whateveryouwant).
#
PERMISSION_SECURITY="easy local"


#
# How long to store old log files. If set to 0, log files will be untouched.
# The log files below will be checked by cron.daily. The number
# after the name means the minimum size in k, the file has to have, before
# it will be backed up (root gets a mail, if it happens).
#
# /tmp/log_mg.* (1024), /var/log/wtmp (400), /var/log/isdn (4096),
# /var/lib/xdm/xdm-errors (200), /var/spool/uucp/Log (2048),
# /var/spool/uucp/Stats (1024), /var/log/debug (1024), /var/log/warn (1024),
# /var/log/messages (4096), /var/log/xferlog (4096),
# /local/www/logs/access_log (4096), /local/www/logs/error_log (1024)
# /var/adm/isdn.log (1024), /var/log/isdncalls (1024)
#
MAX_DAYS_FOR_LOG_FILES=365

#
# cron.daily can make backup the rpm database. Set the path here, and
# cron.daily will make backup everytime it is called and the db has
# changed.  This backups are recommended.  If you don not want this
# feature, set it to "".
#
RPMDB_BACKUP_DIR=/var/adm/backup/rpmdb

#
# here you can set the maximum number of backup files for the rpm
```

```
# database.
#
MAX_RPMDB_BACKUPS=5

#
# cron.daily can check for old files in tmp-dirs. It will delete all files
# not accessed for more than MAX_DAYS_IN_TMP. If MAX_DAYS_IN_TMP is not set
# or set to 0, this feature will be disabled.
#
MAX_DAYS_IN_TMP=0

#
# You can specify in TMP_DIRS_TO_CLEAR, which directories have to be
# searched for old files, to be deleted.
#
TMP_DIRS_TO_CLEAR="/tmp /var/tmp"

#
# In OWNER_TO_KEEP_IN_TMP, you can specify, whoms file shall not be deleted.
#
OWNER_TO_KEEP_IN_TMP="root"

#
# Do you want to have "." in root path?  This is not recommended, but
# many people do prefer it (yes/no).
#
CWD_IN_ROOT_PATH="no"


#
# If you want to allow root logins from other machines, set ROOT_LOGIN_REMOTE
# to "yes".
#
ROOT_LOGIN_REMOTE="yes"

#
# If you want that new passwords will be checked through cracklib,
# set PASSWD_USE_CRACKLIB to "yes".
#
PASSWD_USE_CRACKLIB="no"

#
# Some packages by SuSE include dynamically linked motif progs as well
# as statically linked (*.SuSE-dynamic resp. *.SuSE-static). SuSEconfig
# can analyze your system and link the matching program to *.  If you
# set this to "clean", the other binary will be deleted. (no/link/clean)
#
HOW_TO_HANDLE_COMMERCIAL_LIBS=link

#
# CONSOLE_SHUTDOWN determines how ctrl-alt-del is handled.
# Attention: CHECK_INITTAB has to be set to yes, to activate this feature.
# (ignore/reboot/halt)
#
CONSOLE_SHUTDOWN=reboot
```

```
#
# run the Name Service Caching Daemon at boot time? (yes/no)
#
START_NSCD=yes

umask 022

#
# Attention! This variable PATH is NOT setting the PATH for user or root
# shells. It is only used internally for /sbin/init.d/*, SuSEconfig and
# cron.daily. Please do NOT change PATH here.
#
PATH=/sbin:/bin:/usr/sbin:/usr/bin


##
## Formating the boot script messages:
##   The boot scripts should use the variables rc_done and rc_fail to
##   symbolize their success.  See /sbin/init.d/skeleton for an example
##   how to use these variables.
##   rc_done_up and rc_failed_up do the same as rc_done and rc_failed
##   but one line above (usefull for starting daemons who talk to user).
##   The variable rc_reset is used by the master resource control script
##   /sbin/init.d/rc to turn off all attributes and switch on the standard
##   character set.
##
##    \033          is just ascii ESC
##    \033[<NUM>G   move to column <NUM>
##    \033[1m       switch bold on
##    \033[31m      switch red on
##    \033[32m      switch green on
##    \033[33m      switch yellow on
##    \033[m        switch color/bold off
##
    rc_done="\033[71G\033[32mdone\033[m"
  rc_failed="\033[71G\033[31m\033[1mfailed\033[m"
 rc_skipped="\033[71G\033[1mskipped\033[m"
 rc_done_up="\033[1A$rc_done"
rc_failed_up="\033[1A$rc_failed"
  rc_unused="\033[71G\033[1munused\033[m"
   rc_reset="\033[m\017"
#
# Should the ATD (at daemon) be started, for the execution of at jobs?
(yes/no)
#
START_ATD=yes
#
# Update groff DESC to get page sizes correct? (yes/no)
#
# If the correct page size isn't found in your printcap
# you can set GROFF_PAGESIZE to the following values
#
#      letter, legal, a4, or b5
#
# supported by both groff *and* ghostscript
```

```
#
UPDATE_GROFF_CONF=yes
GROFF_PAGESIZE=
#
# Start the ident daemon in multi-user? ("yes" or "no")
# Identd looks up specific TCP/IP connections and returns either
# the user name or other information about the process that owns
# the connection.
#
START_IDENTD="yes"
#
# Start the rpc.rusersd daemon in multi-user? ("yes" or "no")
# rpc.rusersd is a server which returns information about users currently
# logged in to the system.
#
START_RUSERSD="no"
#
# Start the rpc.rstatd daemon in multi-user? ("yes" or "no")
# rpc.rstatd is a RPC daemon which collects performance statistics from
# kernel on serving machine.
#
START_RSTATD="no"

#
# Should PCMCIA service be started at boottime? (yes/no)
#
START_PCMCIA="yes"

#
# PCMCIA: This variable determines the used chipset. Valid Values are
# "i82365" or "tcic". If it is left empty, pcmcia will not be startet
# at boot up.
#
PCMCIA=""

#
# PCMCIA_PCIC_OPTS - socket driver timing parameters here. These
# parameters are described in "man i82365" (or "man tcic").
# e.g.: PCMCIA_PCIC_OPTS="par1=val1 par2=val21,val22 par3=val3"
# For more information, look for "PCIC_OPTS" in the PCMCIA-HOWTO.
# You can find it under /usr/doc/packages/pcmcia.
#
# If PCMCIA locks your System, try the following option with a list
# of free interrupts (and which won't be used later)
# e.g. PCMCIA_PCIC_OPTS="irq_list=3,4,5,7,9,10,11"
#
PCMCIA_PCIC_OPTS=""


#
# PCMCIA_CORE_OPTS - Put pcmcia_core options here. These options
# are described in "man pcmcia_core"
# For more information, look for "CORE_OPTS" in the PCMCIA-HOWTO.
# You can find it under /usr/doc/packages/pcmcia.
PCMCIA_CORE_OPTS=""
```

```
#
# May SuSEconfig modify your perllocal.pod? (yes/no)
#
CREATE_PERLLOCAL_POD="yes"

#
# May SuSEconfig run h2ph when kernelsources have changed
#
GENERATE_PERL_SYSTEM_INCLUDES="yes"

#
# Default loglevel for klogd
#
KERNEL_LOGLEVEL=1

#
# if not empty: parameters for syslogd
# for example SYSLOGD_PARAMS="-r -s my.dom.ain"
#
SYSLOGD_PARAMS=""

# Start apmd? (yes/no)
# apmd watches the status of the battery and triggers certain actions
# when particular events occur. You can customize this actions in the
# file /etc/rc.config.d/apmd.rc.config.
#
START_APMD=no
#
# Should isapnp be used to initialize your PNP at bootup?  (yes/no)
#
START_ISAPNP=yes
#
# Should the Apache httpd be started at bootup? (yes/no)
#
START_HTTPD=yes
#
# The name of the central server for the Online documentation
# This should be a fully qualified host name, e.g. host.domain.top
#
DOC_HOST="localhost"

#
# Set this to yes on the central documentation server
# Then the online-help-system indices are automatically adjusted
# and access to the http-rman service is allowed
#
DOC_SERVER="yes"

#
# List auf host/domain patterns for use with /etc/hosts.allow
# access restrictions on http-rman, e.g. ".mydomain.top" to
# allow access from all hosts of domain mydomain.top
#
DOC_ALLOW="LOCAL"
#
```

```
# Start printer daemon lpd? (if you use plp, you can also disable it here
# an enable it in /etc/inetd.conf) (yes/no)
#
START_LPD=yes

# Set this to native, if you want *real* Multithreading,
# e.g. in combination with SMP-systems

JAVA_THREADS_TYPE="green"

#
# KDM_SHUTDOWN determines who will be able to shutdown the
# system in kdm. Valid values are: "root", "all", "none", "local",
#
KDM_SHUTDOWN="Auto"

#
# space separated list of users for which icons should be shown in KDM
# if empty, then take system defaults
#
KDM_USERS=""

#
# path of jpeg or xpm image to be shown in the background of kdm
# or background color (color names from /usr/X11R6/lib/X11/rgb.txt)
#
KDM_BACKGROUND=/opt/kde/share/wallpapers/paper01.jpg

#
# title string of kdm, special string HOSTNAME displays name of computer
#
KDM_GREETSTRING=""

#
# Pixmap which appears on the top of a KDE window, do not specify full path
# path name; it will be searched in KDE IconPath; no Gimmick if empty
# just try: "chamelia.xpm"
#
KWM_GIMMICK_PIXMAP=""
#
# SuSEconfig.wm can create a .fvwm2rc, .fvwmrc, .bowmanrc, .fvwm2rc95,
# .mwmrc, .ctwmrc, depending on the installed packages. If
# you want your systemwide wm config files to be updated after install
# / removal of packages set SUSEWM_UPDATE to "yes", otherwise to "no"
#
SUSEWM_UPDATE="yes"

#
# This is the (space separated) list of window managers for which you
# want to generate the config file. Valid values are:
#   "fvwm", "fvwm2", "fvwm95", "bowman", "mwm", "ctwm", "kwm", "all".
# Default setting is "all" which is for generating files for all wms.
#
SUSEWM_WM="all"
```

```
#
# Your fvwm2/95 is slow? Don't want small pixmaps in menus? So set
# SUSEWM_XPM to "no", if pixmaps in menus are wanted set it to "yes",
# which is the default value. The package 3dpixms has to be installed.
#
SUSEWM_XPM="yes"
#
# start idled? ("yes" or "no")
# idled kills idle logins after a while
#
START_IDLED="no"

#
# ifconfig (eg. "10.10.1.3 broadcast 10.10.1.31 netmask 255.255.255.224",
"bootp" or "dhcpclient"
#
IFCONFIG_0="129.162.79.56 broadcast 129.162.79.255 netmask 255.255.255.0 up"

#
# IP Address
#
IPADDR_0="129.162.79.56"

#
# Network device name (e.g. "eth0")
#
NETDEV_0="eth0"

#
# From:-Line in email and News postings
# (otherwise the FQDN is used)
#
FROM_HEADER="YAST_ASK"

#
# Shall auto mount daemon autofs be started? (yes/no)
#
START_AUTOFS="yes"

#
# autofs daemon options (e.g. --timeout 60)
#
AUTOFS_OPTIONS=""

#
# Should NIS(YP) be used for autofs (yes/no)
#
USE_NIS_FOR_AUTOFS=yes

#
# Should NIS+ be used for autofs (yes/no)
#
USE_NISPLUS_FOR_AUTOFS=no
#
# some interfaces need time to initialize. Add the latency time in seconds
```

```
# so these can be handled properly
#
DHCLIENT_SLEEP=1
# start i4l? ("yes" or "no")
#   see: /usr/doc/packages/i4l
#
I4L_START="no"

# IrDA is the infrared interface often found on laptops. If you like to
# activate support for the infrared port, please configure IrDA for your
# system by editing/etc/irda/drivers.  Please take a look into the IrDA
# HOWTO, available in /usr/doc/howto/en/IR-HOWTO.gz on how to configure it.
#
START_IRDA=no

# Currently the UART (SIR) mode is supported in the normal configuration. If
# you like to have FIR (4 MBit/s, only for a few chipsets supported), you
# should edit the file /etc/irda/drivers. The variable IRDA_PORT sets the
# used UART port, variable IRDA_IRQ sets the used interrupt.
#
IRDA_PORT=/dev/ttyS1

IRDA_IRQ=3
#
# Start the pppd with wvdial as a chat-script in demand mode
# This results in a dial on demand connection to the isp
# via modem
#
PPPD_DOD_START="no"
#
# Should /etc/yp.conf be created automatically? ("yes" or "no")
# If set to yes /etc/passwd and /etc/group will also be checked.
#
CREATE_YP_CONF="yes"

#
# NIS-domain, ask the admin of the NIS or NIS+ Server.
#
YP_DOMAINNAME=""

#
# YP-Servers. Attention! You've to fill in IP adresses, a name or a nick
# name here. It must be defined in /etc/hosts (case sensitive).
# DNS does not work with ypbind. (e.g. "192.168.116.11 192.168.7.7")
#
YP_SERVER=""

#
# start the ypbind daemon for NIS ? ("yes" or "no")
# This entry is ignored if no YP_DOMAINNAME is set.
# In this case, ypbind is not started.
#
START_YPBIND="yes"
#
# Load this console font upon bootup:
```

```
#
CONSOLE_FONT=""

#
# Does your console font need a screenmap? Insert it into CONSOLE_SCREENMAP.
#
CONSOLE_SCREENMAP=""

#
# some fonts/keymap need a unicode map (TRANSLATION in former releases).
#
CONSOLE_UNICODEMAP=""

#
# run kde or gnome as default wm
#
DEFAULT_WM="kde"

#
# Set to "-u" if your system clock is set to GMT, otherwise "".
#
GMT=""

#
# gpm will be started in runlevel 2 with this parameters
#
GPM_PARAM="-t ps2 -m /dev/mouse"

#
# The module for initrd during boot
#
INITRD_MODULES="usbcore eepro100"

#
# Keyboard mapping for the text console.
#
KEYTABLE="us.map.gz"

#
# The language setting for the old YaST
#
LANGUAGE="english"

#
# The Unix device for the mouse
#
MOUSE="/dev/psaux"

#
# Should the NFS server be started on this host? ("yes" or "no")
#
NFS_SERVER="yes"

#
# The language setting for the old YaST
```

```
#
RC_LANG="en_US"

#
# Should gpm be started on this machine? ("yes" or "no")
#
START_GPM="yes"

#
# Start the inet daemon in multi-user? ("yes" or "no")
#
START_INETD="yes"

#
# Start portmap? ("yes" or "no") Needed for nfsserver or NIS
#
START_PORTMAP="yes"

#
# Start USB module ?
#
START_USB="no"

#
# The timezone setting
#
TIMEZONE="US/Central"

#
# should the kernel based NFS server be started on this host
#
USE_KERNEL_NFSD="no"
#
# Set to yes to allow SuSEconfig to make changes to /etc/inittab
#
CHECK_INITTAB="yes"

#
# Here you can set the default Display manager (kdm/xdm/console).
#
DISPLAYMANAGER="kdm"
#
# Should SuSEconfig sort your /etc/hosts? ("yes" or "no")
#
BEAUTIFY_ETC_HOSTS="yes"

#
# Should SuSEconfig do some checks and modifications in /etc/hosts? ("yes" or
"no")
#
CHECK_ETC_HOSTS="yes"

#
# Should SuSEconfig create and check the /etc/host.conf? ("yes" or "no")
#
```

```
CREATE_HOSTCONF="yes"

#
# Shall SuSEconfig maintain /etc/resolv.conf (needed for DNS)? ("yes" or "no")
#
CREATE_RESOLVCONF="yes"

#
# Shall the dynamic host configuration (DHCP) client be started? ("yes" or
"no")
#
DHCLIENT="no"

#
# The fully qualfied hostname of this computer. (e.g. "linux.suse.de")
#
FQHOSTNAME="masaya.geophysics.swri.edu"

#
# ifconfig (eg. "10.10.1.3 broadcast 10.10.1.31 netmask 255.255.255.224",
"bootp" or "dhcpclient"
#
IFCONFIG_1="10.0.0.1 broadcast 10.0.0.255 netmask 255.255.255.0 up"

#
# IP Address
#
IPADDR_1="10.0.0.1"

#
# Space separated list of nameservers that should be used for /etc/resolv.conf
#
NAMESERVER="129.162.26.41 129.162.176.2"

#
# Number of network devices: "_0" for one, "_0 _1 _2 _3" for four card
#
NETCONFIG="_0 _1"

#
# Network device name (e.g. "eth0")
#
NETDEV_1="eth1"

#
# Domain searchlist that should be used in /etc/resolv.conf
#
SEARCHLIST="swri.edu geophysics.swri.edu"
#
# Start DQS Queue Master?
#

START_QMASTER="no"
```

```
#
# Start DQS Execution Daemon?
#

START_DQSEXECD="no"

**********************************************************************


#
# /etc/route.conf
#
# In this file you can configure your static routing...
#
# This file is read by /sbin/init.d/route.
#
#
# Destination              Dummy/Gateway   Netmask              Device
#
# Examples:
#
# Net devices
# 193.141.17.192           0.0.0.0         255.255.255.192      eth0
#
# Gateways
# default                  Riemann
# 0.0.0.0                  193.141.17.193
#
#
# Host behind Gateway
# 193.141.17.142           193.141.17.193  255.255.255.255
#
# Net behind a Gateway
# 193.141.17.145           193.141.17.193  255.255.255.0
#
# Multicast route for e.g. eth0. IP multicasting, forwarding and perhaps
# multicast routing in kernel should be enabled.   More information will
# be found in the NET-3-HOWTO.     Most people do NOT need this feature.
#
# 224.0.0.0       0.0.0.0      240.0.0.0   eth0
#
# ISDN (i4l)
# 192.168.0.1              0.0.0.0         255.255.255.255       ippp0
# default                  192.168.0.1

129.162.79.0             0.0.0.0                 255.255.255.0           eth0
10.0.0.0                 0.0.0.0                 255.255.255.0           eth1

default                  129.162.79.1
```

Scientific Notebook 115E
20-1402-461
Inititals: CC

Chuck Connor
July 13, 2000

The following pages include / document html for seissmic hazard assessment.

Calculation of the Acceleration Spectral Density

The purpose of this document is to provide calculations of the acceleration spectral density function using
empirical constants and site-derived parameters. The parameters are described in the following and are tuned to
western US sites. See work by Boore (1986, BSSA 76: 43-64). This work was performed in part to evaluate
seismic hazard assessments by DOE and its contractors, for the U.S. Nuclear Regulatory Commission.

The acceleration spectral density is written as: Source term * Path term * Site term / g

where:

Source term = the operator that depends of source conditions
Estimating the Source Term

The source term describes the contribution of the earthquake and surrounding rock to the acceleration at the site. Here, the source term is approximated as a point source.
See work by Boore (1986, BSSA 76: 43-64), Brune (1970, JGR 75: 4997-5009), and Hanks and Kanamori (1979, JGR 84: 2348-2350).

The Source term is written as: (C * f2* Mo) / [1 + (f / fc)2]
where:
C = 1/(rho* beta3) * 2 * 0.55 * 1/sqrt(2) * pi
f = frequency
fc = source corner frequency
Mo = seismic moment
pi = 3.14159
rho = rock density at the source
beta = shear wave velocity at the source

The Source corner frequency is: beta * [sigma / (8.44 * Mo)] 1/3
where:
sigma = stress drop parameter

Scientific Notebook 115E
20-1402-461
Inititals: CC

Chuck Connor
July 13, 2000

The seismic moment is related to the earthquake magnitude by: log Mo = 1.5 Mw + 16.1
where:
Mw = seismic magnitude

Path term = the operator that depends on ray path conditions
Estimating the Path Operator

The path operator describes the frequency dependent attenuation between the source and the site. Here, the path operator depends on a frequency dependent quality factor, which in turn depends on two empirically derived parameters, Qo and n. For the INEEL site, 100 <= Qo <= 660, and 0.3 <= n <= 0.8. These values seem typical for western US studies. See work by Boore (1986, BSSA 76: 43-64), Brune (1970, JGR 75: 4997-5009), and Ou and Herrmann (1990, BSSA 80: 1397-1417).

The Path Operator is written as: 1 / R * exp[(-pi*f*R)/(beta* Q(f))]
where:

R = distance from source to site
f = frequency
pi = 3.14159
Q(f) = frequency dependent quality factor
beta = shear wave velocity at the source

The quality factor is: Qo * f^n
where:
Qo = regional crustal code
n = empirical parameter

Site term = the operator that depends on site near-surface conditions
g = gravity (980 cm/s2)

For graphs of the Source term, Path term, and Site term:
Source term Graph
Path Operator Graph
Site Operator Graph

****

```
import java.awt.*;
import java.applet.*;
```

```
import graph.*;
/*
**************************************************************************
**
**                Applet SourcetermGraph
**
**************************************************************************
**   Written by Chuck Connor May, 2000
**
**   This program calculates the expected acceleration spectral density
**   function for given input parameters.
**************************************************************************
*
* This applet uses the Accspectrum class
* and plots the result.
*
**************************************************************************/

public class SourcetermGraph extends Applet {

   G2Dint graph        = new G2Dint();   // Graph class to do the plotting
   //   SpecialFunction sp;
   Axis xaxis;
   Axis yaxis;
   DataSet data;

   TextField magnitudeinput    = new TextField(10);      // earthqauke magnitude input
   TextField betainput   = new TextField(10);     //shear wave velocity at source input
   TextField rhoinput    = new TextField(10);     // rock density at source input
   TextField sigmainput    = new TextField(10);     // stress drop parameter input

   Button plot         = new Button("Calculate"); // Button to plot it.

    public void init() {
      Label title          = new Label("Acceleration Spectral Density",Label.CENTER);
      Panel panel          = new Panel();
      GridBagLayout gridbag  = new GridBagLayout();
      GridBagConstraints  c  = new GridBagConstraints();
      Font font            = new Font("Helvetica",Font.PLAIN,14);
```

```
title.setFont(new Font("TimesRoman",Font.PLAIN,24));


setLayout(new BorderLayout() );
add("North",title);
add("Center",panel);

magnitudeinput.setText(getParameter("MAGNITUDE"));
betainput.setText(getParameter("BETA"));
rhoinput.setText(getParameter("RHO"));
sigmainput.setText(getParameter("SIGMA"));

panel.setLayout(gridbag);

Label magnitudelabel   = new Label("Earthquake Magnitude");
Label betalabel = new Label("Shear Wave Velocity");
Label rholabel   = new Label("Rock Density");
Label sigmalabel   = new Label("Stress Drop Parameter");



magnitudelabel.setFont(font);
betalabel.setFont(font);
rholabel.setFont(font);
sigmalabel.setFont(font);


magnitudeinput.setFont(font);
magnitudeinput.setBackground(Color.lightGray);

betainput.setFont(font);
betainput.setBackground(Color.lightGray);
rhoinput.setFont(font);
rhoinput.setBackground(Color.lightGray);
sigmainput.setFont(font);
sigmainput.setBackground(Color.lightGray);


plot.setFont(font);
```

```
plot.setBackground(Color.lightGray);

c.weightx = 1.0;
c.weighty = 1.0;
c.gridwidth = 4;
c.gridwidth=GridBagConstraints.REMAINDER;
c.fill  =  GridBagConstraints.BOTH;

gridbag.setConstraints(graph,c);

c.fill  =  GridBagConstraints.NONE;
c.weightx=0.0;
c.weighty=0.0;
c.gridheight=1;

c.gridwidth=1;
c.anchor = GridBagConstraints.EAST;
gridbag.setConstraints(magnitudelabel,c);

c.anchor = GridBagConstraints.CENTER;
c.gridwidth=GridBagConstraints.RELATIVE;
c.fill  =  GridBagConstraints.HORIZONTAL;
gridbag.setConstraints(magnitudeinput,c);

c.fill = GridBagConstraints.NONE;
c.gridwidth=GridBagConstraints.REMAINDER;

gridbag.setConstraints(plot,c);


c.gridwidth=1;
c.anchor = GridBagConstraints.EAST;
gridbag.setConstraints(betalabel,c);
c.gridwidth=2;
c.anchor = GridBagConstraints.WEST;
c.gridwidth=GridBagConstraints.REMAINDER;
gridbag.setConstraints(betainput,c);

c.gridwidth=1;
```

```
c.anchor = GridBagConstraints.EAST;
gridbag.setConstraints(rholabel,c);
c.gridwidth=2;
c.anchor = GridBagConstraints.WEST;
c.gridwidth=GridBagConstraints.REMAINDER;
gridbag.setConstraints(rhoinput,c);

c.gridwidth=1;
c.anchor = GridBagConstraints.EAST;
gridbag.setConstraints(sigmalabel,c);
c.gridwidth=2;
c.anchor = GridBagConstraints.WEST;
c.gridwidth=GridBagConstraints.REMAINDER;
gridbag.setConstraints(sigmainput,c);



panel.add(graph);
panel.add(magnitudelabel);
panel.add(magnitudeinput);
panel.add(plot);

panel.add(betalabel);
panel.add(betainput);
panel.add(rholabel);
panel.add(rhoinput);
panel.add(sigmalabel);
panel.add(sigmainput);



xaxis = graph.createXAxis();
xaxis.setTitleText("Period (sec)");
xaxis.setTitleFont(new Font("Helvetica",Font.PLAIN,18));
xaxis.setLabelFont(new Font("Helvetica",Font.PLAIN,14));


yaxis = graph.createYAxis();
yaxis.setTitleText("Source Term");
```

```
    yaxis.setTitleFont(new Font("Helvetica",Font.PLAIN,18));
    yaxis.setLabelFont(new Font("Helvetica",Font.PLAIN,14));


    data = new DataSet();

    xaxis.attachDataSet(data);
    yaxis.attachDataSet(data);
    graph.attachDataSet(data);


    graph.setDataBackground(new Color(230,240,240));
    graph.setBackground(new Color(240,250,250));

    plot();

  }

  void plot() {


    double mag, beta, rho, sigma;

    // the following are not used in the source erm model
    // but are used to instantiate the class

    double dist = 0.0;
    double nu = 0.0;
    double kappa = 0.0;
    double qo = 0.0;

    double freq;
    boolean error = false;

    try {
       mag = Double.valueOf(magnitudeinput.getText()).doubleValue();
    } catch(Exception e) {
       this.showStatus("Error with magnitude!");
       System.out.println("magnitude error "+e.getMessage());
```

```
        return;
      }

      try {
        beta = (Double.valueOf(betainput.getText()).doubleValue());
      } catch(Exception e) {
         this.showStatus("Error with beta!");
         System.out.println("beta error "+e.getMessage());
         return;
      }

      try {
        rho = Double.valueOf(rhoinput.getText()).doubleValue();
      } catch(Exception e) {
         this.showStatus("Error with rho value!");
         System.out.println("rho error "+e.getMessage());
         return;
      }
     try {
        sigma= Double.valueOf(sigmainput.getText()).doubleValue();
      } catch(Exception e) {
         this.showStatus("Error with sigma value!");
         System.out.println("sigma error "+e.getMessage());
         return;
      }



      int j, kount;

      double d[] = new double[500];

      Accspectrum accmodel = new Accspectrum(beta, rho, sigma, dist, mag, qo, nu, kappa);

     j =0;
     for (kount=1; kount <= 100; kount++, j +=2) {
       try {
          this.showStatus("Calculating point: "+ kount);
```

```
            freq = (double)kount/2.0 ;
            d[j] = (1.0/freq);
            d[j+1] = (accmodel.ptsource(freq));

         } catch (Exception e) {error = true;}

      }



      data.deleteData();

      try {
          data.append(d,100);
      } catch(Exception e) {
         this.showStatus("Error while appending data!");
         System.out.println("Error while appending data!");
         return;
      }


      graph.repaint();
   }



   public boolean action(Event e, Object a) {

      if(e.target instanceof Button) {
         if( plot.equals(e.target) ) {
            plot();
            return true;
         }
      }


      return false;
   }
```

```
}

**********************************************************************

public class Accspectrum {
 public double beta, rho, sigma, distance, mw, qo, nu, kappa;

   // all of the following input parameters are used to estimate
   // the acceleration spectral density, Units of all
   // of the following are in gm-cm-s, with the exception of equake magnitude
   // returns acceleration in gravity (g) where 1 g = 980 cm s^-2

   // beta is the  shear wave velocity at the source
   // rho is the rock density at the source
   // sigma is the stress parameter
   // distance is the distance from source to site
   // mw is the earthquake magnitude
   // qo and nu are parameters used to estimate the frequency dependent quality factor model
   // kappa is the site and distance dependent intrinsic attenuation factor

 public Accspectrum (double beta, double rho, double sigma, double distance, double mw,
                double qo, double nu, double kappa) {
   this.beta = beta;
   this.rho = rho;
   this.sigma = sigma;
   this.distance = distance;
   this.mw = mw;
   this.qo = qo;
   this.nu = nu;
   this.kappa = kappa;

  }

 public  double log10(double xx) {
```

```
    // calculates the log base 10 of input
      double logxx;
      logxx = Math.log(xx)/Math.log(10.0);
      return logxx;
    }

  public double mw2mo(double mw) {
    // calculates the seismic moment, mo
    // given mw
    // using the relation developed in
    // Hanks and Kanamoru, 1979, A moment
    // magnitude scale, JGR, 84:2348-2350
    // with the following formula, mo - dynes-cm

      double mo;
      mo = Math.pow(10, 1.5*mw + 16.1);
            return  mo;
  }

    public double fc(double beta, double sigma, double mw) {
      //calculates the source corner frequency
      //given beta = shear wave velocity at the source
      // mw is the point source focal depth
      // sigma is the high frequency stress parameter
      // from Brune, 1970, Tectonic stress and the spectra of
      // seismic shear waves from earthquakes, JGR 75: 4779-
      // 5009. Also see Brune, J.N., 1971, Correction, JGR 76: 5002.

      double corner;
      corner = beta * Math.pow(sigma/(8.44*mw2mo(mw)), 1.0/3.0);
      return corner;
    }

    public double eqconst(double beta, double rho) {

      // eqconst returns an equation constant that contains
      // rho - the source region density
      // beta - the source region shear wave velocity
      // a free surface effect (factor of 2)
```

```
    // source radiation pattern averages over sphere 0.55
    // partition of energy into two horizontal components
    // (factor of 1/sqrt(2))
    // see Boore, DM,1986, Short period P and S-wave radiation
    // from large earthquakes: Implications for spectral scaling
    // relations, BSSA 76: 43-64.

    double c, betaroot;

    c = 1.0/(beta*beta*beta*rho) * 2.0 * 0.55 * 1.0/1.41421 *1.0/(4*Math.PI);
    return c;
  }

  public double pftf(double kappa, double freq) {

    // pftf returns the value of a high frequency
    // tuncation filter for a given input frequency
    // freq. kappa is a epicenter - site distance parameter
    // that depends on distance, shear wave velocity
    // and the quality factor average to some depth H
    // at the site. Kappa is frequency independent but
    // does depend (HOW?) on distance.

    double p;
    p = Math.exp(-Math.PI*kappa*freq);
    return p;
  }

  public double qf(double qo, double nu, double freq) {

    // qf returns the frequency dependent quality factor
    // seismic attenuation from the source to the site is
    // estimated from qo and nu parameters.

    double quality_factor;
    quality_factor = qo * Math.pow(freq, nu);
    return quality_factor;
  }
```

```
public double amp(double freq) {
   // place holder until the true
   // story on the amp factor
   // is determined

   double log10freq;
   double amplification_factor;
   double xmax, xmin, ymax, ymin, y, m;
   double betaterm, rhoterm;
   double rho_site = 2.0;

   rhoterm = Math.sqrt(rho/rho_site);

   log10freq = log10(freq);

   xmax=0.0;
   xmin = 0.0;
   ymin = 0.0;
   ymax = 0.0;

   if (log10freq > -2.0 && log10freq <= -1.0) {
      xmin = -2.0;
      xmax = -1.0;
      ymin = 0.01;
      ymax = 0.01;
   }

   if (log10freq > -1.0 && log10freq <= -0.5) {
      xmin = -1.0;
      xmax = -0.5;
      ymin = 0.01;
      ymax = 0.04;
   }
   else if (log10freq > -0.5 && log10freq <= 0.0) {
      xmin = -0.5;
      xmax = 0.0 ;
      ymin = 0.04;
      ymax = 0.13;
   }
```

```
    else if (log10freq > 0.0  && log10freq <= 0.5) {
       xmin = 0.0 ;
       xmax =  0.5;
       ymin = 0.13;
       ymax = 0.34;
    }

    else if (log10freq > 0.5  && log10freq <= 1.0) {
       xmin = 0.5;
       xmax = 1.0 ;
       ymin = 0.34;
       ymax = 0.37;
    }
    else if (log10freq > 1.0  && log10freq <= 2.0) {
       xmin = 1.0;
       xmax = 2.0 ;
       ymin = 0.37;
       ymax = 0.37;
    }

    else System.out.println("frequency out of range of amp factor");

    m = (ymax - ymin)/(xmax-xmin);
    y = m*(log10freq - xmin) + ymin;

    betaterm = Math.pow(10, y);
    System.out.println(freq+ "   " + log10freq + "    " + y);
    amplification_factor = betaterm*rhoterm;
    return amplification_factor;
  }

  public double ptsource(double freq) {

    double point_source_term;
    double den1, den2, denom;

    den1 = freq/fc(beta, sigma, mw);
    den2 = den1*den1;
```

```
      denom = 1.0 + den2;
      point_source_term =  eqconst(beta,rho)*freq*freq*mw2mo(mw)/denom;
      return point_source_term;
   }

   public double pathop(double freq) {

      double term1;
      double path_operator;
      term1 = -Math.PI*freq*distance/(beta * qf(qo, nu, freq));
      path_operator = Math.exp(term1)/distance;
      return path_operator;
   }


   public double siteop(double freq) {

      double site_operator;
      site_operator = amp(freq) * pftf(kappa, freq);

      return site_operator;
   }

   public double asd1(double freq) {
      // calculate the acceleration spectral density from the
      // source, path, and site terms. Note: divide by
      // 980.0 to convert acceleration to gravity (g) units.
      double acc_spec_dens;
      acc_spec_dens = ptsource(freq)*pathop(freq)*siteop(freq)/980.0;
      return acc_spec_dens;

   }

}
```

Parameter Description

Earthquake magnitude - the magnitude of the earthquake (0.0-9.0)
Site - source distance - enter in cm. E.g., 2.0e6 = 20 km.

Shear wave velocity - enter the estimated shear wave velocity in the source region (in cm/s). E.g., 3.6e5 = 3.6 km /s.
Rock density - enter the rock density in the source region in gm /cm3.
Stress Drop Parameter - enter the stress drop in dyne/cm 2. E.g., 7.5e7 = 75 bar.
Kappa - empirical parameter for seismic wave attenuation (e.g., 0.02 - 0.06 sec).
Qo - regional crustal code (e.g., 100-660).
Nu - empirical parameter for frequncy dependent quality factor (e.g., 0.3-0.8).

About Code

Written in Java by C.B. Connor in May, 2000

This code uses the graph package for Java developed by Leigh Brookshaw

This code was developed at the Center for Nuclear Waste Regulatory Analyses (CNWRA) with funding from the U.S. Nuclear Regulatory Commission (NRC) under contract NRC-02-97-009. This code is an independent product of the CNWRA and does not necessarily reflect the views or regulatory position of the NRC.

Quality of Data, Analyses, and Code Development

Data: No CNWRA-generated original data are contained in this report. Sources of other data should be consulted for determining the level of quality of those data.

Codes: The seismic codes (AccSpec) including Accspectrum.java, AccspectrumGraph.java, PathoperatorGraph.java, SourcetermGraph.java, SitoperatorGraph.java, have been developed following the procedures described in the CNWRA Technical Operating Procedure (TOP)-18, Development and Control of Scientif and Engineering Software, which implements the quality assurance (QA) guidance contained in the CNWRA QA manual.

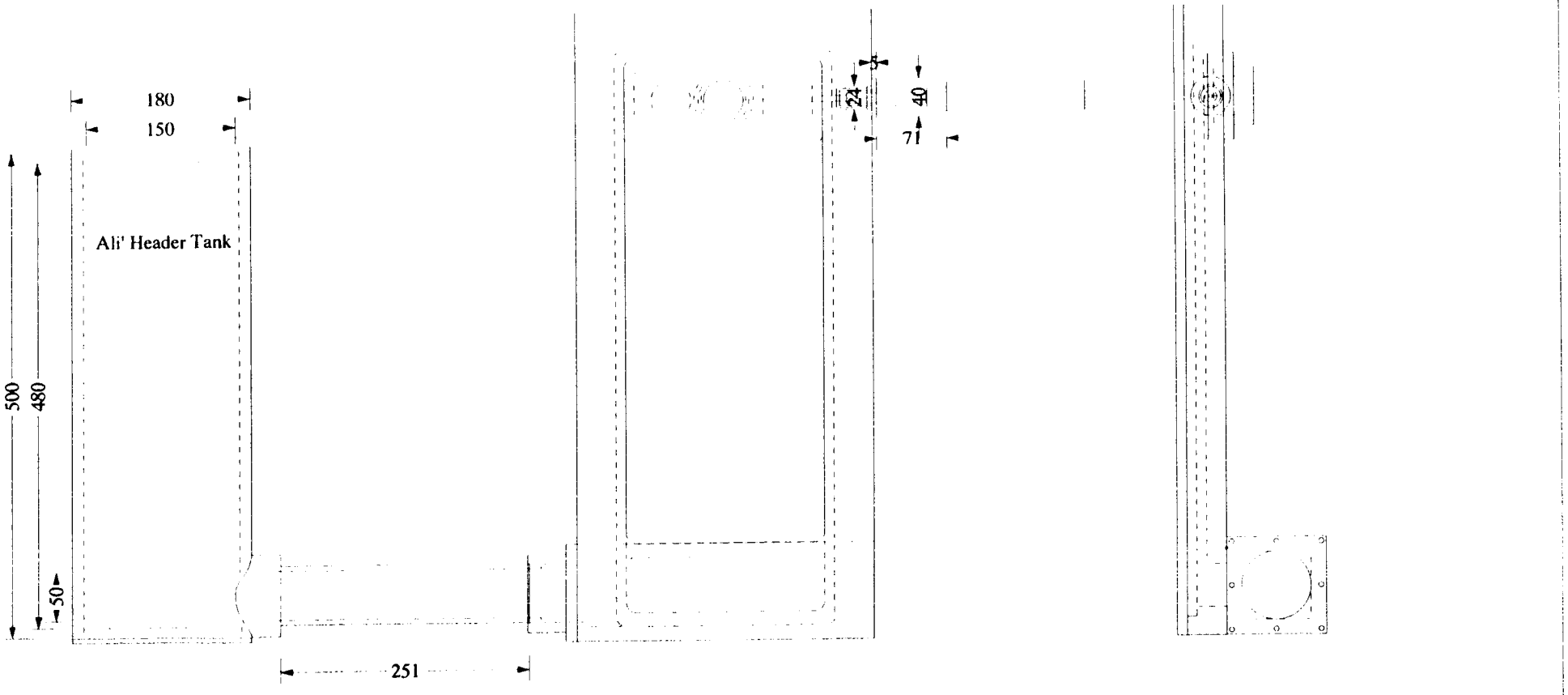
Disclaimer
"This computer code was prepared as an account of work performed at the CNWRA for the Nuclear Regulatory Commission (NRC), an independent agency of the U.S. government. Neither the developers of the codes nor any of their sponsors make any warranty, expressed or implied, or assume any legal liability or responsibility for the accuracy, completeness, or usefulness, or any apparatus, process, or product disclosed, or represent that its use would not infringe on privately-owned rights."

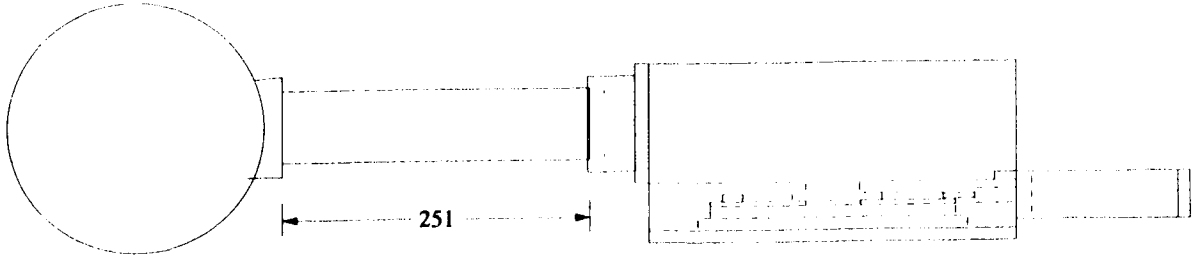
"In no event unless required by applicable law, will the sponsors or those who have written or modified this code, be liable for damages, including, any loss of profits, lost monies, or other special, incidental or consequnetial damages arising out of the use or inability to use this code (including but not limited to loss of data or data rendered inaccurate or losses sustained by third parties for failure of this program to operate with other programs), even if you have been advised of the potential for such damages, or any other claim by any other party."

Scientific Notebook 115E
20-1402-461
Inititals: CC

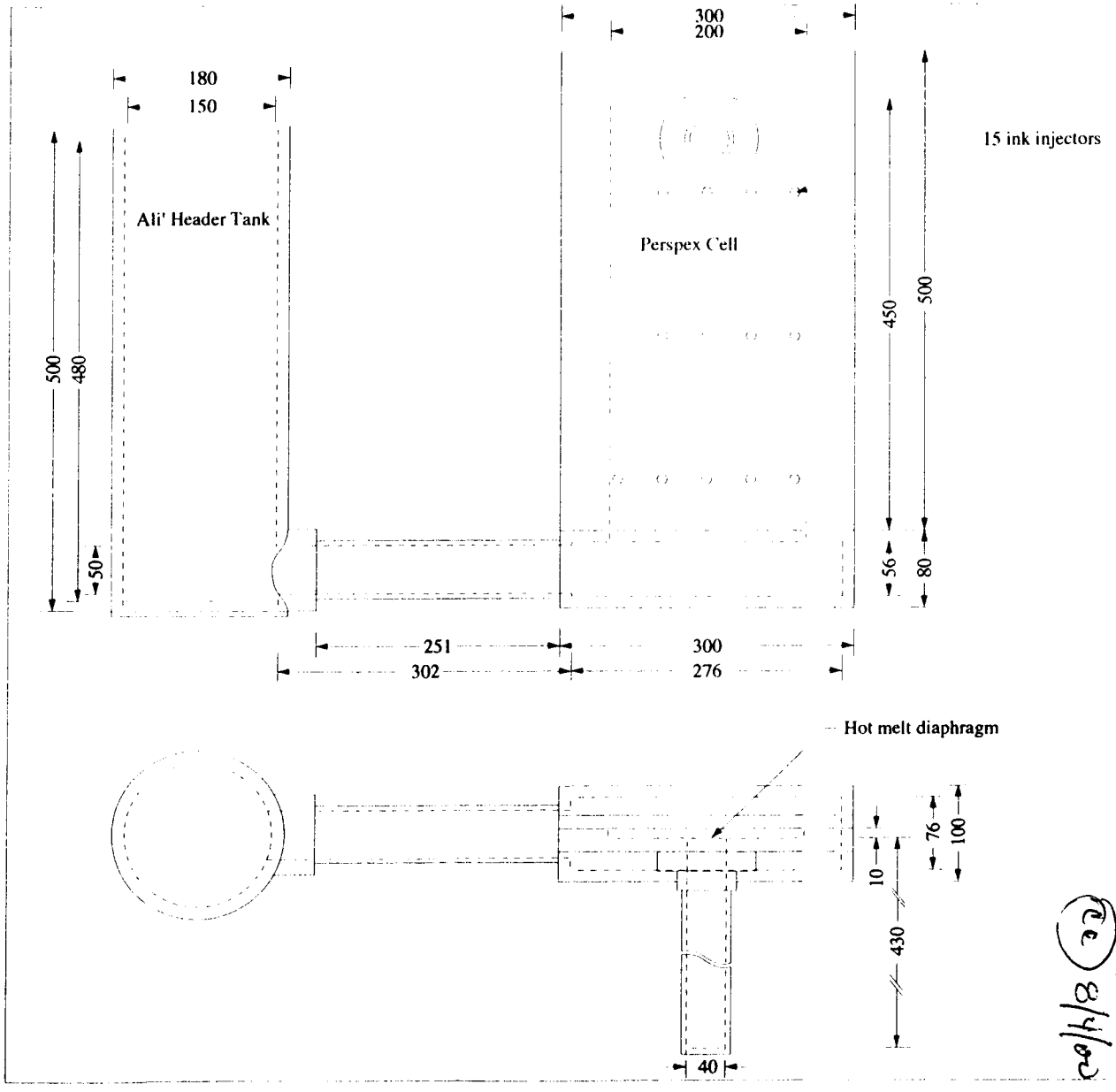
Chuck Connor
April 1, 2000

The following are drawings for the physical analog models for igneous activity (see previous notebook entries.

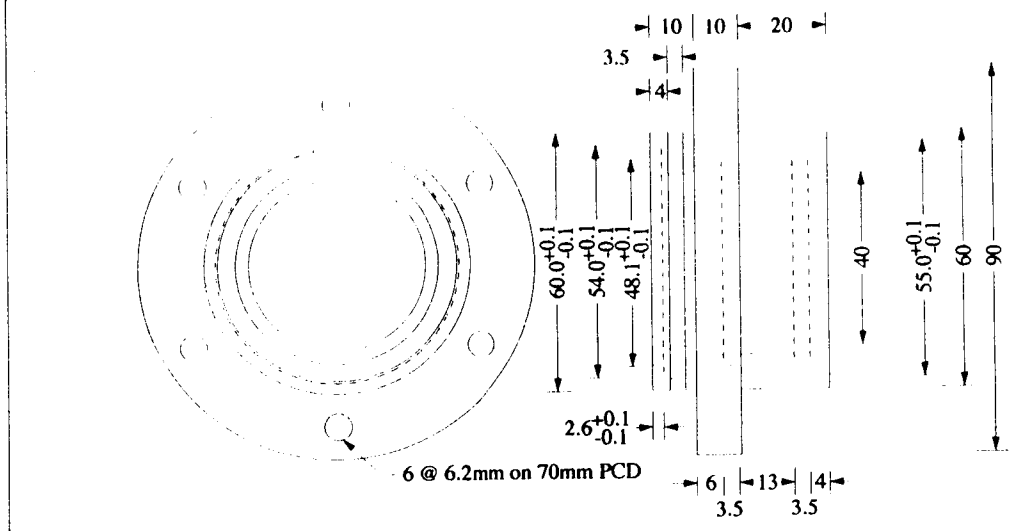


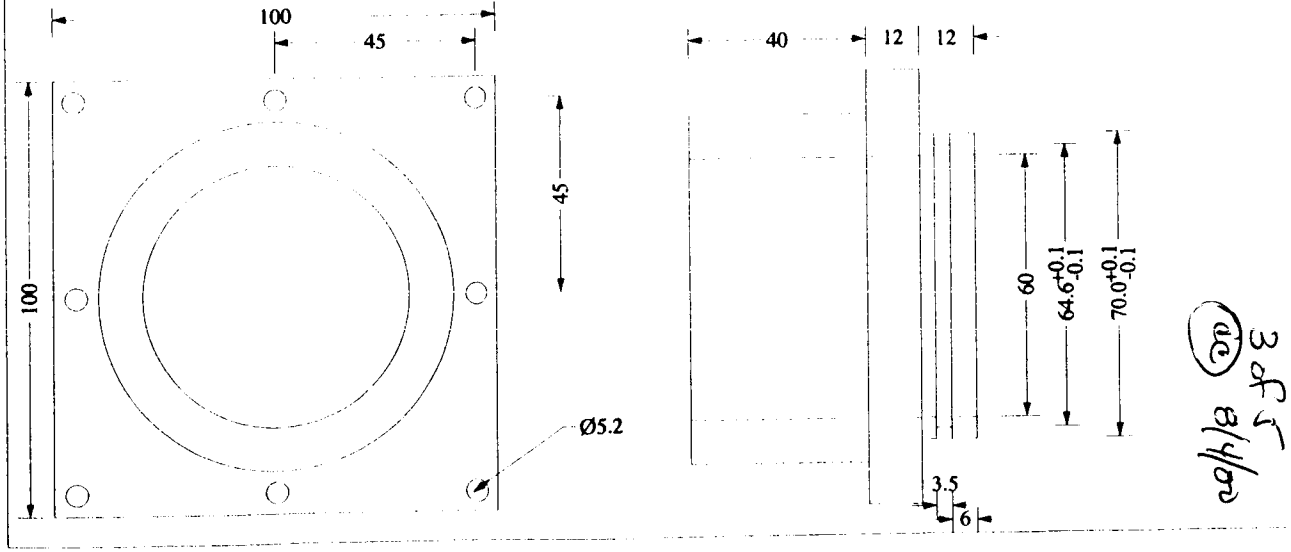


Department of Earth Sciences			
Title	Dyke Assembly		
Material	Ali'	Scale	1:1
Date	07/04/99	Drawn By.	M. Dury
Customer	Maths (Texas)		



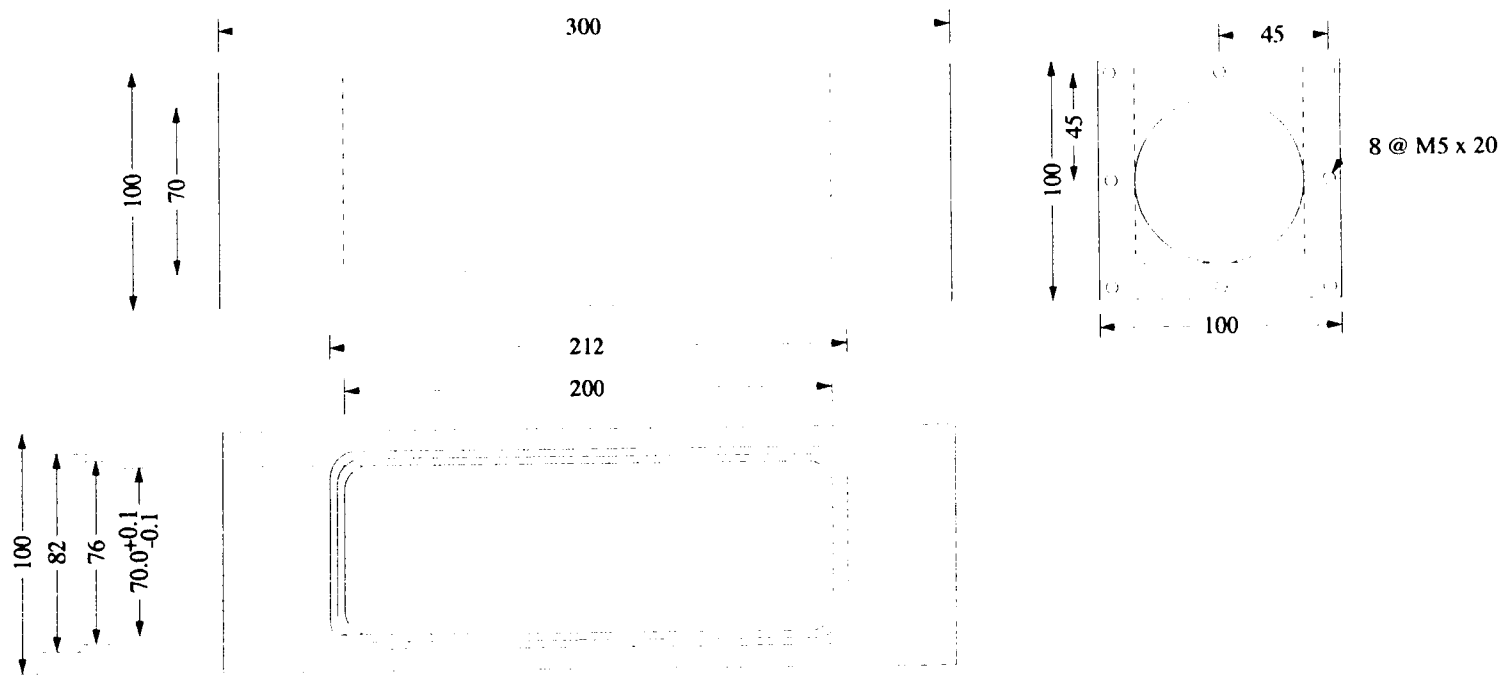
Department of Earth Sciences			
Title	Dyke Cell		
Material	Ali'/Perspex	Scale	1:4
Date	13/4/99	Drawn By.	M. Dury
Customer	Maths		





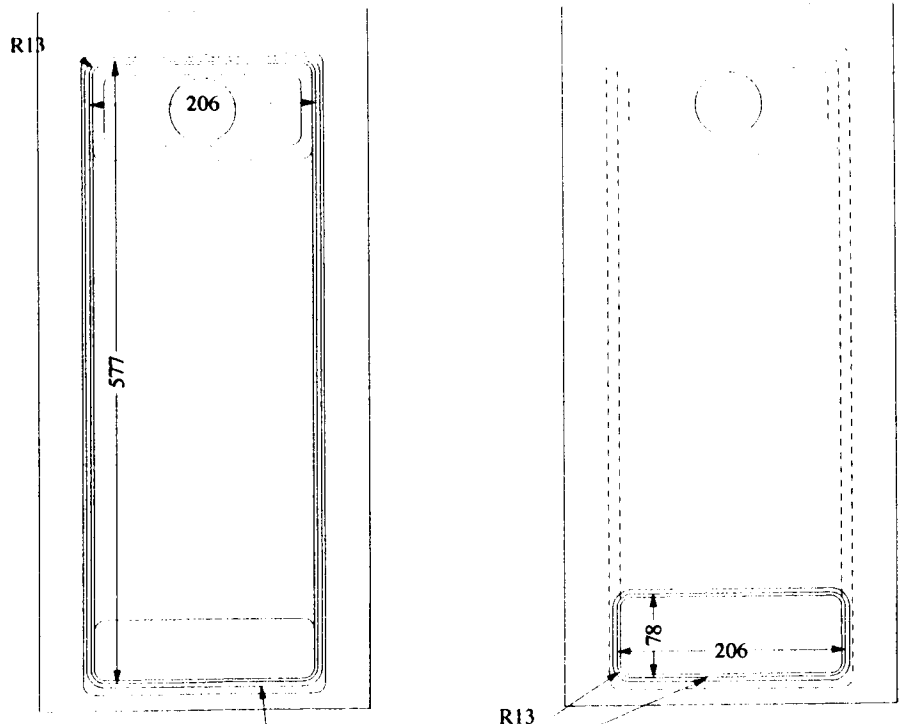
3 of 3 dc 8/4/98

Department of Earth Sciences			
Title	Dyke Details		
Material	Ali'	Scale	1:1
Date	07/04/99	Drawn By.	M. Dury
Customer	Maths (Texas)		



4 of 5
ee 8/04/00

	Department of Earth Sciences		
Title	Dyke Detail Base		
Material	Ali'	Scale	1:2
Date	07/04/99	Drawn By.	M. Dury
Customer	Maths (Texas)		



Department of Earth Sciences				
Title	Dyke Assembly			
Material	Ali'		Scale	1:1
Date	07/04/99	Drawn By.	M. Dury	
Customer	Maths (Texas)			

No original data have been removed. Entries in Scientific Notebook 115E for the period Feb - July, 2000 have been made by C. Comer C B Comer

9/18/00

- For EMSNs maintained by an individual, the last page of the semi-annual entry shall include the statement "Entries into Scientific Notebook No.## for the period date to date have been made by name/signature/date." It is not necessary to sign every page of the printed EMSN. For EMSN maintained by a project, the end of each individual's entry shall include the statement "Entries into Scientific Notebook No.## for pages ## to ## have been made by name/signature/date."

- The last page of the semi-annually printed entry of an EMSN will have a statement to the effect that "No original text entered into this Scientific Notebook has been removed." The statement will be signed and dated.

- If no entries were made during a six (6) month period, a memo from the individual author or project Principal Investigator indicating the absence of new entries will be included in the QA record.

- The printed copies of the EMSN will be bound and submitted in accordance with Section 4.2. The EM in charge of the project will review the printed copy prior to binding to ensure the completeness of the EMSN.

- Corrections to entries in the printed version of the EMSN can be made as specified in section 3.6.3, i.e., changes must be clearly identified by using the available features of the computer software.

I have reviewed scientific notebook 1151* and find it in compliance with QAP-001. There is sufficient information regarding procedure used for conducting the research and acquiring and analyzing the data so that another qualified scientist could repeat the activity or activities recorded in this scientific notebook

* FOR 1402.461

H. Lawrence McKague 7/25/00

H. Lawrence McKague
GLGP Element Manager

Information in Volume 15 of 17 is not U.S. Nuclear Regulatory Commission-Yucca Mountain-related information and is therefore not included in this file.

16 of 17

SCIENTIFIC NOTEBOOK 115E

CHUCK CONNOR

IGNEOUS ACTIVITY
20.01402.461

JULY–AUGUST 2000

SCIENTIFIC NOTEBOOK 115E

CHUCK CONNOR

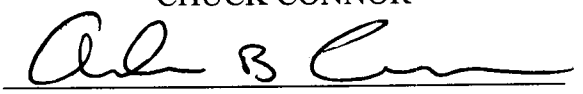
IGNEOUS ACTIVITY
20.01402.461

JULY–AUGUST 2000

SOUTHWEST RESEARCH INSTITUTE
CENTER FOR NUCLEAR WASTE REGULATORY ANALYSES
SAN ANTONIO, TEXAS

SCIENTIFIC NOTEBOOK 115E

CHUCK CONNOR



PROJECT

IGNEOUS ACTIVITY
20-1402-461

July-August 2000

PARTS IN THIS NOTEBOOK:

Section 1: Ash plume code development, testing , and summary
Section 2: preparation of paper on ash dispersion modeling in parallel
Section 3: status of Cerro Negro modeling
Section 4: final revision of the Onno Bokhove - Woods paper

Scientific Notebook 115E
20-1402-461
Inititals: CC

Chuck Connor
Aug 21, 2000

Modeling volcanic ash dispersion

Modeling ash dispersion is part of the IA activities in the Airborne transport ISI and is used in PA to calculate risk for the critical group. Our current approach to modeling ash dispersion in PA and IA is to use Suzuki's model to mimic the thermo-fluid-dynamics of ash dispersion in the atmosphere using the following:

$$X(x,y) = \int_{\rho_{min}}^{\rho_{max}} \int_{0}^{H} \frac{5P(z)f(\rho)Q}{8\pi C t^{5/2}} \exp\left[-\frac{5(x-ut)^2+y^2)}{8Ct^{5/2}}\right] dz d\rho$$

where:

X is the mass of ash and radionuclides accumulated at geographic location x, y, relative to the position of the volcanic vent,

$P(z)$ is a probability density function for diffusion of ash out of the eruption column, treated as a line source extending vertically from the vent to total column height, H,

$f(\rho)$ is a probability density function for grain size, ρ,

Q is the total mass of material erupted,

u is wind speed in the x-direction,

t is diffusion time of ash,

and C is eddy diffusivity.

In the Suzuki model, the erupting column is treated as a line source reaching some maximum height governed by the energy and mass of the eruption. A linear decrease in the upward velocity of particles is assumed, resulting in segregation of ash or ash and waste particles in the ascending column by settling velocity, which is a function of grainsize, shape, and density. Tephra and spent fuel (SF) particles are removed from the column based on their settling velocity, the decrease in upward velocity of the column as a function of height, and a probability density function which attempts to capture particle diffusion out of the column. Dispersion of the tephra and SF diffused out of the column is modeled for a uniform wind field and is governed by the diffusion-advection equation with vertical settling. Thus, results derived using this model depend heavily on assumptions about the shapes of the distributions $P(z)$ and $f(\rho)$. These distribution functions are empirically derived from analogous eruptions. Current IA activities

related to the Airborne Transport ISI include evaluation of these parameters and development of alternative approaches to using the Suzuki code. For example, we plan to investigate the effect of nonlinear velocity profiles. Similarly, the effects of more complex wind-fields, such as a stratified atmosphere, can be modeled.

The Suzuki model is easily recast to use parallel computing techniques. Since to double integral calculated for any point (x,y) or any parameter set, does not depend on previous calculations (other parameter sets or other points), the Suzuki model can be parallelized using a simple script. Doing so will enable much faster calculation and more experimentation to be done. This will benefit NRC work, and will also be used to market CNWRA in the hazards community.

The following lists a comparison of the Suzuki code written in java and tested earlier (see earlier section s in 115E) with the new parallel version of the code. The comparison indicates that the new parallel version of the code returns the same result, and is successful.

This comparison is followed by a listing of the parallel codes and associated scripts.

| Test Output |

The output of tephra_varwind_master.c/slave.c, the new parallel and stratified wind ash deposition implementations, were compared with Suzuki.java. The results of both programs using the same input data were identical.

The using the following input:

```
1E15 -5.0 5.0 -1.0 1.0 .8 0.5 0.5 10000 10.0

1 wind field: starting height   u   udir
                    0.0        1000  0
Running tephar_varwind_master/slave:
500000.000000 500000.000000 8.021364

5 wind field: starting height   u   udir
                      0.0        1000  0
                      1.0        1000  0
                      3.0        1000  0
```

```
                    5.0         1000  0
                    7.0         1000  0
Running tephar_varwind_master/slave:
500000.000000 500000.000000 8.021364

Running Suzuki.java (1 windfield: u=1000 udir=0):
mass of ash accumulated at 500000.0,500000.0 (gm / cm^2):8.02136391451127
```

Master code for parallel processing of tephra dispersion codes

```
/*
** Program name:   tephra_var_wind_master.c
**
** Abstract:  estimation of tephra accumulation at a point
**          using the method developed by Suzuki (1983) with
**           concurently running "slave" processes.
**
** Input:
**
**  <datafile>
**       which contains 12 input variables seperated by white space
**
**  <dimension file>
**       which contains
         "xorigin #points_on_xaxis yorigin #points_on_yaxis spacing_of_points"
**       in centemeters
**  <decompose>
**     GRID or INPUT
**
** Output:
**       mass deposited at a specfic location from the volcanic vent:
**        x (cm)   y (cm)   mass (gm/cm^2)
**
** Authors: Chuck Connor (cconnor@swri.edu)
```

```
**        Brandi Winfrey (b_winfrey@hotmail.com)
**        Nathan Franklin (nfrankli@trinity.edu)
**        Southwest Research Institute
**        6220 Culebra Rd
**        San Antonio TX 78238 USA
**        e-mail: cconnor@swri.edu, b_winfrey@hotmail.com, nfrankli@trinity.edu
**
**       This code was developed and tested with:
**              Compiler gcc 2.7
**              PVM version 3.4
** Code History:
**
**    revised:   18 Feb 2000
**            Brandi Winfrey
**            new create (from java implementation) by:
**
**    revised:    April -  July 2000
**             Nathan Franklin
**
**       Variable Wind Field
**       ------
**       The tephra code now allows for a stratified wind field. User can specify
**       a multiple wind fields. For the ash exiting the column at a given height,
**      the code determines the average wind field that the ash will experience.
**        -input functions have be changed to accomidate the new multiple wind
**        input data
**        -the wind input data is sent to each each of the slave processes.
**        -the slave process determines the average wind speed and direction at
**         each point along the column height integration
**
**       Parallel
**       ------
**      The previous serial code and been split into two serperate programs:
**      a master program and a slave program. The parallel decomposition is done
**      by the master program. The user can specify for the grid domain to be used
**      in the decomposition (GRID) or the input variables (INPUT). The problem is
**      distributed equally to all the slave proccesses for computation.
**
**
```

```
**
**
**
*/

#define NUMBER_VARIABLES      10    // number of input variables
                                    // (not including wind field variables)
#define NUMBER_VARIABLE_SETS  600   // number of input variables
#define MAX_NUMBER_SLAVES     20    // max # of possible processes
#define SIZE_GRIDDIMENSIONS   5
#define SIZE_RESULTS          5000
#define NUMBER_POSSIBLE_WINDFIELDS 15
#define WINDj 51
#define WINDi 1000

#include <stdio.h>
#include <stdlib.h>
#include "pvm3.h"
#include <time.h>
#define SLAVENAME "tephra_varwind_slave1.1"

/*  Input Functions */
int  VariablesIn(double x[], double wind[][WINDj], int *i_wind, char fn[]);
int  DimensionsIn(double x[], int MaxSize, char fn[]);


/*  Decomposition Functions */
void GridDecomposition(int nproc, int tids[], double grid_dimensions[],
                int n_variables, double variables[], int i_wind,
                int j_wind, double wind[][WINDj]);
void InputDecomposition(int nproc, int tids[], double grid_dimensions[],
                 int n_variables, double variables[], int j_wind,
                 double wind[][WINDj]);

void Quit(int tids[], int nproc);



int main(argc,argv)
```

```
int argc;
char *argv[];
{
   int mytid;              // my id
   int tids[MAX_NUMBER_SLAVES];          // slave task ids
   int nproc;              //number of proccesses
   int numt;
   int i,j,ii;
   int who;
   int node[MAX_NUMBER_SLAVES];
   int length_result;      //length of result (returned portion of grid)
   int msgtype;
   int nhost;              // number  of hosts
   int narch;
   int  n_grid_dimensions,n_variables, i_wind;
   int j_wind = WINDj;
   int time_length;        //length in seconds

   struct timeval start, stop;

   double variables[NUMBER_VARIABLES*NUMBER_VARIABLE_SETS];
   double wind[WINDi][WINDj];
   double grid_dimensions[SIZE_GRIDDIMENSIONS];
   double result[SIZE_RESULTS];
   struct pvmhostinfo *hostp;  //information of about each hosts:
                       // pvmd task id, name, arch, and relative
                              // speed

// ---------------------------------
// | Check  command line arguments |
// ---------------------------------


  // standard usage io
  if ( argc != 4  ) {
      fprintf ( stderr, "Usage: %s <variable file> <grid dimensions> <decomposition: GRID or
INPUT>\n", argv[0] );
    exit (0);
  }
```

```
  if(strcmp(argv[3],"GRID")!=0 && strcmp(argv[3],"INPUT")){
       fprintf(stderr,"Usage: %s <variable file> <grid dimensions> <decomposition: GRID or
INPUT>\n", argv[0] );
    fprintf ( stderr, "<decomposition: GRID or INPUT>:%s\n", argv[3] );
    exit(1);
  }
  gettimeofday(&start, (struct timezone*)0); //start clock

// ----------------------------
// | Get variables from file |
// ----------------------------

  n_variables=VariablesIn(variables, wind, &i_wind, argv[1]);
  n_grid_dimensions=DimensionsIn(grid_dimensions,SIZE_GRIDDIMENSIONS,argv[2]);
  if(n_grid_dimensions!=5)
  {
       printf("\nGrid dimension File Error: check %s!\n",argv[2]);
       exit(1);
  }
//debug:
//PRINT TEST
/*   printf("# of wind sets: %d\n\n",i_wind);
  for(i=0;i<i_wind;i++)
  {
    for(ii=0;ii<(NUMBER_VARIABLES);ii++)
          printf("%f ",variables[i*NUMBER_VARIABLES + ii]);
     printf("\n");
     printf("# of wind: %.0f\n",wind[i][0]);
        for(j=1;j<=(wind[i][0]*3);j++)
          printf("%f ",wind[i][j]);
     printf("\n");
  }
 printf("wind array:");
    for(i=0;i<=i_wind;i++){
     printf("%f    ",wind[i][0]);
     for(j=1;j<=(wind[i][0]*3);j++)
       printf("%f ",wind[i][j]);
     printf("\n");
     }
```

```
*/

// ----------------
// | Start Slaves |
// ----------------

   //enroll in pvm
   mytid = pvm_mytid();

   // Set number of slaves to start
   pvm_config( &nhost, &narch, &hostp );
   nproc = nhost;


   //printf("Spawning %d slaves ... " , nproc);


   numt=pvm_spawn(SLAVENAME, (char**)0, 0, "", nproc, tids);

  if( numt < nproc ){
     printf("\n Trouble spawning slaves. Aborting. Error codes are:\n");
     for( i=numt ; i<nproc ; i++ ) {
       printf("TID %d %d\n",i,tids[i]);
     }
     Quit(tids, nproc);
   }
   //printf("SUCCESSFUL\n");


// -------------------------------------
// | Broadcast initial data to slaves |
// -------------------------------------

        pvm_initsend(PvmDataDefault);
        pvm_pkint(&nproc, 1, 1);
        pvm_pkint(tids, nproc, 1);
      pvm_mcast(tids, nproc, 0);
```

```
// ----------------------------------------
// | Decompose Problem and Send to Slaves |
// ----------------------------------------
  //INPUT DECOMPOSITION
  if(strcmp(argv[3],"INPUT")==0){
    InputDecomposition(nproc, tids,grid_dimensions, n_variables, variables, j_wind, wind);
  }
  //GRID DECOMPOSITION
  else{
   GridDecomposition(nproc, tids,grid_dimensions, n_variables, variables, i_wind, j_wind, wind);
  }
// -------------------------------------
// | Wait for results from the slaves |
// -------------------------------------

   msgtype = 50;  //going to listen to mesages with this type
   //printf("Waiting for results\n");
   //debug:        printf("%f ",wind[i][j]);
   //printf("debug: \n");
   //printf("%f %f %f\n", wind[0][0], wind[0][1], wind[0][2]);
   //printf("%f %f %f\n", wind[1][0], wind[1][1], wind[1][2]);
   //printf("%f %f %f\n\n", wind[2][0], wind[2][1], wind[2][2]);
   for( i=0 ; i<nproc ; i++ ){
     pvm_recv( -1, i );
     pvm_upkint( &who, 1, 1 );
     pvm_upkstr(node);
     pvm_upkint( &length_result, 1, 1);
     pvm_upkdouble( &result[0], length_result, 1 );
     //printf("Receiving from %s procces # %d: \n",node,who);

     for(j=0; j<length_result; j=j+3)
       printf( "%f %f %f\n", result[j],result[j+1],result[j+2]);

   }

// ----------------------------
// | End of program... quit  |
// ----------------------------
   gettimeofday(&stop, (struct timezone*)0); //stop clock
```

```
   time_length= stop.tv_sec -start.tv_sec;
   //printf("%s lasted %d secs\n",argv[0],time_length);

   pvm_exit();
   return 1;
}


/* ----------------- VariablesIn ---------------------- */
/* function reads in a file and fills two arrays: wind array and variable array.
   It returns the number of doubles in the variable array.
*/
int VariablesIn(double x[], double wind[][WINDj], int *i_wind, char fn[])
{
 //INPUT:  a file with 10 variables plus wind data
 //OUTPUT: x[] which contains the sets of 10 variables,
 //                wind[][] which contains the wind data
 //          (wind[0][0] is the number of wind fields),

 //INPUT DATA
 // 11th variable is the number of wind layers,
 // each wind layer is the starting height in km followed by
 // the wind speed and direction.
 //                                    v
 //2.04 -5.0 5.0 -1.0 1.0 1.14 0.5 0.5 3182.9 5.0   2    0 1273.70 -0.02 2.5 1300 .05
 // 0   1   2   3  4  5   6  7   8   9   10   11 12      13 14  15  16
  FILE *fp;
  int N=0, count=0, r,i=0,j;
  double temp;

  fp=fopen(fn, "r");
  if(fp==NULL)
  {
    printf("\nFile Error\n----------\n %s does not exist.\n",fn);
    pvm_exit();
    exit(1);
  }
  r=fscanf(fp, "%lf", &temp);
  while(r!=EOF)
```

```
  {
    x[N]=temp;
    N++;
    count++;
    r=fscanf(fp, "%lf", &temp);

   if(count==10)  //getting the wind field
    {
        wind[i][0]=temp;
        j=0;
     r=fscanf(fp, "%lf", &temp);
        while( r!=EOF && j<(wind[0][0] * 3) )
        {
         wind[i][j+1]=temp;
      r=fscanf(fp, "%lf", &temp);
         j++;
        }
     i++;
     count=0;
    }
  }
  *i_wind=i;
  fclose(fp);
  return N;
}

/* ---------------- DimensionsIn -------------------- */
/* function fills an array with doubles from a file.
   it returns the number of doubles.
*/
int DimensionsIn(double x[], int MaxSize,  char fn[])
{
  FILE *fp;
  int N=0, count=0, r;
  double temp;

  fp=fopen(fn, "r");
  if(fp==NULL)
  {
```

```
    printf("\nFile Error\n----------\n %s does not exist.\n",fn);
    pvm_exit();
    exit(1);
  }
  r=fscanf(fp, "%lf", &temp);
  while(r!=EOF)
  {
    if(N >= MaxSize){
      printf("Array not large enough!\n");
      pvm_exit();
      exit(1);
    }
    x[N]=temp;
    N++;
    count++;
    r=fscanf(fp, "%lf", &temp);
  }
  fclose(fp);
  return N;
}

/* ----------------- GridDecomposition ---------------------- */
/* function decomposes the problem along the x-axis.
   it then distributes the portions of the problem to the slaves.
*/
void GridDecomposition(int nproc, int tids[], double grid_dimensions[],
                       int n_variables, double variables[], int i_wind,
                       int j_wind, double wind[][WINDj])
{
   int total_number_of_xpoints, number_of_xpoints, my_xpoints;
   int remainder;
   int i;
   double where;
    total_number_of_xpoints=grid_dimensions[1];
    number_of_xpoints=total_number_of_xpoints/nproc;
    remainder=n_variables % NUMBER_VARIABLES;
    where=grid_dimensions[0];
    if(remainder!=0)
    {
```

```
      fprintf ( stderr, "ERROR in the variable file\n");
      Quit(tids, nproc);
    }
    remainder=total_number_of_xpoints % nproc;
    for(i=0; i<nproc; i++)
    {
     if(remainder!=0 && i <remainder)
          my_xpoints=number_of_xpoints+1;
     else
          my_xpoints=number_of_xpoints;
     pvm_initsend(PvmDataDefault);

     pvm_pkint(&n_variables, 1, 1);
     pvm_pkdouble( &variables[0], n_variables, 1 );

     pvm_pkint(&i_wind, 1, 1);
     pvm_pkint(&j_wind, 1, 1);
     pvm_pkdouble( &wind[0][0], i_wind*j_wind, 1 );

     grid_dimensions[0]=where;
     grid_dimensions[1]=my_xpoints;

     pvm_pkdouble(&grid_dimensions[0], 5, 1 );

     pvm_send(tids[i],4);
     where=where+my_xpoints*grid_dimensions[4];
    }
}


/* ---------------- InputDecomposition ---------------------- */
/* function decomposes the problem by the sets of input variables.
   it then distributes the portions of the problem to the slaves.
*/
void InputDecomposition(int nproc, int tids[], double grid_dimensions[],
                int n_variables, double variables[], int j_wind,
                double wind[][WINDj]){

    int i, total_number_of_runs, number_of_runs,
```

```
        where, my_runs, remainder, size;

    total_number_of_runs=n_variables / NUMBER_VARIABLES;
    remainder=n_variables % NUMBER_VARIABLES;
    if(remainder!=0)
    {
      fprintf ( stderr, "ERROR in the variable file\n");
      Quit(tids, nproc);
    }
    number_of_runs=total_number_of_runs/nproc;
    remainder=total_number_of_runs % nproc;
    where=0;
    for(i=0; i<nproc; i++)
    {
     if(remainder!=0 && i < remainder)
           my_runs=number_of_runs+1;
     else
           my_runs=number_of_runs;

     pvm_initsend(PvmDataDefault);

     size=my_runs*NUMBER_VARIABLES;
     pvm_pkint(&size, 1, 1);
                    pvm_pkdouble(  &variables[where  *  NUMBER_VARIABLES],
my_runs*NUMBER_VARIABLES, 1 );

     pvm_pkint(&my_runs, 1, 1);
     pvm_pkint(&j_wind, 1, 1);
     pvm_pkdouble( &wind[where][0], my_runs*j_wind, 1 );

     pvm_pkdouble( &grid_dimensions[0], 5, 1 );

     pvm_send(tids[i],4);
     where=where+my_runs;
    }

}

void Quit(int tids[], int nproc)
```

```
{
    int i;

    for(i=0 ; i<nproc ; i++ ){
      pvm_kill( tids[i] );
    }
    pvm_exit();
    exit(1);
}



/*
** Program name:  tephra_varwind_slave1.1.c
**
** Purpose:   To calculate ash deposition. This program is started
**           by a master program.  It recieves the variable and grid
**           data from the master and then computes the calculations.
**         It returns the answers to the master and quits.
**
*/



#include <stdio.h>
#include <stdlib.h>
#include "pvm3.h"
#include <math.h>

// ----------------------------------------
// |   Global variables and constants    |
// ----------------------------------------

#define PI              3.141592654
#define NUMBER_OF_VARIABLES 10  //number of variables = 12 variables
#define SIZE_GRIDDIMENSIONS  5
#define XSTEPS 300  //# steps for integration over the eruption column
#define YSTEPS 100  //# steps for integration over the particle size range
```

```
//following are assumed const.
double c      = 400.0;       // eddy diffusivity in the atmosphere
double phiair = 0.001293;    // density of air
double nua    = 0.00018;     // dynamic viscosity of air
double gravity = 980.0;      // gravity

// ---------------------
// |  Functions     |
// ---------------------

//grid():         Makes the grid
int   grid(double *data, double *input_variables, int n_variables,
        double **wind, int i_wind, double xorigin,
        double xsteps, double yorigin, double ysteps,
        double spacing, int n_slaves, int n_me);

//tephra_accumulation():      Used in grid() to find tephra accumulation
double tephra_accumulation(double xpos, double ypos, int where, double **wind,
                  double total_ash_mass, double ymin, double ymax,
                  double dmean, double dsigma, double phiash,
                  double dfshape, double dbeta, double w0, double h);

//functions used by tephra_accumulation to find accumulation:
double phi2cm( double xx );
double w( double zspace, double w0 , double h );
double p(double zspace, double xrho, double dbeta,
      double w0, double h, double dfshape, double phiash);
double ts( double zspace );
double t( double zspace, double xrho, double dfshape, double phiash );
double v0( double xrho, double dfshape, double phiash );
double frho( double xrho, double dmean, double dsigma );
double f(double xspace, double yspace, double zspace, double xrho,
       double u, double total_ash_mass,  double dmean,  double dsigma,
       double phiash,  double dfshape,  double dbeta,  double w0,  double h);
void  average_wind(double wind_av[][2], int where, double **wind, int xsteps, double xdelt);


//functions for allocating and freeing arrays of doubles
double *AllocateOneDimArray(int length);
```

```
void  FreeOneDimArray(double *a, int length);
double **AllocateTwoDimArray(int ilength, int jlength);
void  FreeTwoDimArray(double **a, int ilength, int jlength);


int main( int argc,  char *argv[])
{

// ------------------------------
// |   Setting up variables    |
// ------------------------------


// PVM variables:
//---------------
   int mytid;      /* my task id */
   int tids[32];   /* task ids   */
   int me, nproc, master, msgtype;
   char buf[100];

// Tephra program variables:
//--------------------------
   // Grid dimensions
   int length_data_array;
   int n_variables;
   int i_wind, j_wind;
   int i;

   double x_origin;
   double x_steps;
   double y_origin;
   double y_steps;
   double spacing;

   double *arraygrid;  //array of output grid
   double *variables;  //array of input variables
   double **wind;      //wind array
   double grid_dimensions[SIZE_GRIDDIMENSIONS];    //array of grid dimensions
```

```
// --------------------------------
// | Receive data from the master |
// --------------------------------

   master = pvm_parent();
   mytid = pvm_mytid();
   msgtype = 0;            // setting msgtag to 0. Master should
                           // be also broadcasting with 0.
   pvm_recv( -1, msgtype );

   pvm_upkint(&nproc, 1, 1);
   pvm_upkint(tids, nproc, 1);


   /* Determine which slave I am (0 -- nproc-1) */
   for( i=0; i<nproc ; i++ ){
     if( mytid == tids[i] )
          me = i;
   }
   pvm_recv(master, 4);
   pvm_upkint( &n_variables, 1, 1);
   variables  = AllocateOneDimArray(n_variables);

   pvm_upkdouble( &variables[0], n_variables, 1 );
   pvm_upkint( &i_wind, 1, 1);
   pvm_upkint( &j_wind, 1, 1);
   wind      = AllocateTwoDimArray(i_wind,j_wind-1);
   pvm_upkdouble( &wind[0][0], i_wind*j_wind, 1 );
   pvm_upkdouble( &grid_dimensions[0], 5, 1 );


// --------------------------------
// | Do calculations with data    |
// --------------------------------


   spacing=grid_dimensions[4];
   x_steps=grid_dimensions[1];
```

```
   x_origin=grid_dimensions[0];
   y_origin=grid_dimensions[2];
   y_steps=grid_dimensions[3];

   //allocate output array:  length= # grid points * # of runs * 3 (x,y,acc)
   length_data_array = 3 * x_steps*y_steps*(n_variables/NUMBER_OF_VARIABLES);
   arraygrid  = AllocateOneDimArray(length_data_array);

   //get grid -> arraygrid
  grid(arraygrid, variables, n_variables, wind, i_wind, x_origin, x_steps,y_origin, y_steps, spacing,
nproc, me);

   //get my hostname
   gethostname (buf, 64);


// ---------------------------------
// |   Send results to master     |
// ---------------------------------
   pvm_initsend( PvmDataDefault );
   pvm_pkint( &me, 1, 1 );
   pvm_pkstr(buf);
   pvm_pkint( &length_data_array, 1, 1 );
   //debug
   /*arraygrid[0]=wind[0][0];
   arraygrid[1]=wind[0][1];
   arraygrid[2]=wind[0][2];
   arraygrid[3]=wind[1][0];
   arraygrid[4]=wind[1][1];
   arraygrid[5]=wind[1][2];
   arraygrid[6]=wind[2][0];
   arraygrid[7]=wind[2][1];
   arraygrid[8]=wind[2][2];
   */
   pvm_pkdouble( &arraygrid[0], length_data_array, 1 );
   pvm_pkdouble( &arraygrid[0], length_data_array, 1 );
   msgtype = me;
   pvm_send( master, msgtype );
```

```
   FreeOneDimArray(grid_dimensions,SIZE_GRIDDIMENSIONS);
   FreeTwoDimArray(wind,i_wind,j_wind);
// --------------------------------
// |   Program finished...exit   |
// --------------------------------
   printf("end of program\n");
   pvm_exit();

   return 1;
}
```

```
int grid(double *data, double *input_variables, int n_variables, double **wind, int i_wind, double
xorigin, double xsteps, double yorigin, double ysteps, double spacing, int n_slaves, int n_me)
{
  // following are the 10 input variables
  double total_ash_mass,ymin, ymax, dmean, dsigma, phiash;
  double dfshape, dbeta, w0, h;

  int i,j;
  int x_step_count,y_step_count,number_of_runs;
  int grid_xsteps;
  int grid_ysteps;

  double xpos, ypos;

  grid_xsteps=xsteps +0;
  grid_ysteps=ysteps;
  xpos=xorigin;
  ypos=yorigin;

// ------------------------------------------------
// | Parallel:  Take my portion of variable set   |
```

```
// ------------------------------------------------
  number_of_runs = n_variables/NUMBER_OF_VARIABLES; // # of variables
                                  // divided by # of variables
                                  // in a run


  i=0;
  x_step_count=0;

        while(x_step_count<grid_xsteps)
        {

              y_step_count=0;

            while(y_step_count<grid_ysteps)
              {
            //move to next point in grid
            xpos=xorigin + (x_step_count*spacing);
                  ypos=yorigin + (y_step_count*spacing);

            //at this point find tephra accumulation
            //     for each set of input variables
            for(j=0;j<number_of_runs;j++)
                  {
               total_ash_mass=input_variables[0+((j)*NUMBER_OF_VARIABLES)];
               ymin   =input_variables[1+((j)*NUMBER_OF_VARIABLES)];
               ymax   =input_variables[2+((j)*NUMBER_OF_VARIABLES)];
               dmean  =input_variables[3+((j)*NUMBER_OF_VARIABLES)];
               dsigma =input_variables[4+((j)*NUMBER_OF_VARIABLES)];
               phiash =input_variables[5+((j)*NUMBER_OF_VARIABLES)];
               dfshape=input_variables[6+((j)*NUMBER_OF_VARIABLES)];
               dbeta  =input_variables[7+((j)*NUMBER_OF_VARIABLES)];
               w0     =input_variables[8+((j)*NUMBER_OF_VARIABLES)];
               h      =input_variables[9+((j)*NUMBER_OF_VARIABLES)];


                        data[i]=xpos;
                        i++;
                        data[i]=ypos;
```

```
                        i++;
                        data[i]=tephra_accumulation(xpos, ypos, j, wind, total_ash_mass, ymin,
ymax, dmean, dsigma, phiash, dfshape, dbeta, w0, h);
            i++;

          }//end of for

              y_step_count++;
            }//end of 'y' while

            x_step_count++;
        }//end of 'x' while
  return i;
}


/* ---------- tephra_accumulation ---------- */
/* calculates the epected mass (gm/cm^2) of ash to accumulate in a
given location xspace, yspace using function f and integrating over the column hieght and particle
size range.
*/
double tephra_accumulation(double xpos, double ypos, int where, double **wind, double
total_ash_mass, double ymin, double ymax, double dmean, double dsigma, double phiash, double
dfshape, double dbeta, double w0, double h)
{
    double xsectionwidth;
    double ysectionwidth;
      double xmin, xmax;
      double xstepwidth;
    double ystepwidth;
      double new_xpos0, new_ypos0;
      double new_xpos1, new_ypos1;
      double x0, y0, x1, xi, y1, yi,ans;
    double total;
    double u0,udir0;
    double u1,udir1;
    double wind_av[XSTEPS][2];
    int xsteps, ysteps, yyi, xxi;
   // define the limits of integration
```

Chuck Connor
Aug 21, 2000

```
//--------------------------------
// xmin is the elevation of the vent
// xmax is the total height of the eruption column
// ymin and ymax are minimum and maximum
// grainsizes, respectively

 // do the integration over the eruption
    // column height in 300 steps
    xsteps = XSTEPS;

    // do the intergration over the particle
    // size range in 100 steps
    ysteps = YSTEPS;

    xmin= 0.0;
    xmax =h;
 xsectionwidth = xmax - xmin;
 ysectionwidth = ymax - ymin;
 xstepwidth = xsectionwidth / xsteps;
 ystepwidth = ysectionwidth / ysteps;
    average_wind(wind_av, where, wind, xsteps, xstepwidth);

    // initialize the summation for the
    // contribution of each d phi and
    // each dz to the total accumulation
    // of tephra on the ground at the
    // point xspace, yspace
    total  = 0.0;


    //do the integration
 //-------------------
    for(yyi=1; yyi < ysteps; yyi++) {

            // define the small slice d phi

       yi = (double)yyi;
       y0 = ymin + yi * ystepwidth;
       y1 = ymin + (yi+(double)1.0) * ystepwidth;
```

```
            for(xxi=1; xxi < xsteps; xxi ++) {

                // define the small slice dz
            xi = (double)xxi;
            x0 = xmin + (xi) * xstepwidth;
            x1 = xmin + (xi+(double)1.0) * xstepwidth;

                u0=wind_av[xxi][0];
                u1=wind_av[xxi+1][0];
                udir0=wind_av[xxi][1];
                udir1=wind_av[xxi+1][1];

                //define the coordinate system with
                // respect to wind direction
                //---------------------------------
                // here we are just assuming that
                // the wind is varying N or south of
                // a western direction, need to be genrealized

              new_xpos0 = xpos * cos(udir0) + ypos * sin(udir0);
            new_ypos0 = -xpos*sin(udir0) + ypos*cos(udir0);
                new_xpos1= xpos * cos(udir1) + ypos * sin(udir1);
            new_ypos1= -xpos*sin(udir1) + ypos*cos(udir1);



            // calculate the value of the ash accumulation  at
            // the corners  of the slice, and sum
           ans =   f(new_xpos0, new_ypos0, x0, y0, u0, total_ash_mass, dmean, dsigma, phiash,
dfshape, dbeta, w0, h)
                  + f(new_xpos1, new_ypos1, x1, y0, u1, total_ash_mass, dmean, dsigma, phiash,
dfshape, dbeta, w0, h)
                  + f(new_xpos0, new_ypos0, x0, y1, u0, total_ash_mass, dmean, dsigma, phiash,
dfshape, dbeta, w0, h)
                  + f(new_xpos1, new_ypos1, x1, y1, u1, total_ash_mass, dmean, dsigma, phiash,
dfshape, dbeta, w0, h);
              total += ans;
            }
        }
```

```
    // complete the integration by multiplying by dz and d phi
    // and dividing by 4, since 4 corners were used
    total *= xstepwidth * ystepwidth / 4.0;

    //all finished: return the answer
    return total;
}


/* ---------- average_wind ---------- */
/* function average_wind finds the average wind (wind_av[h][0]) and wind direction
  (wind_av[h][1]) for ash leaving the column at at specific heights along the column.
*/

void average_wind(double wind_av[][2], int where, double **wind, int xsteps, double xdelt)
{
        int i,j;
        double sum_x=0;
        double sum_y=0;
        j=1;
    //windfield[0]= number of windfields.
    //windfield[1]= starting height
    //windfield[2]= u
    //windfield[3]= udir
    //windfield[4]= starting height ...
    //windfield[5]= u
    //windfield[6]= udir
    //calculate average wind at heights for the itegration over the column.
    for(i=1;i < xsteps;i++) {

     // if there are more windfields and if the current height
     // is greater than the starting height of the next windfied:
     // jump to next wind field
     //   * this step assumes that the height of each individual wind
     //     field is greater than the
     if( (wind[where][0] * 3) >= (j+3)  &&  (i*xdelt) >= wind[where][j+3])
      j=j+3;

      //calculate the sum of the x components and the sum
```

```
        // of the y components of the wind vector
           sum_x= (wind[where][j+1] * cos(wind[where][j+2])) + sum_x;
           sum_y= (wind[where][j+1] * sin(wind[where][j+2])) + sum_y;

        //Use these components to calculate the average speed
        // and average direction of the wind exprienced for ash
        // departing at this column height.

        wind_av[i][0]=( sqrt(sum_x*sum_x + sum_y*sum_y) ) / i;
        wind_av[i][1]= atan2(sum_y,sum_x);
      }
}


/* ----------------- phi2cm ---------------------- */
/* function phi2cm converts the ash diameter from
   units of phi to cm
*/

double phi2cm(double xx) {
  double cms;

  cms = 0.1 * 1.0 / pow(2, xx);
  return cms;
}

/* ----------------- w ---------------------- */
/* this function estimates the velocity
  as a function of height in the
  eruption column, given a
  linear decrease in velocity with
  height and an initial velocity
  of w0
*/

/* returns the upward particle velocity  */

double w(double zspace, double w0, double h) {
  double particle_velocity;
```

```
  particle_velocity = w0 * (1.0 - zspace/h);

  // following only ocurrs as round-off error
 if (zspace > h) particle_velocity = 0.0;
  return particle_velocity;
}


/* ---------------- p ------------------- */
/* this function calculates the probability
   density for diffusion of ash out of the
   eruption column. Some variation from Suzuki
   implemented to conserve mass

   depends on beta, w0, w(z), v0(xrho)
   where w0 is the eruption velocity in cm/s
   w(z) in the upward particle velocity at height z (cm/s)
   v0(xrho) is the particle settling velocity (cm/s)
   h is the column height (km)

   integrating fz from 0 to h should give unity

*/

double p(double zspace, double xrho, double dbeta, double w0, double h, double dfshape, double
phiash) {
  double y, y0;
  double prob_dens_func;
  double voo;
  double demon1, demon2;
  voo = v0(xrho, dfshape, phiash);
  y  = dbeta * w(zspace, w0, h) / voo;
  y0 = dbeta * w0 / voo;
  demon1 = dbeta * w0 * y * exp(-y);
  demon2 = voo*h * (1.0-(1.0+y0) * exp(-y0));
  prob_dens_func = demon1/demon2;

  return prob_dens_func;
}
```

```
/* ----------------- ts -------------------- */
/* function ts determines the particle diffusion time const
  in the atmosphere
  where zspace = height of the particle, converted here from
              km to cm, this should be wrt sea-level,
              zspace is given here in cm because
              of the units of eddy diffusivity
       c = eddy diffusivity in the atmosphere
          given as in cgs, e.g., 400 cm/s^(5/2)

  returns the particle diffusion time
*/

double ts (double zspace) {
  double particle_diffusion_time;
  double demon1, demon2;

  zspace *= 1.0e+5;
  demon1  = 5.0 * zspace * zspace;
  demon2  = 288.0 * c;
  particle_diffusion_time = pow(demon1/demon2, 2.0/5.0);

  return particle_diffusion_time;
}


/* ----------------- t -------------------- */
/* function t calculates the particle fall time
  in the atmposphere
  where: zspace = height in the atmosphere, here in km
              wrt to sealevel, because the const.
              -0.0625 is in units of 1/km
*/
double t (double zspace, double xrho, double dfshape, double phiash) {
  double particle_fall_time;
  double demon1, demon2;
```

```
  demon1 = (1.0 - exp(-0.0625*zspace)) / v0(xrho, dfshape, phiash);
  demon2 = pow(demon1, 0.926);
  particle_fall_time = 0.752e6 * demon2;
  return particle_fall_time;
}


/* ---------------- v0 ------------------- */
/* return a value for the particle terminal velocity
  at sea level, given the particle diameter, passed as x
  note that the shape factor does not depend on x
  note that the ash dens does not depend on x
*/

double v0 (double xrho, double dfshape, double phiash) {

  double velocity0;
  double demon1, demon2, demon3;

  demon1 = 9.0 * nua * pow(dfshape, -0.32);
  demon2 = 81.0 * nua * nua * pow(dfshape, -0.64);
  demon3 = 1.5 * phiair * phiash * gravity * xrho*xrho*xrho * sqrt((1.07-dfshape));

  velocity0 = phiash * gravity * xrho * xrho /
          (demon1 + sqrt(demon2 + demon3));

  return velocity0;
}



/* ---------------- frho ------------------- */
/* this function calculates the expected fraction of particles
  in a given grainsize class (xrho) assuming a normal
  distribution in phi units about the mean, dmean,
  with standard deviation dsigma.
*/

double frho(double xrho, double dmean,  double dsigma) {
```

```
  double func_rho;
  double demon1, demon3, demon2;

  demon1  = 1.0 / (2.506628 * dsigma);
  demon3  = xrho - dmean;
  demon2  = exp(-demon3*demon3 / (2.0*dsigma*dsigma));
  func_rho = demon1 * demon2;

  if (func_rho >= 0.0) {
    return func_rho;
  }
  else {
    printf("error in ash size distribution - method frho");
    return func_rho;
  }
}
/* ---------- f ---------- */
/* function f calculates the epected mass of ash to accumulate in a
given location xspace, yspace (cm from the volcano) in gm / cm^2
*/

double f(double xspace, double yspace, double zspace, double xrho, double u, double
total_ash_mass, double dmean, double dsigma, double phiash, double dfshape, double dbeta,
double w0, double h) {
  double ash_fall;
  double ashdiam;
  double t1, t2;
  double demon1, demon2, demon3, demon4;

  ashdiam = phi2cm(xrho);
  t1 = t(zspace, ashdiam, dfshape,phiash);
  t2 = ts(zspace);
    demon1 =  5.0 * total_ash_mass * p(zspace,ashdiam, dbeta, w0, h, dfshape, phiash) *
frho(xrho,dmean,dsigma);
  demon2 =  8.0 * PI * c * pow((t1+t2), 5.0/2.0);
  demon3 = -5.0 * ((xspace-u*t1) * (xspace-u*t1) + yspace * yspace);
  demon4 =  8.0 * c * pow(t1+t2, 5.0/2.0);
```

```
  ash_fall = demon1 / demon2 * exp(demon3/demon4);

  if (ash_fall >= 0.0) {

    return ash_fall;
  }
  else {
    printf("h: %f, zspace: %f, ashdiam: %f\n",h, zspace,ashdiam);
    return ash_fall;
    }
}

/* ---------- AllocateOneDimArray ---------- */
/* allocates a one dimensional array of doubles (using malloc).
  length is the size of the array.
*/
double *AllocateOneDimArray(int length)
{
 double *a;
 a=(double *)malloc((unsigned) (length)*sizeof(double));
 if (!a) fprintf(stderr, "Allocation failure\n");
 return a;
}

/* ---------- FreeOneDimArray ---------- */
/* free a one dimensional array of doubles.
  length is the size of the array.
*/
void FreeOneDimArray(double *a, int length)
{
        free((char*)(a));
}

/* ---------- AllocateTwoDimArray ---------- */
/* allocates a one dimensional array of doubles (using malloc).
  length is the size of the array.
*/

double **AllocateTwoDimArray(int ilength, int jlength)
```

```
{
 int i;
 double **a;
 a=(double **)malloc((unsigned) (ilength)*sizeof(double));
 if (!a) fprintf(stderr, "Allocation failure\n");
 for(i=0; i<=ilength;i++)
 {
  a[i]=(double *)malloc((unsigned) (jlength)*sizeof(double));
  if (!a[i]) fprintf(stderr, "Allocation failure\n");
 }
 return a;
}

/* ---------- FreeTwoDimArray ---------- */
/* free a two dimensional array of doubles.
   length is the size of the array.
*/

void FreeTwoDimArray(double **a, int ilength, int jlength)
{
  int i;
  for(i=ilength; i>=0;i--)
    free((char*)(a[i]));
  free((char*)(a));
}
```

****

Shell script for compiling these codes

```
#! /bin/csh -f

# Nathan Franklin
# July 6, 2000
# this shell script and the codes to be complied need
# to be located in $HOME/pvm3/examples
```

```
gcc -g -I../../include -malign-loops=2 -malign-jumps=2 -malign-functions=2 -DLINUX
-DPVMDPATH=\"pvmd\" -DPVMDFILE=\"/usr/bin/pvmd\"
-DPVM_DEFAULT_ROOT=\"/usr/lib/pvm3\" -DOVERLOADHOST -DSYSVSIGNAL
-DNOWAIT3 -DRSHCOMMAND=\"/usr/bin/rsh\" -DNEEDENDIAN -DFDSETNOTSTRUCT
-DHASERRORVARS -DCTIMEISTIMET -DSYSERRISCONST -o tephra_varwind_master1.1
tephra_varwind_master1.1.c -L../../lib/LINUX -Wall -lpvm3

gcc -g -I../../include -malign-loops=2 -malign-jumps=2 -malign-functions=2 -DLINUX
-DPVMDPATH=\"pvmd\" -DPVMDFILE=\"/usr/bin/pvmd\"
-DPVM_DEFAULT_ROOT=\"/usr/lib/pvm3\" -DOVERLOADHOST -DSYSVSIGNAL
-DNOWAIT3 -DRSHCOMMAND=\"/usr/bin/rsh\" -DNEEDENDIAN -DFDSETNOTSTRUCT
-DHASERRORVARS -DCTIMEISTIMET -DSYSERRISCONST -o tephra_varwind_slave1.1
tephra_varwind_slave1.1.c -L../../lib/LINUX -Wall -lm -lpvm3

mv tephra_varwind_master1.1 ~beowulfuser/nathan/.
mv tephra_varwind_slave1.1 ~beowulfuser/nathan/.


**********************************************************************

/*
** Program name:  tephra_varwind_slave1.1.c
**
** Purpose:   To calculate ash deposition. This program is started
**            by a master program.  It recieves the variable and grid
**            data from the master and then computes the calculations.
**          It returns the answers to the master and quits.
**
*/



#include <stdio.h>
#include <stdlib.h>
#include "pvm3.h"
#include <math.h>

// ----------------------------------------
// |   Global variables and constants    |
```

```
// ----------------------------------------

#define PI              3.141592654
#define NUMBER_OF_VARIABLES 10  //number of variables = 12 variables
#define SIZE_GRIDDIMENSIONS  5
#define XSTEPS 300  //# steps for integration over the eruption column
#define YSTEPS 100  //# steps for integration over the particle size range

//following are assumed const.
double  c      = 400.0;       // eddy diffusivity in the atmosphere
double  phiair  = 0.001293;   // density of air
double  nua    = 0.00018;     // dynamic viscosity of air
double  gravity = 980.0;      // gravity

// --------------------
// |  Functions    |
// --------------------

//grid():         Makes the grid
int   grid(double *data, double *input_variables, int n_variables,
        double **wind, int i_wind, double xorigin,
        double xsteps, double yorigin, double ysteps,
        double spacing, int n_slaves, int n_me);

//tephra_accumulation():      Used in grid() to find tephra accumulation
double tephra_accumulation(double xpos, double ypos, int where, double **wind,
                  double total_ash_mass, double ymin, double ymax,
                  double dmean, double dsigma, double phiash,
                  double dfshape, double dbeta, double w0, double h);

//functions used by tephra_accumulation to find accumulation:
double phi2cm( double xx );
double w( double zspace, double w0 , double h );
double p(double zspace, double xrho, double dbeta,
      double w0, double h, double dfshape, double phiash);
double ts( double zspace );
double t( double zspace, double xrho, double dfshape, double phiash );
double v0( double xrho, double dfshape, double phiash );
double frho( double xrho, double dmean, double dsigma );
```

```
double f(double xspace, double yspace, double zspace, double xrho,
        double u, double total_ash_mass,  double dmean,  double dsigma,
        double phiash,  double dfshape,  double dbeta,  double w0,  double h);
void   average_wind(double wind_av[][2], int where, double **wind, int xsteps, double xdelt);


//functions for allocating and freeing arrays of doubles
double *AllocateOneDimArray(int length);
void   FreeOneDimArray(double *a, int length);
double **AllocateTwoDimArray(int ilength, int jlength);
void   FreeTwoDimArray(double **a, int ilength, int jlength);


int main( int argc,  char *argv[])
{

// -------------------------------
// |   Setting up variables    |
// -------------------------------


// PVM variables:
//---------------
   int mytid;      /* my task id */
   int tids[32];   /* task ids   */
   int me, nproc, master, msgtype;
   char buf[100];

// Tephra program variables:
//--------------------------
   // Grid dimensions
   int length_data_array;
   int n_variables;
   int i_wind, j_wind;
   int i;

   double x_origin;
   double x_steps;
   double y_origin;
```

```
   double y_steps;
   double spacing;

   double *arraygrid;  //array of output grid
   double *variables;  //array of input variables
   double **wind;      //wind array
   double grid_dimensions[SIZE_GRIDDIMENSIONS];    //array of grid dimensions


// ---------------------------------
// | Receive data from the master |
// ---------------------------------

   master = pvm_parent();
   mytid = pvm_mytid();
   msgtype = 0;            // setting msgtag to 0. Master should
                           // be also broadcasting with 0.
   pvm_recv( -1, msgtype );

   pvm_upkint(&nproc, 1, 1);
   pvm_upkint(tids, nproc, 1);


   /* Determine which slave I am (0 -- nproc-1) */
   for( i=0; i<nproc ; i++ ){
     if( mytid == tids[i] )
         me = i;
   }
   pvm_recv(master, 4);
   pvm_upkint( &n_variables, 1, 1);
   variables  = AllocateOneDimArray(n_variables);

   pvm_upkdouble( &variables[0], n_variables, 1 );
   pvm_upkint( &i_wind, 1, 1);
   pvm_upkint( &j_wind, 1, 1);
   wind      = AllocateTwoDimArray(i_wind,j_wind-1);
   pvm_upkdouble( &wind[0][0], i_wind*j_wind, 1 );
   pvm_upkdouble( &grid_dimensions[0], 5, 1 );
```

```
// --------------------------------
// | Do calculations with data    |
// --------------------------------


   spacing=grid_dimensions[4];
   x_steps=grid_dimensions[1];
   x_origin=grid_dimensions[0];
   y_origin=grid_dimensions[2];
   y_steps=grid_dimensions[3];

   //allocate output array:  length= # grid points * # of runs * 3 (x,y,acc)
   length_data_array = 3 * x_steps*y_steps*(n_variables/NUMBER_OF_VARIABLES);
   arraygrid  = AllocateOneDimArray(length_data_array);

   //get grid -> arraygrid
  grid(arraygrid, variables, n_variables, wind, i_wind, x_origin, x_steps,y_origin, y_steps, spacing,
nproc, me);

   //get my hostname
   gethostname (buf, 64);


// --------------------------------
// |   Send results to master     |
// --------------------------------
   pvm_initsend( PvmDataDefault );
   pvm_pkint( &me, 1, 1 );
   pvm_pkstr(buf);
   pvm_pkint( &length_data_array, 1, 1 );
   //debug
   /*arraygrid[0]=wind[0][0];
   arraygrid[1]=wind[0][1];
   arraygrid[2]=wind[0][2];
   arraygrid[3]=wind[1][0];
   arraygrid[4]=wind[1][1];
   arraygrid[5]=wind[1][2];
   arraygrid[6]=wind[2][0];
   arraygrid[7]=wind[2][1];
```

```
   arraygrid[8]=wind[2][2];
   */
   pvm_pkdouble( &arraygrid[0], length_data_array, 1 );
   pvm_pkdouble( &arraygrid[0], length_data_array, 1 );
   msgtype = me;
   pvm_send( master, msgtype );


   FreeOneDimArray(grid_dimensions,SIZE_GRIDDIMENSIONS);
   FreeTwoDimArray(wind,i_wind,j_wind);
// --------------------------------
// |  Program finished...exit  |
// --------------------------------
   printf("end of program\n");
   pvm_exit();

   return 1;
}
```

```
int grid(double *data, double *input_variables, int n_variables, double **wind, int i_wind, double
xorigin, double xsteps, double yorigin, double ysteps, double spacing, int n_slaves, int n_me)
{
  // following are the 10 input variables
  double total_ash_mass,ymin, ymax, dmean, dsigma, phiash;
  double dfshape, dbeta, w0, h;

  int i,j;
  int x_step_count,y_step_count,number_of_runs;
  int grid_xsteps;
  int grid_ysteps;

  double xpos, ypos;
```

```
  grid_xsteps=xsteps +0;
  grid_ysteps=ysteps;
  xpos=xorigin;
  ypos=yorigin;

// ------------------------------------------------
// | Parallel:  Take my portion of variable set   |
// ------------------------------------------------
  number_of_runs = n_variables/NUMBER_OF_VARIABLES; // # of variables
                                    // divided by # of variables
                                    // in a run


  i=0;
  x_step_count=0;

        while(x_step_count<grid_xsteps)
        {

              y_step_count=0;

             while(y_step_count<grid_ysteps)
              {
            //move to next point in grid
            xpos=xorigin + (x_step_count*spacing);
                  ypos=yorigin + (y_step_count*spacing);

            //at this point find tephra accumulation
            //     for each set of input variables
            for(j=0;j<number_of_runs;j++)
                  {
               total_ash_mass=input_variables[0+((j)*NUMBER_OF_VARIABLES)];
               ymin   =input_variables[1+((j)*NUMBER_OF_VARIABLES)];
               ymax   =input_variables[2+((j)*NUMBER_OF_VARIABLES)];
               dmean  =input_variables[3+((j)*NUMBER_OF_VARIABLES)];
               dsigma =input_variables[4+((j)*NUMBER_OF_VARIABLES)];
               phiash =input_variables[5+((j)*NUMBER_OF_VARIABLES)];
               dfshape =input_variables[6+((j)*NUMBER_OF_VARIABLES)];
```

```
                dbeta  =input_variables[7+((j)*NUMBER_OF_VARIABLES)];
                w0     =input_variables[8+((j)*NUMBER_OF_VARIABLES)];
                h      =input_variables[9+((j)*NUMBER_OF_VARIABLES)];


                        data[i]=xpos;
                        i++;
                        data[i]=ypos;
                        i++;
                        data[i]=tephra_accumulation(xpos,  ypos,  j, wind, total_ash_mass, ymin,
ymax, dmean, dsigma, phiash, dfshape, dbeta, w0, h);
              i++;

             }//end of for

                y_step_count++;
               }//end of 'y' while

               x_step_count++;
         }//end of 'x' while
  return i;
}


/* ----------------- tephra_accumulation -------------------- */
/* calculates the epected mass (gm/cm^2) of ash to accumulate in a
given location xspace, yspace using function f and integrating over the column hieght and particle
size range.
*/
double tephra_accumulation(double xpos, double ypos, int where, double **wind, double
total_ash_mass, double ymin, double ymax, double dmean, double dsigma, double phiash, double
dfshape, double dbeta, double w0, double h)
{
      double xsectionwidth;
      double ysectionwidth;
         double xmin, xmax;
         double xstepwidth;
      double ystepwidth;
         double new_xpos0, new_ypos0;
```

```
        double new_xpos1, new_ypos1;
        double x0, y0, x1, xi, y1, yi,ans;
      double total;
      double u0,udir0;
      double u1,udir1;
      double wind_av[XSTEPS][2];
      int xsteps, ysteps, yyi, xxi;
    // define the limits of integration
    //---------------------------------
    // xmin is the elevation of the vent
    // xmax is the total height of the eruption column
    // ymin and ymax are minimum and maximum
    // grainsizes, respectively

     // do the integration over the eruption
        // column height in 300 steps
        xsteps = XSTEPS;

        // do the intergration over the particle
        // size range in 100 steps
        ysteps = YSTEPS;

        xmin= 0.0;
        xmax =h;
      xsectionwidth = xmax - xmin;
      ysectionwidth = ymax - ymin;
      xstepwidth = xsectionwidth / xsteps;
      ystepwidth = ysectionwidth / ysteps;
        average_wind(wind_av, where, wind, xsteps, xstepwidth);

        // initialize the summation for the
        // contribution of each d phi and
        // each dz to the total accumulation
        // of tephra on the ground at the
        // point xspace, yspace
        total  = 0.0;


        //do the integration
```

```
//-------------------
   for(yyi=1; yyi < ysteps; yyi++) {

         // define the small slice d phi

      yi = (double)yyi;
      y0 = ymin + yi * ystepwidth;
      y1 = ymin + (yi+(double)1.0) * ystepwidth;

         for(xxi=1; xxi < xsteps; xxi ++) {

            // define the small slice dz
         xi = (double)xxi;
         x0 = xmin + (xi) * xstepwidth;
         x1 = xmin + (xi+(double)1.0) * xstepwidth;

            u0=wind_av[xxi][0];
            u1=wind_av[xxi+1][0];
            udir0=wind_av[xxi][1];
            udir1=wind_av[xxi+1][1];

            //define the coordinate system with
            // respect to wind direction
            //---------------------------------
            // here we are just assuming that
            // the wind is varying N or south of
            // a western direction, need to be genrealized

          new_xpos0 = xpos * cos(udir0) + ypos * sin(udir0);
        new_ypos0 = -xpos*sin(udir0) + ypos*cos(udir0);
            new_xpos1= xpos * cos(udir1) + ypos * sin(udir1);
        new_ypos1= -xpos*sin(udir1) + ypos*cos(udir1);



         // calculate the value of the ash accumulation  at
         // the corners  of the slice, and sum
        ans =  f(new_xpos0, new_ypos0, x0, y0, u0, total_ash_mass, dmean, dsigma, phiash,
dfshape, dbeta, w0, h)
```

```
                + f(new_xpos1, new_ypos1, x1, y0, u1, total_ash_mass, dmean, dsigma, phiash,
dfshape, dbeta, w0, h)
                + f(new_xpos0, new_ypos0, x0, y1, u0, total_ash_mass, dmean, dsigma, phiash,
dfshape, dbeta, w0, h)
                + f(new_xpos1, new_ypos1, x1, y1, u1, total_ash_mass, dmean, dsigma, phiash,
dfshape, dbeta, w0, h);
            total += ans;
          }
        }

    // complete the integration by multiplying by dz and d phi
    // and dividing by 4, since 4 corners were used
    total *= xstepwidth * ystepwidth / 4.0;

    //all finished: return the answer
    return total;
}


/* ---------------- average_wind -------------------- */
/* function average_wind finds the average wind (wind_av[h][0]) and wind direction
  (wind_av[h][1]) for ash leaving the column at at specific heights along the column.
*/

void average_wind(double wind_av[][2], int where, double **wind, int xsteps, double xdelt)
{
        int i,j;
        double sum_x=0;
        double sum_y=0;
        j=1;
    //windfield[0]= number of windfields.
    //windfield[1]= starting height
    //windfield[2]= u
    //windfield[3]= udir
    //windfield[4]= starting height ...
    //windfield[5]= u
    //windfield[6]= udir
    //calculate average wind at heights for the itegration over the column.
    for(i=1;i < xsteps;i++) {
```

```
        // if there are more windfields and if the current height
        // is greater than the starting height of the next windfied:
        // jump to next wind field
        //   * this step assumes that the height of each individual wind
        //     field is greater than the
        if( (wind[where][0] * 3) >= (j+3)  &&  (i*xdelt) >= wind[where][j+3])
         j=j+3;

          //calculate the sum of the x components and the sum
          // of the y components of the wind vector
             sum_x= (wind[where][j+1] * cos(wind[where][j+2])) + sum_x;
             sum_y= (wind[where][j+1] * sin(wind[where][j+2])) + sum_y;

          //Use these components to calculate the average speed
          // and average direction of the wind exprienced for ash
          // departing at this column height.

          wind_av[i][0]=( sqrt(sum_x*sum_x + sum_y*sum_y) ) / i;
          wind_av[i][1]= atan2(sum_y,sum_x);
      }
}


/* ----------------- phi2cm ---------------------- */
/* function phi2cm converts the ash diameter from
   units of phi to cm
*/

double phi2cm(double xx) {
  double cms;

  cms = 0.1 * 1.0 / pow(2, xx);
  return cms;
}

/* ----------------- w ------------------- */
/* this function estimates the velocity
  as a function of height in the
```

```
  eruption column, given a
  linear decrease in velocity with
  height and an initial velocity
  of w0
*/

/* returns the upward particle velocity  */

double w(double zspace, double w0, double h) {
  double particle_velocity;

  particle_velocity = w0 * (1.0 - zspace/h);

  // following only ocurrs as round-off error
 if (zspace > h) particle_velocity = 0.0;
  return particle_velocity;
}


/* ----------------- p -------------------- */
/* this function calculates the probability
  density for diffusion of ash out of the
  eruption column. Some variation from Suzuki
  implemented to conserve mass

  depends on beta, w0, w(z), v0(xrho)
  where w0 is the eruption velocity in cm/s
  w(z) in the upward particle velocity at height z (cm/s)
  v0(xrho) is the particle settling velocity (cm/s)
  h is the column height (km)

  integrating fz from 0 to h should give unity

*/

double p(double zspace, double xrho, double dbeta, double w0, double h, double dfshape, double
phiash) {
  double y, y0;
  double prob_dens_func;
```

```
  double voo;
  double demon1, demon2;
  voo = v0(xrho, dfshape, phiash);
  y  = dbeta * w(zspace, w0, h) / voo;
  y0 = dbeta * w0 / voo;
  demon1 = dbeta * w0 * y * exp(-y);
  demon2 = voo*h * (1.0-(1.0+y0) * exp(-y0));
  prob_dens_func = demon1/demon2;

  return prob_dens_func;
}



/* ----------------- ts ---------------------- */
/* function ts determines the particle diffusion time const
  in the atmosphere
  where zspace = height of the particle, converted here from
              km to cm, this should be wrt sea-level,
              zspace is given here in cm because
              of the units of eddy diffusivity
       c = eddy diffusivity in the atmosphere
          given as in cgs, e.g., 400 cm/s^(5/2)

  returns the particle diffusion time
*/

double ts (double zspace) {
  double particle_diffusion_time;
  double demon1, demon2;

  zspace *= 1.0e+5;
  demon1  = 5.0 * zspace * zspace;
  demon2  = 288.0 * c;
  particle_diffusion_time = pow(demon1/demon2, 2.0/5.0);

  return particle_diffusion_time;
}
```

```
/* ----------------- t -------------------- */
/* function t calculates the particle fall time
   in the atmposphere
   where: zspace = height in the atmosphere, here in km
                   wrt to sealevel, because the const.
                   -0.0625 is in units of 1/km
*/
double t (double zspace, double xrho, double dfshape, double phiash) {
   double particle_fall_time;
   double demon1, demon2;

   demon1 = (1.0 - exp(-0.0625*zspace)) / v0(xrho, dfshape, phiash);
   demon2 = pow(demon1, 0.926);
   particle_fall_time = 0.752e6 * demon2;
   return particle_fall_time;
}


/* ----------------- v0 -------------------- */
/* return a value for the particle terminal velocity
   at sea level, given the particle diameter, passed as x
   note that the shape factor does not depend on x
   note that the ash dens does not depend on x
*/

double v0 (double xrho, double dfshape, double phiash) {

   double velocity0;
   double demon1, demon2, demon3;

   demon1 = 9.0 * nua * pow(dfshape, -0.32);
   demon2 = 81.0 * nua * nua * pow(dfshape, -0.64);
   demon3 = 1.5 * phiair * phiash * gravity * xrho*xrho*xrho * sqrt((1.07-dfshape));

   velocity0 = phiash * gravity * xrho * xrho /
           (demon1 + sqrt(demon2 + demon3));

   return velocity0;
}
```

```
/* ---------- frho ---------- */
/* this function calculates the expected fraction of particles
   in a given grainsize class (xrho) assuming a normal
   distribution in phi units about the mean, dmean,
   with standard deviation dsigma.
*/

double frho(double xrho, double dmean,  double dsigma) {

   double func_rho;
   double demon1, demon3, demon2;

   demon1   = 1.0 / (2.506628 * dsigma);
   demon3   = xrho - dmean;
   demon2   = exp(-demon3*demon3 / (2.0*dsigma*dsigma));
   func_rho = demon1 * demon2;

   if (func_rho >= 0.0) {
      return func_rho;
   }
   else {
      printf("error in ash size distribution - method frho");
      return func_rho;
   }
}
/* ---------- f ---------- */
/* function f calculates the epected mass of ash to accumulate in a
given location xspace, yspace (cm from the volcano) in gm / cm^2
*/

double f(double xspace, double yspace, double zspace, double xrho, double u, double
total_ash_mass, double dmean, double dsigma, double phiash, double dfshape, double dbeta,
double w0, double h) {
   double ash_fall;
   double ashdiam;
   double t1, t2;
   double demon1, demon2, demon3, demon4;
```

```
  ashdiam = phi2cm(xrho);
  t1 = t(zspace, ashdiam, dfshape,phiash);
  t2 = ts(zspace);
    demon1 =  5.0 * total_ash_mass * p(zspace,ashdiam, dbeta, w0, h, dfshape, phiash) *
frho(xrho,dmean,dsigma);
  demon2 =  8.0 * PI * c * pow((t1+t2), 5.0/2.0);
  demon3 = -5.0 * ((xspace-u*t1) * (xspace-u*t1) + yspace * yspace);
  demon4 =  8.0 * c * pow(t1+t2, 5.0/2.0);

  ash_fall = demon1 / demon2 * exp(demon3/demon4);

  if (ash_fall >= 0.0) {

    return ash_fall;
  }
  else {
    printf("h: %f, zspace: %f, ashdiam: %f\n",h, zspace,ashdiam);
    return ash_fall;
    }
}

/* ---------------- AllocateOneDimArray ------------------- */
/* allocates a one dimensional array of doubles (using malloc).
  length is the size of the array.
*/
double *AllocateOneDimArray(int length)
{
 double *a;
 a=(double *)malloc((unsigned) (length)*sizeof(double));
 if (!a) fprintf(stderr, "Allocation failure\n");
 return a;
}

/* ---------------- FreeOneDimArray ------------------- */
/* free a one dimensional array of doubles.
  length is the size of the array.
*/
void FreeOneDimArray(double *a, int length)
{
```

```
        free((char*)(a));
}

/* ----------------- AllocateTwoDimArray -------------------- */
/* allocates a one dimensional array of doubles (using malloc).
  length is the size of the array.
*/

double **AllocateTwoDimArray(int ilength, int jlength)
{
 int i;
 double **a;
 a=(double **)malloc((unsigned) (ilength)*sizeof(double));
 if (!a) fprintf(stderr, "Allocation failure\n");
 for(i=0; i<=ilength;i++)
 {
  a[i]=(double *)malloc((unsigned) (jlength)*sizeof(double));
  if (!a[i]) fprintf(stderr, "Allocation failure\n");
 }
 return a;
}

/* ----------------- FreeTwoDimArray -------------------- */
/* free a two dimensional array of doubles.
  length is the size of the array.
*/

void FreeTwoDimArray(double **a, int ilength, int jlength)
{
  int i;
  for(i=ilength; i>=0;i--)
   free((char*)(a[i]));
  free((char*)(a));
}
```

Scientific Notebook 115E
20-1402-461
Inititals: CC

Chuck Connor
Aug 29, 2000

Probabilistic Modeling of the Dispersion and Accumulation of Volcanic Ash using a "Not-So-High-Performance" PC Cluster

N. Franklin, C. Connor, B. Winfrey, L. Connor, J. De la Esperilla, and R. Martin

To be submitted to Computers in the Geosciences

.

ABSTRACT

Information potentially subject to copyright protection was redacted from Section 2, pages 1 through 20 of this scientific notebook. The redacted material is from the reference listed above.

Scientific Notebook 115E
20-1402-461
Inititals: CC

Chuck Connor
Aug 30, 2000

The following summarizes continued model development for the Cerro Negro electromagnetic data set described earlier in 115E

SHALLOW THERMAL STRUCTURES BENEATH CERRO NEGRO VOLCANO USING TRANSIENT ELECTROMAGNETICS

S. K. Sandberg[1], C. B. Connor[2], P. C. LaFemina[2,3], J. A. Saballos[4], and W. Strauch[4]

[1]University of Southern Maine, 37 College Avenue, Gorham, Maine, U.S.A.
[2]Center for Nuclear Waste Regulatory Analyses, Southwest Research Institute, San Antonio, Texas, U.S.A.
[3]Now at University of Miami
[4]INETER

Information potentially subject to coyright protection was redacted from this location (Section 3 pages 1 through 8). The redacted material may have been published. Additional information is not known.

Scientific Notebook 115E
20-1402-461
Inititals: CC

Chuck Connor
Aug 30, 2000

CC 9/20/00 CC 9/20/00

The following 55 pages are all part of section 4. They are the final draft of the Woods -Bokhove paper submitted to the NRC as an IM in the IA KTI.

Explosive magma-air interactions by volatile-rich basaltic melts in a dike-drift geometry

Onno Bokhove[1] and Andrew W. Woods[2]

School of Mathematics, Bristol

Short title: EXPLOSIVE MAGMA-AIR INTERACTIONS

Abstract.

We study the ascent of relatively volatile-rich basaltic magma through a vertical dike that intersects a horizontal tunnel or drift of comparable cross-sectional area to the dike and located about $300 - 400\,m$ below the surface. This process is a simplified representation of some aspects of the possible interaction of a basaltic fissure eruption either with a sub-surface, man-made waste-repository, or with a natural sub-surface cavern, such as the limestone karsts in China. In the model, we assume that prior to breakthrough of the dike, the tunnel is maintained at atmospheric pressure. We examine the decompression and flow that develops following breakthrough into the tunnel. The model provides an averaged one-dimensional picture of the flow, averaging over the prescribed dike and tunnel geometry. It is based on the assumption that the basaltic magma remains in chemical equilibrium with the dissolved volatile phase. This volatile phase is mainly water and is exsolved from the melt as the mixture decompresses. The model predicts that for 2 weight percent water, the magma-gas mixture decompresses rapidly into the drift, and as it expands it generates a shock wave in the air displaced down the tunnel. This wave travels at a speed of order $500\,m/s$. If the tunnel end is closed, the shock wave is reflected between the tunnel end and magma-air interface and may be amplified by a factor of $15 - 50$, with a high pressure region developing at the end of the tunnel. Owing to the difference in density and speed of sound in the air and the magma-gas mixture, a complex series of interacting shock waves develops near the end of the tunnel. The results indicate that due to this explosive expansion of magma

in the drift, a region of maximum pressure in the drift may develop far from the dike.

1. Introduction

There has been considerable interest in quantifying eruption recurrence and long term eruption forecasting owing to the proposal to site a nuclear waste repository at Yucca Mountain, Nevada. This site is located in an area that has experienced relatively recent volcanic activity in the geological record. Indeed, the Lathrop Wells cinder cone is the product of the most recent eruptions in the vicinity of Yucca Mountain (Connor *et al.*, 1997; and Heizler *et al.*, 1999). New methodology for predicting volcanic recurrence rates, based on the historical record and geological constraints, have been developed by Condit and Connor (1996) and Connor *et al.* (1993, 1995, 1997, 1999). This work has identified that in the immediate area of Yucca Mountain, there is a probability of volcanic activity in the range $10^{-3} - 10^{-4}$ over the next 10^4 years. Such estimates of volcanic activity are significant, and lead to the interesting scientific question of the magma flow that may ensue if relatively volatile-rich basaltic magma, ascending in a dike, were to meet an underground tunnel, at a depth of order $300 - 400\,m$, but initially maintained at atmospheric pressure. This problem is of great interest, because of the possible effects on repository performance, *i.e.*, the possible release of nuclear waste into the environment. The processes involved in such magma-dike-tunnel interaction may be extremely complex. In addition to the rapid decompression of the magma-volatile mixture, the geometry of the dike and tunnel may also evolve as pressures near the dike tip rapidly decrease. In this study we examine a simplified picture of such an event, examining some of the complex flow processes that might develop subject to the

assumption that the dike and tunnel geometry are fixed or prescribed. This approach follows much of the established literature in which magma ascent is studied (Wilson et al., 1980; Wilson and Head, 1981; Jaupart and Allègre 1991, Woods, 1995), and enables us to develop some fundamental insights into aspects of the basic magma physics. As well as being of interest for the problem at hand, our study may also provide new insights into observations of magmatic intrusions in limestone karsts, which have recently been reported in China (Wu, personal communication, 1999). Our work is also related to a curious event during the eruption of Krafla volcano in Iceland, 1977, in which a horizontal sill intersected a vertical geothermal bore-hole and triggered a small explosive eruption (Larsen *et al.*, 1979).

We initiate our model at the point when the dike, ascending through the crust, breaks through into the tunnel or drift. At this stage we expect that there will be a rapid decompression of the magma, and a dramatic increase in the flow rate. Prior to breakthrough, the magma in the dike will be driven upward with a typical speed of order $1 - 2\,m/s$ by a combination of chamber overpressure and magma buoyancy, typically of the order of $1 - 2\,MPa$. Following breakthrough, the mixture will decompress and accelerate rapidly, reaching speeds of order $100\,m/s$. The ensuing flow is expected to resemble dynamics in a classical shock tube problem. The latter consists of initially quiescent flow with different pressures at either side of a diaphragm that is removed instantly (e.g., Whitham, 1974). The shock-tube analogy arises because the decompressing magma is effectively compressible due to the presence of the water-vapor

bubbles, and because the original speed of the magma in the dike is low compared with the ensuing speed of the explosive volcanic flow in the air-filled drift. Although we will consider magma-air interactions in a more complex geometry where gravity plays a role, many essential aspects of the flow evolution of magma and air are captured by classical shock tube and idealized shock reflection problems. We therefore explore simplified models to develop some basic understanding before moving on to numerical simulations.

As a first step in developing a leading-order description of the volcanic fluid dynamics in a dike-tunnel setting, the complex three-dimensional fluid dynamics of magma and air is simplified into a one-dimensional flow-tube model in an idealized geometry. The basaltic magma and exsolved volatiles form a multiphase mixture; following Wilson and Head (1991), we parameterize this as a pseudo-one-phase fluid with a monotonic relationship between pressure p and density ρ. Hence the magma is effectively compressible from a macroscopic view point. Although there has been some analysis of the effects of the relative motion between the bubbles, the bulk phenomena are comparable (Vergnoille and Jaupart, 1986). For simplicity we therefore neglect these effects herein.

Modeling magma-air interaction requires some simplification of the geometry of the system; in the context of Yucca Mountain, there are plans to build a series of parallel tunnels, so that only a sector of the ascending magma would be tapped off into any given tunnel. We therefore focus on the flow in a single tunnel using the following simplification. A magma dike of characteristic width of the order of $1 - 2\,m$ ascends

through the rock from a magma chamber deep in the Earth's crust. At a certain time, it encounters a series of tunnels typically spaced about $80\,m$ apart at a depth of $300 - 400\,m$. This spacing defines the length of dike that interacts with one tunnel (Fig. 1a). The geometry is then mapped into a shock tube-type problem by assuming that the dike breaks through into the tunnel at time $t = 0$ either instantly or more slowly by increasing the connecting area over time. The geometry is sketched in Fig. 1b.

Fracture mechanics in dike or crack propagation is complicated and poorly understood. Crack propagation in an ascending dike is generally argued to be magma or fluid controlled. Lister (1990) and Lister and Kerr (1991) show that a steadily ascending dike with constant flow rate has a bulbuous head in which the width of the dike tip initially increases parabolically over $20 - 100\,m$, depending on the particular parameters. The magma is incompressible and in the dike tip a suction region of vapor forms, which, deep down into the crust, is underpressured relative to the ambient lithostatic pressure. It is the negative elastic stress required to lower the lithostatic pressure to the saturated vapor pressure of the liquid in the dike tip. Rubin (1993, 1995) considered similarity solutions for a growing dike when the excess pressure and dike tip suction remain constant. A cusped crack lies ahead of the magma, which is taken as incompressible at the depths Rubin is considering. The ratio of the length of this cavity behind the crack over the dike length diminishes rapidly as function of the ratio of the suction pressure in the cavity to the excess pressure. More recently, Mériaux *et al.* (1999) argued that crack propagation is dominated by the magma fluid dynamics, and that the damaged

zone in the rock around the tip is narrow, in support of the original predictions of Lister (1990). However, the solutions of Lister *et al.* correspond to an idealized situation. In reality, the stress distribution in the rock will change due to the presence of the tunnels and hence the ascent of the dike will become unsteady. In particular, a dike approaching a series of drifts is thought to rotate into the direction orthogonal to the local least compressive stress (*e.g.* Rubin 1995). Although one could consider the higher-order effects of the change in stress distribution by the drift as a far-field contribution to the ascending dike, we prescribe the (time-dependent) geometry of the dike walls from the onset —this may be considered as a worst-case scenario. Following the results of Lister (1990), we consider a dike tip that narrows over about $40\,m$ to a nozzle area at the dike-drift intersection. We present both simulations in which the nozzle area is fixed in time and ones in which the nozzle acts as an iris and opens from a very small value to its larger final value. The latter is a very simple parameterization to examine the effect of the opening or closing of the dike-drift intersection. As far as we know, no model has been developed in which a (steadily) ascending dike is modeled with compressible bubbly magma in the nose and denser, perhaps incompressible, magma at lower depths. Our geometrical picture is based on the assumption that the dike propagation is controlled by magma flow. We note that in some situations, a dry, *i.e.*, empty fracture may propagate ahead of the magma-filled dike (Rubin 1993, 1995). This is not considered in the present work, since we focus on a worst-case scenario.

Using the left-right symmetry in a dike that intersects a drift in a perpendicular

manner, we only consider one half of the dike-drift configuration. As a further simplification of the geometry, a one-dimensional flow-tube model is constructed by smoothly connecting the flow from dike into drift as is sketched in Fig. 1c. In the flow-tube model, all fluid variables such as velocity, density, pressure and energy depend only on time t and on a curvilinear spatial coordinate ξ_1 along the average flow path. The flow-tube configuration in Fig. 1c displays the connecting arc between dike and drift. The dashed-dotted line indicates the central line of flow ξ_1. The dynamics in the flow-tube model consists of compressible dynamics for basaltic magma and gas dynamics for air. Alternatively, the flow-tube model can be derived by averaging the three-dimensional compressible flow equations over the cross-sectional areas of the flow tube. To obtain closure, the resulting Reynolds stress terms are parameterized to leading order by simple frictional parameterizations. In its inviscid limit, the one-dimensional flow-tube model is equivalent to flow in a circular tube with varying cross-section and gravitational influence. However, the introduction of frictional terms again introduces geometrical effects which lead to stronger friction in the narrow dike than in the drift. In three dimensions, the high-pressure basaltic magma is expected to shoot upward and sideways after encountering the underpressured air-filled tunnel, since the magma follows the path of least resistance and flows rapidly into the drifts. An oblique shock wave in air then develops, as in a two-dimensional shock tube problem, and reflects repeatedly against the drift walls. When the end of a drift and the dike-drift intersection are well-separated, this shock straightens after several reflections

into a quasi-one-dimensional propagating shock towards the end of the drift. After breakthrough the magma has to turn into the drift and form its own flow tube set by the intrinsic fluid dynamics. In the one-dimensional model this effective flow path is modeled geometrically by introducing a nozzle of typical area $\pi\, d\, w$; this corresponds to the area of a circular band as wide as the dike with d the diameter of the drift and w the (half)-width of the dike. In some of the simulations, this nozzle or iris is slowly opened from a very small value to its final value, corresponding to an opening dike after breakthrough. The time-varying cross-sectional area turns out to exert a hydraulic control on the dynamics in the flow-tube model.

The advantage of a one-dimensional flow-tube model above higher-dimensional fluid models lies in its simplicity. Neglecting a detailed study of the transition from vertical magma flow in the dike to a horizontal one in the drift may seem severe, but flow-tube models allow study of transient volcanic fluid dynamics for a substantially larger part of parameter space. As a result, it is therefore relatively straightforward to study the sensitivity of numerical solutions to parameter variation. We assess the sensitivity of explosive magma-air interactions as function of volatile content, overpressure, friction, and with respect to variations in the cross-sectional area between dike and drift —the latter as function of width and time. In addition, we consider simplified magma-air interactions in cases where the dike closes due the drop in pressure in the dike after breakthrough. This is done by investigating the flow after breakthrough from a dike of fixed finite length, of the order of $50\, m$ to $500\, m$. Only a finite amount of magma is then

released into the drift, even though we may expect the dike to open again at a later stage when pressure is rising near the dike-drift intersection.

The results presented herein provide fundamental insights into magma-air interactions in a dike and drift system. They represent bounding calculations on the effects of explosive magma-air interactions in man-made or natural sub-surface caverns by on the one hand simplifying the geometry and fluid-dynamics, but on the other hand compensating these simplifications by assessing a large range of parameter space. Our premise is that reality will lie somewhere within the parameter range studied.

The mathematics of a flow tube model are introduced in section 2. The dike-drift geometry and the basic processes involved are explained in 3. The fluid dynamics of two basic processes, the shock tube phase and the shock reflection phase, are explored in a series of idealized shock tube and shock reflection problems in magma and air (section 3.1). The reference simulation of magma-air interactions in the full-fledged dike-drift geometry is then interpreted in terms of these idealized processes (section 3.2). Finally, the sensitivity of these interactions is studied in a parameter study in section 4 as a function of various parameters and geometries.

2. Flow-tube model

The fluid equations of the compressible dynamics of magma and air are introduced next. These equations form the basis for our analytical and numerical investigations of explosive magma-air dynamics.

The multiphase basaltic fluid of melt and volatiles is modeled as an isothermal compressible fluid with a parameterized equation of state. The density ρ of this fluid equals the reciprocal of the volume occupied by a unit mass of the mixture of exsolved volatiles (gas), dissolved volatiles (liquid) and melt. The mass fraction of exsolved volatiles is $n(p)$. Dissolved volatiles and melt are lumped together as an incompressible mixture of mass fraction $1 - n(p)$ and an overall density σ, giving

$$\rho(p) = \left(\frac{n(p)\, R_v\, T}{p} + \frac{1 - n(p)}{\sigma} \right)^{-1}. \tag{1}$$

Here $R_v \approx 462\, J\, kg^{-1}\, K^{-1}$ is the gas constant for H_2O in the basaltic mixture, fixed temperature $T = 1000 - 1200\, K$, and lumped melt-liquid density $\sigma \approx 2500\, kg\, m^{-3}$. Bubbles and melt are assumed to be in chemical equilibrium such that Henry's law (Sparks 1978)

$$n(p) = n_0 - n_{s_H} \equiv n_0 - s_H\, p^{\beta} \tag{2}$$

applies with total volatile content $n_0 = 1.0 - 2.5$ weight percent ($wt\%$) and with Henry's constant $s_H \approx 3 \times 10^{-6}\, Pa^{-1/2}$ and $\beta \approx 1/2$ for basaltic magmas. The model is a leading-order approximation to the complicated physics of the multi-phase melt-volatile mixture. Supersaturation effects are neglected, and we assume there is negligible slip between the phases.

Eventually all the volatiles become dissolved when the pressure reaches a critical pressure p_c where $n(p = p_c) = 0$ [see (2)]. For example, given the values of s_H and β above and $n_0 = 0.01, 0.025\, wt\%$ we get $p_c = (n_0/s_H)^{1/\beta} \approx 1.11, 6.9\, MPa$. For $p > p_c$, the flow is incompressible and the speed of sound in our model is therefore infinite. In the

present study we only consider pressures $p < p_c$, although one could in principle couple the incompressible and compressible regions of the pseudo one-phase fluid modeled by (1) and (2) for $p < p_c$, and $\rho = \sigma$ for $p > p_c$, as has been done in models with stationary flow.

In the dike-drift configuration a representative $\xi_1(x, y, z)$-coordinate may be identified with corresponding cross-section $A(\xi_1, t)$, normal to ξ_1-isolines, as is illustrated in Fig. 1c by the dashed-dotted line. Any time-dependence of the cross-sectional area A aims to model the response of the rock walls to pressure and its gradients in the dike or drift, or to the movement of large objects in the drift as a consequence of pressure gradients and viscous forces. A one-dimensional system, obtained by averaging over cross sections and by neglecting flow in cross-sectional planes, is derived by considering mass and momentum density conservation in a control volume $h_1\, A(\xi_1, t)\, \mathrm{d}\xi_1$. Momentum density is defined as $\rho\, A\, u_1$ and $h_1 = h_1(\xi_1)$ is the scale factor of the curvilinear coordinate. Pressure in the ξ_1-direction acts both on the slices at ξ_1 and $\xi_1 + \Delta\xi_1$, giving a contribution $\sim -\partial(p\, A)/\partial\xi_1$, and on the flow tube walls between these slices, giving a contribution $\sim p\, \partial A/\partial\xi_1$. The gravitational force is a volume force. For the basaltic fluid we then find the following equations of motion (e.g., Whitham, 1974):

$$\begin{aligned} \frac{\partial(\rho\, A\, u)}{\partial t} + \frac{1}{h_1}\frac{\partial}{\partial\xi_1}\Big(\rho\, A\, u^2 + p\, A\Big) &= -A\, F_1 + \frac{p}{h_1}\frac{\partial A}{\partial\xi_1} - \frac{\rho\, g\, A}{h_1}\frac{\partial z(\xi_1)}{\partial\xi_1}, \\ \frac{\partial(\rho\, A)}{\partial t} + \frac{1}{h_1}\frac{\partial}{\partial\xi_1}\Big(\rho\, A\, u\Big) &= 0, \end{aligned} \tag{3}$$

where $u = u_1$ is the ξ_1-component of the velocity, and $-F_1$ the parameterized forcing and/or dissipation. In a vertical dike or conduit, we have $\xi_1 = z$ and $h_1 = 1$, and

in a horizontal tunnel $\xi_1 = x$ and $h_1 = 1$. We can regard (3) as an average of the three-dimensional compressible flow equations over cross sections in which the frictional stress terms are replaced by dissipation F_1.

It is not well understood how to model the frictional forces of the bubbly magmatic liquid owing to its complex rheology. An effective viscosity has been shown to increase with volatile exsolution and also with the pressure of bubbles (Jaupart and Allègre, 1991); an empirical parameterization of frictional dissipation in a high-viscosity magmatic foam, averaged over the cross-sectional area, and proportional to the velocity was proposed

$$F_1 = \frac{3\,\mu_0\, e^{(5-100\,n_0)}\,(1-\phi)^{-5/2}}{2\,L_e^2}\,u \tag{4}$$

with L_e a typical length scale, e.g., the width of the dike or the radius of drift or conduit, and $\mu_0 = 10 - 100\,kg\,m^{-1}\,s^{-1}$. Relation (4) has validity when the vesicularity or void fraction

$$\phi = \frac{1}{1 + (1-n)\,p/(n\,R_v\,T\,\sigma)} \tag{5}$$

of the mixture remains below the point of fragmentation when $\phi < \alpha$ with $\alpha = 70 - 90\%$ (Woods, 1995). As ϕ evolves through this regime, the gas becomes the continuous phase and frictional forces diminish. So when $\phi > \alpha$, a simple parameterization for turbulent flow in a pipe was proposed by Wilson *et al.* (1980):

$$F_1 = 0.0025\,\frac{\rho\,u^2}{L_e}. \tag{6}$$

We can, of course, view these frictional parameterizations as a very crude, leading-order

(turbulent) closure for the unknown Reynolds stress terms. Both parameterizations refer through the length scale L_e to the geometry of the cross section. In the inviscid limit when $F_1 = 0$ the shape of the cross sections is arbitrary while their area is specified, but when the frictional terms described above are added the area shape is specified through a characteristic cross-sectional width L_e. In our dike-drift system L_e is five times smaller in the dike than in the drift; friction in the dike is thus five to 25 times larger than in the drift.

Similarly, the one-dimensional, averaged compressible equations of motion for air (e.g., Batchelor, 1967) are:

$$\begin{aligned}
\frac{\partial(\rho_a A u_a)}{\partial t} + \frac{1}{h_1}\frac{\partial}{\partial \xi_1}\left(\rho_a A u_a^2 + p_a A\right) &= -A F_a + \frac{p_a}{h_1}\frac{\partial A}{\partial \xi_1} - \frac{\rho_a g A}{h_1}\frac{\partial z(\xi_1)}{\partial \xi_1}, \\
\frac{\partial(\rho_a A)}{\partial t} + \frac{1}{h_1}\frac{\partial}{\partial \xi_1}\left(\rho_a A u_a\right) &= 0, \\
\frac{\partial \Omega}{\partial t} + \frac{1}{h_1}\frac{\partial}{\partial \xi_1}\left((\Omega + p_a A) u_a\right) &= -p\frac{\partial A}{\partial t} - u_a A F_a
\end{aligned} \tag{7}$$

in which subscripts "a" distinguish variables in air from ones in the basaltic fluid, and in which

$$\Omega = \rho_a A\left(\frac{1}{2}u_a^2 + e + g\,(z - z_0)\right) \tag{8}$$

is the energy density of air with internal energy e. Air is modeled as an ideal gas with $p_a = \kappa(s)\,\rho_a^\gamma = (\gamma - 1)\,\rho_a\, e$, where $\kappa = \kappa(s)$ is a function of entropy s and $\gamma = c_p/c_v = 1.4$ is the ratio of specific heats at constant pressure and volume, respectively. Viscous forces in air will be ignored, *i.e.*, $F_a = 0$.

The basaltic fluid and air are separated by an interface. In the one-dimensional

flow-tube model, the interface between the basaltic fluid on one side and air on the other is marked by a fluid parcel at position $\xi_1 = \xi_i(t)$. The dynamics of this parcel is governed by

$$\begin{aligned} \frac{d\xi_i}{dt} &= \frac{u_i}{h_1}, \\ \frac{du_i}{dt} &= -\frac{1}{\rho\, h_1}\frac{\partial p}{\partial \xi}\Big|_{\xi_1=\xi_i(t)} = -\frac{1}{\rho_a\, h_1}\frac{\partial p_a}{\partial \xi_1}\Big|_{\xi_1=\xi_i(t)}, \end{aligned} \tag{9}$$

where velocity $u_i(t) = u(\xi = \xi_i(t), t)$ and pressure are continuous across the interface, while density is generally not. Since we focus on high-speed inertial effects, diffusion of heat across the interface is neglected.

Except for the forcing, geometric and gravitational terms, the remaining hyperbolic part of the equations of motion is written in conservative form. The dynamics of shocks in basaltic flows is governed by momentum and mass conservation across discontinuities. Energy is additionally conserved in air because entropy is increasing across a shock in accordance with the second law of thermodynamics (Courant and Friedrichs, 1948). Conservative formulations of (3) and (7) form the basis of numerical discretizations in which shocks are properly simulated.

2.1. Scaling and numerics

Numerical integration is performed using dimensionless equations and numerical values are chosen to lie close to unity to improve numerical stability. By considering the equations of motion and state for the magma, the following scalings for various

parameters have been adopted:

$$p = P_0\, p', \quad \rho = \rho_0\, \rho', \quad t = (L_s/U)\, t', \quad \mathbf{x} = L_s\, \mathbf{x}',$$

$$\mathbf{u} = U\, \mathbf{u}', \quad n(p) = n_0\, n'(p'), \quad \sigma = \rho_0\, \sigma'$$

with $P_0 = n_0\, R_v\, T\, \rho_0$ and $U^2 = P_0/\rho_0$. A Froude number $F_l = P_0/(\rho_0\, g\, L_s)$, frictional number $\nu = \mu_0/(L_s\, \sqrt{P_0\, \rho_0})$, and volatile saturation $\epsilon = s\, {P_0}^{\beta}/n_0$ will appear in the scaled equations of motion and equations of state for basaltic magmas and air.

In order to solve the partial differential equations (3) and (7) for such high-speed flows, in which frictional effects will play a minor role, the major task is to discretize the inviscid, conservative and gravitational components of the dynamics properly. Shock-capturing second- and third-order Local Lax Friedrich and Essentially Non-Oscillatory numerical schemes have been used (Shu and Osher, 1988, 1989; Liu and Osher, 1998). Code validation and development have been reported in detail in Bokhove (1999). In particular, code validation consisted of comparing numerical solutions with exact inviscid solutions of stationary, moving and reflected shocks, and shock tubes for pure magma, pure air, and magma and air combined.

3. Basic processes in a dike-drift system

A typical cross section of the magma dike and repository drift (Fig. 1) is given in Fig. 2. The drift is assumed to be unfilled as a simplification, since the cross-sectional area of the canisters is small relative to the cross section of the drift. In the one-dimensional flow-tube model the connection between dike and drift is for simplicity

represented as an elliptical arc.

Characteristic scales in the configuration are: the length of the magma dike L_{dike}, the distance from the origin $\mathcal{O}$ to the end of the drift L_{drift}, arc length $L_{arc} = \frac{1}{2}\, d\, E(m)$ where $E(m)$ is the elliptical integral of the second kind with $m = (d^2 - w^2)/d^2$, and the overall length L_{end}. For dikes of greater extent, the pressure rises above the critical pressure $p_c = (n_0/s_H)^{1/\beta}$ for which all volatiles are dissolved in the magma. Therefore, we restrict dike lengths so that the pressure remains below this critical pressure p_c. Given the geometry, the effect of gravity in the flow-tube model is present in the dike, but diminishes in the arc and ultimately has no influence in the drift [cf., (3)].

The cross-sectional area $A(\xi_1, t)$ of the flow tube is constant in the drift and in most of the dike, A_{dike}, for $\xi_1 < L_{end} - L_{drift} - (1/2)\, L - d$ but narrows in a nozzle or transition zone with minimum area $A = \pi\, d\, w$ before it enlarges to the constant drift area, A_{drift}. The cross-sectional area in the nozzle is chosen to be the area of a circular strip of width w and diameter d, e.g. $\pi\, d\, w$, or a large fraction thereof and is thus smaller than the drift area. The characteristic spacing between drifts determines L. This smooth transition from the narrower nozzle area to the dike area over length $L/2$ has some similarity with the bulbuous dike-tip model solutions of Lister (1990), especially when we let the nozzle area grow from a very small value to its final one. This transition zone model is, however, a much simplified picture of the characteristic three-dimensional flow path and volume between dike and drift. In the flow-tube model, pressure, density and velocity depend only on one spatial coordinate. Most spatial

resolution for the numerical solution is needed in the region of prime interest around the dike-drift intersection and in the drift. Instead of ξ_1 a smooth coordinate transformation to another coordinate is made for which the numerical grid is regular. This yields a coarser ξ_1-grid near the bottom of the magma dike.

We consider a basaltic magma with at first the following reference parameter values $T = 1000\,K, \sigma = 2500\,kg\,m^{-3}, n_0 = 2\,wt\%, s_H = 3 \times 10^{-6}\,Pa^{-1/2}, R_v = 462\,J\,kg^{-1}\,K^{-1}$, and $\beta = 0.5$. Given the proposed dimensions of the repository site at Yucca mountain and its geology, we also use the following reference parameters: $L = 80\,m, w = 1\,m, d = 5\,m, A_{drift} \approx 19.64\,m^2, A_{dike} = 80\,m^2, L_{drift} = 200\,m, L_{dike} = 1075\,m$; friction $\mu_0 = 10\,kg\,m^{-1}\,s^{-1}$. Lithostatic pressure $p_L = \sigma\,g\,D = 10\,MPa$ at $D = 400\,m$ and overpressure $P_o = 10\,MPa$ give a total pressure at the dike tip of about $P_t = 20\,MPa$; and the critical void fraction at the point of fragmentation is taken as $\alpha = 70\%$.

The initial condition for the reference simulation is zero flow on either side of the drift entrance; there is a hydrostatic pressure gradient in the magma dike with a pressure $P_t = 20\,MPa$ at the dike tip, and air at atmospheric pressure and room temperature in the drift. The initial magma pressure and density can be determined by combining the equation of state and the hydrostatic balance condition in one relation, whose root can be found numerically to yield the pressure. Except for the hydrostatic balance, the initial condition is reminiscent of one for a classical shock tube.

We can roughly distinguish three phases in the fluid dynamical processes that occur

after the dike encounters the tunnel. These phases are found in the schematic Fig. 3 of the dike-tunnel system with its curvi-linear coordinate ξ_1 and varying cross section. After the magma breaks through into the tunnel, the first phase, 1, of the magma-air interaction begins. The initial flow closely resembles a classical, idealized shock tube problem in a uniform, horizontal pipe with magma on the left and air on the right. Three consecutive pressure profiles versus spatial coordinate ξ_1 of such an idealized shock tube problem in a horizontal pipe show that as time advances a rarefaction wave travels into the high pressure magma to the left, while a shock wave develops in the air which is displaced by the magma (Fig. 3). The magma-air interface lags behind this shock wave and is marked by the solid circles. This idealized solution follows from an exact implicit relationship between pressure, temperature and velocity, which is solved with a numerical root-finding routine. Since pressure and velocity are continuous across the magma-air interface, this interface is only revealed in density profiles. The influence of gravity will slightly modify the detailed propagation of the high-speed rarefaction wave in the vertical cross sections.

In the second phase, 2, the incoming shock wave in air reflects against the end wall of the tunnel. After a series of shock reflections back and forth between the drift end wall and the magma-air interface, a strongly amplified shock develops in the magma and travels back into the tunnel against the flow of incoming magma. The shock amplification against drift wall and magma-air interface is clearly visible in an idealized numerical calculation of phase 2 (Fig. 3). The initial pressure profile is shown versus

horizontal coordinate ξ_1, and consecutive profiles after regular time intervals are shown in the same graph but with the pressure on the ordinate increased by a constant value in each successive plot. The pressure at the open boundary of the domain, on the left, thus remains $0.2\,MPa$ during the time shown. Consecutive positions of the interface are indicated by circles in the profiles.

In the final phase, 3, the rarefaction wave and reflected shock waves travel down into the dike and diminish in magnitude due to hydrostatic and frictional effects. Two pressure profiles at different times show how the rarefaction wave and shock waves propagate to the left (Fig. 3; the dashed pressure profile is at a later time than the solid one). A second shock wave has arisen because part of the large reflected shock in magma is transmitted through the nozzle or dike-drift intersection and part of it is reflected back into the drift. After these waves leave the domain, the flow comes to rest and in hydrostatic balance, while the air in the drift is compressed to a small volume against the drift end wall.

To gain understanding of the shock reflection phase, two idealized problems will be considered next. We initially deal with a horizontal shock tube, and neglect the effects of gravity and the geometry of the dike-tunnel system. Subsequently we examine how the simple results are modified in a reference simulation. Note that the initial conditions in these idealized simulations and the reference simulation are *different.* We have summarized various simulations in Table 1.

3.1. Idealized solutions of rarefaction and shock waves

It is difficult to derive a solution for the reflection of the shock wave arising from a shock tube because of the evolving rarefaction wave. In order to help develop insight, we separately examine an idealized shock tube problem and the reflection of a shock in air between the tunnel wall and the magma-air interface. For the reflecting shock problem, we assume constant but generally different values of pressure, phase or density on either side of shocks and interfaces in order to understand the magma and air interaction. Parameter values for the magma are taken to equal those we used earlier but the initial conditions are varied.

3.1.1. A shock tube in magma and air. The space-time diagram in Fig. 4 explains the dynamics in a horizontal shock tube, in which gravity and viscosity are ignored. Initially, there is zero flow with the dense, high-pressure magma to the left and the air at atmospheric pressure and room temperature to the right of the interface. Subsequent flow consists of a high-speed shock in air, which propagates to the right ahead of the interface and a backward propagating rarefaction wave. The exact shock relations in air are well known (e.g. Courant and Friedrichs, 1948; Whitham, 1974) and can be connected to the exact expressions for the rarefaction wave in magma by requiring continuity of velocity and pressure across the magma-air interface. In Fig. 5a), we graph the speed of the shock in air versus the initial magmatic overpressure P_t in the dike (solid line). In Fig. 5b), we graph the time required for this shock wave to reach the end of a $200\,m$ tunnel, say, versus the overpressure (solid line). The shock speed

in air is high, of the order of $400 - 600\,m/s$, and reaches the end of a $200\,m$ tunnel, say, in a fraction of a second. The speed of the incoming mixture of ash, gas, and air behind the shock wave is lower, since the shock wave propagates quickly ahead of the interface. This speed of the interface between magma and air is given by the thin solid line (Fig. 5a). An increased overpressure is seen to lead to higher sound speeds and shorter traveling times. In addition to the magma-air shock tube problem, we have also graphed the shock speeds and travel time for shock tube problems with only a magmatic fluid and only air, respectively. In these latter two cases, the high-pressured magma or air lies to the left, while the magmatic or air pressure to the right is atmospheric. These solutions for pure magma and pure air are denoted by dashed and dotted lines respectively (Fig. 5). With a thin dashed-dotted line we also indicate the speed of the internal interface in pure air. (There is no such interface in a pure magmatic fluid since density/pressure and velocity govern the dynamics as the magma is isothermal). By comparing the solutions for pure magma, pure air, and magma and air side by side, we note that shock and interface speeds are highest in pure air and lowest in pure magma, with speeds in the magma and air combination lying between these two pure cases. More importantly, we note that the shock speeds do not drop significantly till $P_t < 2\,MPa$. In other words, even for dike-tip pressures significantly lower than the $10\,MPa$ lithostatic pressure, shocks remain high speed. All the above solutions of shock tube problems are solved by standard methods (e.g. Whitham, 1974; details are found in Bokhove, 1999) and are exact except for the use of a root-finding routine.

3.1.2. Shock reflection in magma and air. When states with constant values are considered in the absence of gravity and explicit friction and in a horizontal pipe mimicking the end of the drift, the dynamics may be solved exactly by tracking each shock-shock, shock-wall and shock-interface interaction in space and time. The space-time diagram in Fig. 6 explains the evolution.

Initially, dense, high-pressure basaltic fluid in constant state 1 flows towards the wall; across the interface in state 2, velocity and pressure in air are continuous and hence the same as in state 1, but the air density has a lower value; the shock in air approaches the wall faster then the interface, and demarcates air state 2 from air state 3 while the latter is at rest. At time T_0, the shock in air reflects and amplifies against the wall, the reflected shock reflects against the interface at time T_1, which has moved forward, and so on. The interface-shock reflection yields a reflected shock and a transmitted shock, the latter propagates in the basaltic fluid. The first couple of transmitted shocks are still swept towards the wall because they are unable to overcome the incoming fluid speed u_1 in state 1. They are, however, traveling away from the interface. Several transmitted shocks are thus generated in the basaltic fluid, the later ones overtake and annihilate the earlier ones and increase the speed of the shock front in the magma. Finally, a strongly-amplified shock in the magma propagates away from the wall with great speed. When a shock generated by an explosion under water meets the air-water interface, a shock in air is formed while a rarefaction wave propagates back into the water. Contrary to this air-water interaction, the initial shock in the tunnel propagates in the

less dense medium: air. Each shock-interface interaction yields two shocks till finally the air has become dense enough to support the propagation of the shock through the interface while a rarefaction wave unfolds in the air (see, *e.g.*, Courant and Friedrichs 1948). Numerical evidence in Fig. 7 supports this picture (Fig. 6). Velocity, density and pressure profiles have been plotted after regular time intervals. Each consecutive profile has been shifted upward by a constant amount, relative to the previous profile. That is, zero pressure for the initial pressure profile corresponds to the zero on the vertical scale, zero pressure for the first profile corresponds to $\Delta P \approx 0.2\,MPa$ on the vertical scale, etcetera, and similarly for velocity and density. (The pressure profiles of phase 2 in Fig. 3 show an enlargement of the first few shock reflections of this simulation.) In particular, pressure and velocity profiles indicate the nature of the pressure rise in the air pocket following each shock reflection, while the movement of the interface emerges in the density profile. Note that the density jump of the shock in air has a comparatively small magnitude. In contrast, a large jump across the transmitted shock in magma appears after the first shock-interface encounter in profile 6. The shock reflection in Fig. 6 can in principle be solved exactly, although it requires root finding routines; further study of this resonating phenomenon is in progress.

As the magma-air interface slows down and finally arrests, the dynamics are no longer presented accurately by the flow-tube model. Upon slowing down, a gravity current must form, fill the floor at the end of the drift, and leave an air pocket in the upper corner, which will slowly start to spread across the ceiling of the drift. The

dynamics of gravity currents is not captured by the flow-tube model. Wave speeds of these gravity currents, of the order of $\sqrt{g'd} \sim 7\,m/s$ (with reduced gravity g'), are much slower than those of the acoustic shock and rarefaction waves.

3.2. Reference simulations

In the following, a comprehensive account is given of a simulation with parameter values typical for relatively volatile-rich basaltic magmas. Remaining simulations that comprise the parameter study are straightforward variations on the basic geometry and parameter set used therein. The initial conditions and geometries of simulations presented onwards *differ* from those presented previously. Nearly all simulations have been verified against double resolution runs. (Simulation numbers refer to tables in Bokhove (1999).)

Flow profiles during the first $2.734\,s$ of the reference simulation are shown in Fig. 8. The typical shock-tube profiles of phase 1 (see Fig. 3) govern the dynamics till the shock in air reflects against the end wall of the tunnel at $\xi_1 = 1277.6\,m$. A rarefaction wave is then seen to travel into the magma dike but it is modified by the presence of gravity in the arc and dike, and at later times by the high viscosity below the fragmentation level. Both pressure and velocity profiles reveal the interplay of shock reflection between drift wall and magma-air interface during the first tens of seconds. The density profile again strikingly marks the movement of the interface and the transmitted shock. Near the end of the simulation at $t = 1.139\,s$ a strongly amplified shock wave is seen to emerge in the

magma. The pressure drop across the shock is about 50 times larger than the magnitude of the initial shock wave in air. Inertial flows dominate during the first couple of seconds and friction and fragmentation are of minor importance. (Remember that fragmentation is solely modeled through the dependence of the frictional parameterization on void fraction.) The shock reflection process in the references simulation therefore closely resembles the inviscid shock reflection process descibed before (compare Fig. 8 with Fig. 7 qualitatively since the precise geometry and initial conditions differ).

When the simulation is followed for $11.39\,s$ several new phenomena become apparent (Fig. 9). The remarkable shock amplification considered in the previous simulation occurs within the first second (the first three frames summarize phase 2). While the rarefaction wave travels back into the dike, the large reflected shock wave propagates away from the wall. Before it reaches the dike-drift transition (at $\xi_1 = 1077.6\,m$) it encounters an anomaly associated with a choking or hydraulic condition at the nozzle (Courant and Friedrichs 1948). This phenomenon appears already earlier in the simulation (Fig. 8). After the first reflected shock has reached the transition, a transmitted shock of diminished size travels into the dike, dissipates and leaves the domain, while a reflected shock travels back into the drift, reflects against the wall, and after it reaches the transition its transmitted and reflected component decay (phase 3). Final profiles are close to hydrostatic equilibrium with a strongly compressed air pocket at the end of the tunnel.

4. Parameter study

The flow-tube model is one of the simplest models of transient volcanic flows. Since the model depends only on time and one spatial dimension, we investigate the dynamics for a large range of parameter values. The sensitivity to parameter changes is shown in two graphs: one graph shows the maximum pressure P_{max} observed in the tunnel, and the other graph shows the shock amplification S versus the parameter range. The shock amplification S is defined as the pressure drop across the reflected shock in magma over the initial pressure drop across the shock in air. (When the maximum pressure coincides, numerically, with the initial pressure at the dike-tunnel intersection, we use the maximum pressure in the part of the tunnel that lies at least $50\,m$ away from this intersection.)

We begin to consider changes in initial dike-tip pressure P_t at the start of the simulation, while leaving all other parameters fixed. The maximum pressure observed in the tunnel lies between $10\,MPa$ to $33\,MPa$ (Fig. 10a) and the shock amplification lies between 28 and 51 (Fig. 10b) for dike-tip pressures in the range $12 - 25\,MPa$. For increasing values of P_t, we find that the rarefaction wave is larger and propagates faster into the dike; yet the position of the magma-air interface hardly changes in the investigated range of dike-tip pressures (not shown).

A change in volatile content, while leaving other parameters fixed, can yield pressures in the magma dike beyond the critical pressure p_c above which all volatiles are dissolved and the magma is incompressible. To avoid these incompressible regions, the

dike-tip pressure and the length of the magma dike have been reduced so that the fluid in the computational domain remains compressible, with pressures below p_c. Maximum pressure in the tunnel and the shock amplification are found in Fig. 11 (note that dike-tip pressure has changed for the lower volatile content simulations). By interpreting Fig. 10 and Fig. 11 in tandem, we see that an increase in volatile content leads to a small decrease of the maximum pressure in the tunnel and the shock amplification, for fixed dike-tip pressure.

The effect of variation in the frictional parameter μ_0 (4) is next explored while leaving other parameters untouched. Increased friction is seen to lead to reduced pressures in the drift and reduced pressure drops for reflected magma shocks; the maximum pressure and the shock amplification are reduced by $10\,MPa$ and 50 %, respectively, when the frictional coefficient μ_0 in (4) increases an order of magnitude (Fig. 12). The reflected shock in magma is also less pronounced, while the magnitude of the shock in air remains similar, because friction in air is negligible. Increased friction drastically slows down the speed of the rarefaction wave in the magma dike (not shown).

When the void fraction (5) of the magma remains small, the mixture resembles a high-viscosity foam and viscosity is parameterized by (4). Above the fragmentation level with critical void fraction α, the lower viscosity is parameterized by the turbulent flow law (6). Therefore, the explosive behavior of magma and air is expected to reduce as a function of increasing fragmentation level α. Three simulations with $\alpha = 70\%$, 80% and 90 %, respectively, reveal a sharp decrease of about 70 % in the reflected shock wave

amplitude (Fig. 13b), while the maximum pressure observed in the tunnel (Fig. 13a) and the rarefaction wave remain the same. An increase of the fragmentation level corresponds to more high viscosity magma and less gaseous low-viscosity turbulent magma. The rarefaction wave propagates in all cases in the highly viscous magma dike and is unchanged, but the higher viscosity for magma in the drift implies lower and slower shock magnitude and speed. The increased viscosity essentially leads to a reduced volume flux, which then yields lower shock amplification of the reflected shock in magma. At later stages the reflected shocks in the simulations with fragmentation levels at 80 % and 90 % undergo further amplification due to an interaction with the hydraulic wave behind the constriction. To contrast the dynamic process with the reference simulation, the time evolution is shown in Fig. 14 for $\alpha = 80\,\%$.

4.1. Alternate dike-drift transitions

The quiescent initial condition used so far mimicks the sudden breakthrough of magma from dike into drift, and the fixed nozzle geometry is meant to represent a typical flow path. Viable alternatives exist that are arguably equally realistic. The connection between dike and drift could, for example, open gradually. A dike-drift nozzle is therefore considered that opens during one second or ten seconds from $0.79\,m^2$ to $\pi\,d\,w = 15.7\,m^2$ after the simulation begins. Thereafter, the cross-sectional area remains constant. (A small but finite initial opening is used to avoid numerical instabilities.) The remaining set-up, including the initial condition, is the same as in the

basis simulation in Fig. 9. When the nozzle is opened in one second, simulation (not shown) indicates that the dynamics slows down, the shock amplification takes about $3\,s$ instead of $1\,s$, but the final reflected shock wave in magma maintains its amplitude. The mass flow is seen to increase more gradually, but the transmitted magmatic shock near the interface still has to acquire enough speed and pressure head, by going through several reflection cycles between end wall and magma-air interface, before it is able to propagate away from the wall.

The situation is altered when the nozzle is opened during ten seconds instead of one second. Shock amplification is still significant, but the pressure at the end of the tunnel is after $3.42\,s$ only a third of the previously observed pressures. In addition, the amplification process is slower. As we follow this simulation to $11.39\,s$, we observe that the final pressure in the tunnel comes close to the initial dike-tip pressure (Fig. 15). We conclude, therefore, that when the nozzle opens within ten seconds the pressure will still rise to values close to the initial dike-tip pressure. However, the initial pressure anomaly at the end of the tunnel is more localized when the nozzle opens faster.

Alternative nozzle geometries at the dike-drift transition can speed up the dynamics. In the following variation on the reference simulation cross-sectional area $A(\xi)$ is fixed in time but varies linearly in ξ_1 from dike to drift. The basic results remain similar, but both shock and rarefaction wave propagate slightly faster. Since the nozzle is expanding from drift to dike, the reflected shock wave in magma is mainly transmitted at the nozzle, in contrast to the situation for the simulation in Fig. 9 where an approaching

shock is partly transmitted and partly reflected because the cross-sectional area first contracts at the nozzle before it expands to its large dike value.

The dynamics in a dike that closes up as magma is withdrawn can be modeled crudely by releasing magma into the drift from a dike with only a short and finite depth. We consider therefore the ensuing magma flows after breakthrough for four representative, finite dikes of depth $50, 100, 300$ and $500\,m$, respectively. In all these four cases, the maximum pressure observed in the drift remains above $12\,MPa$ due to the shock amplification process (Fig. 17), even though the final pressure in the system after about $11\,s$ is lower (between $7\,MPa$ and $17\,MPa$). The shock amplification lies again around 50 for dike depths beyond $100\,m$ (Fig. 17).

5. Discussion and conclusions

In this paper, we have analyzed a flow-tube model of magma-air interactions in an idealized dike-drift geometry (Fig. 1). A dike of constant width and characteristic length is smoothly connected with a uniform and cylindrical horizontal drift. Although in practice the tip of the dike would slowly ascend from a magma chamber and advance towards the subsurface repository drifts, the magma-air interactions studied here start from rest after a diaphragm between dike and drift is broken (Fig. 1b,c).

In accordance with the idealized shock tube, hydraulic control, shock-interface and shock-wall reflection problems, the simulations presented herein show a rarefaction wave traveling into the magma dike, and a complex interaction of rarefaction and reflected

shock wave interactions in the drift. The initial shock wave in the compressed air travels to the end of the tunnel with speeds of order $500\,m/s$. Strong shock amplification, between 15 and 50 times, results as a consequence of a "resonating" process of the initial shock in air between the magma-air interface and the drift end wall. That resonance process is consistent with analysis and simulations in idealized interface-wall shock reflection problems (section 3.1). Typically, the resonating process in a drift with an end wall $200\,m$ from the dike-drift intersection is finished in about one to two seconds. After about ten seconds, the reflected shock wave in magma has propagated about $1000\,m$ into the dike.

The sensitivity of our reference simulation, presented in section 3.2, has been assessed as a function of the initial dike-tip pressure, the volatile content of the magma, friction, fragmentation level, and nozzle geometry (section 4). This nozzle geometry with varying cross-sectional area mimicks the characteristic flow-tube area around the dike-drift transition, where the upward going magma turns around and flows into the horizontal tunnel. In accordance with our expectations, increasing the dike-tip pressure leads to larger rarefaction and amplified reflected shock waves. While the rarefaction and shock waves propagate somewhat faster, the interface movement is basically independent of changes in dike-tip pressure in the range $12 - 25\,MPa$. Changes of volatile content are hard to implement without modifying dike-tip pressure or without a model that can handle both (nearly) incompressible and compressible magma together (see point (ii) below). Along with the volatile content, dike-tip pressure and dike depth

have also been changed in order to avoid incompressible regions, in which all volatiles are dissolved, in our computational domain. But we could discern that for decreasing volatile content, from $2.5\,wt\%$ to $1\,wt\%$, the amplitude of the shock wave diminishes. An increase of frictional parameter μ_0, which appears in the frictional parameterization for magma below fragmentation, shows a slow-down in the speed of the rarefaction wave in the dike and a reduction in amplitude of the reflected shock. It is about 50 % for a tenfold increase of μ_0. A similar reduction of about $30 - 70\,\%$ in the reflected shock amplitude occurs upon increasing the fragmentation level from 70 % through $80 - 90\,\%$. Rarefaction wave propagation speeds and interface positions are not affected by this change because a changing fragmentation level only affects low-pressure regions of magma with a large amount of exsolved volatiles (*i.e.*, in the drift). We have also considered flow in linear and gradually opening nozzle geometries. These variations in nozzle geometry only changed details of the ensuing transient flows but did not eliminate the shock-amplification process (section 4.1). Finally, to crudely model the effect of a closing dike when magma is withdrawn we considered cases in which the dike depth is finite. The maximum pressures observed in the tunnel after breakthrough remain high, above $12\,MPa$ for a $50\,m$ deep finite dike and this rapidly increases for deeper dikes of finite length.

In conclusion, our bounding calculations show that by reducing volatile content and dike-tip pressure to reasonable lower limits and by increasing frictional values and fragmentation levels to reasonable upper limits, the shock-wave amplitudes are

diminished but by no means eliminated. The large increase of pressure at the end of the drift during the initial seconds is therefore argued to be a generic feature of magma-air interactions. A rule of thumb for open dikes is that the final pressure of the reflected shock wave in magma is of the order of the initial dike-tip pressure. In natural or artificial dike-tunnel systems a new dike or conduit may develop if absolute pressures and pressure gradients are sufficiently high, and our study suggests that a breakthrough is likely to arise first at the end of the tunnel away from the dike. That is, figures Fig. 8 and Fig. 9 show that the high final pressure in the drift builds up from the end of the drift through the development of a high-pressure reflected shock wave. Whether breakthrough will occur at the end of a tunnel depends on the complicated and poorly understood interaction between fluid dynamics of the magma-air system and the rock mechanics associated with the dike. Further work is required to study this interaction.

Although the model and parameter study described herein grasp the leading-order behavior of explosive magma-air interactions in a dike-drift system, details of several phenomena remain poorly understood. A number of important aspects which merit further research include: (i) coupling the flow-tube model to a simplified model of rock mechanics (*e.g.* Lister, 1990; Rubin, 1993; Mériaux *et al.*, 1999) in order to assess how the dike walls react to pressure fluctuations in the magma after breakthrough; (ii) the stationary and transient flow in a one-dimensional flow-tube model in which a magma dike or conduit has formed at the end of a drift and has reached the Earth's surface; (iii) better characterization of the viscosity and bulk rheology of the magma-gas mixture,

of the volatile exsolution rate and kinetics, and of the effects of phase separation; (iv) the formulation of a gravity-current model for the interface at the end of the drift, combined with the compressible magma and air flow; and finally (v) more refined modeling of transient flow profiles at the dike-drift transition and of gravity currents near the interface in two-dimensional laterally averaged or three-dimensional models.

Acknowledgments. The numerical work benefited significantly from research of the author performed under an EC MAST-III Surf and Swash Zone Mechanics grant (SASME MAS3-CT97-0081) received by Professor D.H. Peregrine, who also kindly suggested involvement of the author in the project and shared his knowledge in several discussions. This manuscript is the result of work that the author performed for and in part at the Center for Nuclear Waste Regulatory Analyses (CNWRA) for the U.S. Nuclear Regulatory Commission (NRC), through a contract of the CNWRA with the University of Bristol. The presented results are an independent product of the University of Bristol, and they do not necessarily reflect the views or regulatory position of the NRC. OB wishes to thank the staff of the CNWRA for all their support during his stay. Finally, it is a pleasure to thank Dr. C. Connor for all discussions, the presence of a good computational platform at CNWRA and his scientific input.

References

Batchelor, G.K., *Fluid Dynamics.* Cambridge University Press, 615 pp, 1988.

Bokhove, O., Numerical modeling of magma-repository interactions. School of Mathematics/South West Research Institute Internal report, 106 pp. Available upon request from the author and from: South West Research Institute, Center for Nuclear Waste Regulatory Analyses, 6220 Culebra Rd. San Antonio, TX, 78238-5166, U.S.A., 1999.

Condit, C.D., & Connor, C.B., Recurrence rates of volcanism in basaltic volcanic fields: An example from the Springerville volcanic field, Arizona, *Geol. Soc. Am. Bull. 108,* 1225-1241, 1996.

Connor, C.B., & Hill, B.E., Estimating the probability of volcanic disruption at the Yucca Mountain site using nonhomogeneous Poisson models. La Grange Park, Illinois, American Nuclear Society, Focs '93, p. 174-181, 1993.

Connor, C.B., & Hill, B.E., Three nonhomogeneous Poisson models for the probability of basaltic volcanism: Application to the Yucca Mountain region, Nevada, USA. *J. Geophys. Res., 100,* 10107-10126, 1995.

Connor, C.B., Lane-Magsino, S., Stamatakos, J.A., Martin, R.H., LaFemina, P.C., Hill, B.E., & Lieber, S., Magnetic surveys help reassess volcanic hazards at Yucca Mountain, Nevada. *Eos, Transactions, American Geophysical Union. 7,* 73-78, 1997.

Connor, C.B., J.A. Stamatakos, D.A. Ferrill, B.E. Hill, G.I. Ofoegbu, F.M. Conway, B. Sagar, & J.S. Trapp, Geologic factors controlling patterns of small-volume basaltic volcanism: Application to a volcanic hazards assessment at Yucca Mountain, Nevada. *J. Geophys. Res.*, in press., 1999.

Courant, R. & Friedrichs, K., *Supersonic Flow and Shock Waves.* Wiley-InterScience, New York, 1948.

Heizler, M. T., F.V., Perry, B. M. Crowe, L. Peters, & R. Appelt, The age of Lathrop Wells volcano center: An 40Ar/39Ar dating investigation. *J. Geophys. Res., 104,* 767-804, 1999.

Jaupart, C., & Allègre, C., Gas content, eruption rate and instabilities of eruption regime in silicic volcanoes. *Earth Planet. Sci. Lett. 102,* 413-429, 1991.

Larsen, G., Grönvold, K., & Thorarinsson, S., Volcanic eruption through a geothermal borehole at Námafjall, Iceland. *Nature. 278,* 707-710, 1979.

Mériaux, C., Lister, J.R., Lyakhovsky, V., & Agnon, A., Dyke propagation with distributed damage of the host rock. *Earth Planet. Sci. Lett. 165,* 177-185, 1999.

Lister, J.R., Buoyancy-driven fluid fracture - the effects of material toughness and of low-viscosity precursors. J. Fluid Mech.*210,* 263-280, 1990.

Lister, J.R. & Kerr, R.C., Fluid-mechanical models of crack-propagation and their application to magma transport in dykes. *J. Geophys. Res.,* 96, 10049-10077, 1991.

Liu, X.-L. & Osher, S., Convex ENO high order multi-dimensional schemes without field by field decomposition on staggered grids. J. Comp. Phys.*142,* 304-330, 1998.

Mériaux, C., Lister, J.R., Lyakhovsky, V., & Agnon, A., Dyke propagation with distributed damage of the host rock. *Earth Planet. Sci. Lett. 165,* 177-185, 1999

Rubin, A.M., Dikes versus diapirs in viscoelastic rock. *Earth Planet. Sci. Lett. 119,* 641-659, 1993.

Rubin, A.M., Propagation of magma-filled cracks. *Ann. Rev. Earth Planet. Sci.* 23, 287-336,

1995.

Shu, C-W. & Osher, S., Efficient implementation of essentially non-oscillatory shock-capturing schemes. J. Comp. Phys. *7,* 439-471, 1988.

Shu, C-W. & Osher, S., Efficient implementation of essentially non-oscillatory shock-capturing schemes II. J. Comp. Phys. *83,* 32-78, 1989.

Sparks, R.S.J., The dynamics of bubble formation and growth in magma. *J. Volcanol. Geotherm. Res. 29,* 99-124, 1978.

Vergnoille, S., & Jaupart, C., Separated two-phase flow and basaltic eruptions. *J. Geophys. Res., 91,* 12842-12860, 1986.

Whitham, G.B., *Linear and nonlinear waves.* John Wiley, Toronto, 636 pp, 1974.

Wilson, L., Sparks, R.S.J., Huang, C.T., & Watkins, N.D., Explosive volcanic eruptions, IV, The control of magma properties and conduit geometry on eruption column behaviour. *Geophys. J. R. Astron. Soc. 63,* 117-148, 1980.

Wilson, L., & Head, J.W., Ascent and eruption of basaltic magma on Earth and moon. *J. Geophys. Res. 86,* 2171-3001, 1981.

Woods, A.W., The dynamics of explosive volcanic eruptions. *Reviews of Geophysics. 33,* 495-530, 1995.

---

[1]Present affiliation and corresponding author: Dr. O. Bokhove, Faculty of Mathematical Sciences, P.O. Box 217, 7500 AE, Enschede, The Netherlands.

[2]Present address: BP Institute for Multiphase Flow, Madingley Rise, Madingley Road, University of Cambridge, CB3 OEZ, Cambridge, U.K.

Figure 1. (a) A magma dike is moving upward towards a drift filled with air at atmospheric pressure. (b) In an idealized and experimental configuration of a volcanic dike and drift, a vertical diaphragm (depicted at the beginning of the drift as a dashed light area) separates magma from air. When the membrane is removed, the dike-drift interaction begins. (c) The flow in a flow-tube model depends only on a smooth coordinate ξ_1 which follows the dike, and turns via an arc into the drift; variations of the cross-sectional area $A(\xi_1, t)$ of dike, connecting arc, and drift are captured in the model.

Figure 2. A vertical cross section of the magma dike and drift system defines the various length scales involved in typical flow-tube model simulations.

Table 1. Summary of presented simulations in space-time plots.

Simulation	Figure	Configuration	Friction	Dike tip P_t (MPa)	Duration (s)
Resonance	7	horizontal drift	none	-	0.78
Reference 1a	8	dike & drift	$\mu_0 = 10\,kg\,m^{-1}\,s^{-1}, \alpha = 70\,\%$	20	1.139
Reference 1b	9	dike & drift	$\mu_0 = 10\,kg\,m^{-1}\,s^{-1}, \alpha = 70\,\%$	20	11.39
Fragmentation	14	dike & drift	$\mu_0 = 10\,kg\,m^{-1}\,s^{-1}, \alpha = 80\,\%$	20	11.39
Time-varying $A(\xi, t)$	15	dike & drift	$\mu_0 = 10\,kg\,m^{-1}\,s^{-1}, \alpha = 70\,\%$	20	11.39
Linear $A(\xi)$	16	dike & drift	$\mu_0 = 10\,kg\,m^{-1}\,s^{-1}, \alpha = 70\,\%$	20	11.39

Figure 3. This schematic explains the three characteristic phases in time during magma-air interactions in a dike-drift system. Scales are exaggerated. The initial conditions in each of the depicted phases are different for explanatory reasons.

Figure 4. Four regions appear in this space-time sketch for a shock tube in magma, left of the interface, and air, right of the interface.

Figure 5. (a) The speed of the shock in air versus the dike-tip pressure P_t for the separated magma-air fluid (solid line), the shock in magma for a pure magma fluid (dashed line), and the shock in air only (dashed-dotted line). Speeds of pure air and magma-air fluid interfaces are thin versions of their shock lines. (b) The travel time of these shocks in a $200\,m$ long tunnel as function of P_t for the magma-air, magma and air fluids, respectively.

Figure 6. Space-time diagram of the resonating shock interactions between tunnel wall and melt-air interface. The vertical axis is space, the horizontal axis is time. The interface is denoted by a dashed-dotted line and shocks by a solid line. Initially, there is an upper state 1 in the basaltic fluid, an intermediate air state 2, and a lower air state 3 at rest. Effects of gravity and friction are absent.

Figure 7. Velocity, density and pressure profiles are shown for a simulation of resonating shocks between a tunnel wall and magma-air interface. The wall of the horizontal tunnel is at the right; the boundary on the left is open and allows inflow (and outflow). The crosses at the ξ_1-axis indicate the forward moving positions of the interface, except in the pressure plot where circles indicate positions and pressures of the magma-air interface. Initial values of the pressures are $p_1 = 0.2\,MPa$ (magma), $p_2 = 0.2\,MPa$ (air) and $p_3 = 0.1\,MPa$ (air). Observe the strong compression of air as time advances. Gravity plays no role.

Figure 8. Velocity, density and pressure profiles are shown for magma-air interactions in a dike-drift system. The simulation encompasses $2.734\,s$ and each of the 41 profiles is spaced $0.0683\,s$ apart. Profiles in dike and drift have been separated to emphasize the different scaling in dike and drift. The rarefaction wave in the dike, the magma-air interface (only visible in the density profile), and the shock wave have been indicated. Short-time reference simulation 101.

Figure 8. Continued, short-time reference simulation 101.

Figure 9. Velocity, density and pressure profiles are shown for magma-air interactions in a dike-drift system during $11.39\,s$ in 26 profiles each spaced $0.456\,s$ apart. Long-time reference simulation 060.

Figure 9. Continued, long-time reference simulation 060.

Figure 10. (a) Maximum pressure P_{max} in the tunnel, and (b) shock amplification S, the ratio of the maximum pressure drop in magma over the initial pressure drop across the shock in air, are shown versus initial dike-tip pressure P_t for runs: 064) $P_t = 25\,MPa$, 060) $P_t = 20\,MPa$, 061) $P_t = 15\,MPa$, and 063) $P_t = 12\,MPa$.

Figure 11. (a) Maximum pressure P_{max} in the tunnel, and (b) shock amplification S are shown versus volatile content n_0 for simulations (with different initial dike-tip pressures): 067) $n_0 = 2.5\,wt\%, P_t = 20\,MPa$; 060) $n_0 = 2\,wt\%, P_t = 20\,MPa$; 065) $n_0 = 1.5\,wt\%, P_t = 15\,MPa$; and 071) $n_0 = 1\,wt\%, P_t = 10\,MPa$.

Figure 12. (a) Maximum pressure P_{max} in the tunnel, and (b) shock amplification S are shown versus frictional coefficient μ_0 for runs: 060) $\mu_0 = 10\,kg\,m^{-1}\,s^{-1}$, 072) $\mu_0 = 50\,kg\,m^{-1}\,s^{-1}$, and 073) $\mu_0 = 100\,kg\,m^{-1}\,s^{-1}$.

Figure 13. (a) Maximum pressure P_{max} in the tunnel, and (b) shock amplification S are shown versus fragmentation level α for runs: 060) $\alpha = 70\%$, 082) $\alpha = 80\%$, and 084) $\alpha = 90\%$.

Figure 14. Pressure profiles are shown for magma-air interactions in a dike-drift system for simulation 082 with a fragmentation level of 80% instead of the 70% in the reference simulation. The scaling of the axes and time intervals is identical to the one in Fig. 9.

Figure 15. Pressure profiles are shown for magma-air interactions in a dike-drift system with a time-dependent cross section $A(\xi, t)$ increasing to its "reference" value during ten seconds. Simulation 097 ends at $t = 11.39\,s$. 26 profiles are shown separated $0.456\,s$ apart.

Figure 16. Pressure profiles are shown for magma-air interactions in a dike-drift system with a cross section A varying linearly between A_{dike} and A_{drift} in the transition zone. The scaling of axes and the offset in stack plots is the same as in Fig. 9. Simulation 076.

Figure 17. (a) Maximum pressure P_{max} in the tunnel, and (b) shock amplification S are shown versus the length of a finite-depth dike for runs: 102) $50\,m$, 103) $100\,m$, 104) $300\,m$, and 105) $500\,m$.

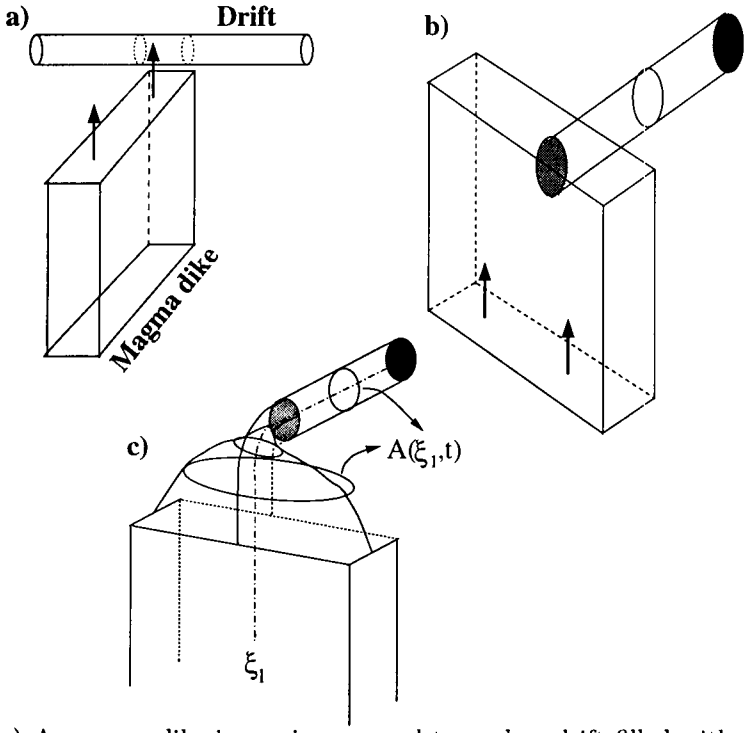


Figure 1: (a) A magma dike is moving upward towards a drift filled with air at atmospheric pressure. (b) In an idealized and experimental configuration of a volcanic dike and drift, a vertical diaphragm (depicted at the beginning of the drift as a dashed light area) separates magma from air. When the membrane is removed, the dike-drift interaction begins. (c) The flow in a flow-tube model depends only on a smooth coordinate ξ_1 which follows the dike, and turns via an arc into the drift; variations of the cross-sectional area $A(\xi_1, t)$ of dike, connecting arc, and drift are captured in the model.

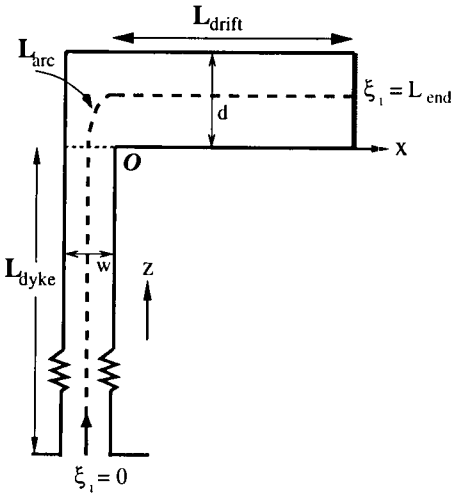


Figure 2: A vertical cross section of the magma dike and drift system defines the various length scales involved in typical flow-tube model simulations.

Simulation	Figure	Configuration	Friction	Dike tip P_t (MPa)	Duration (s)
Resonance	7	horizontal drift	none	-	0.78
Reference 1a	8	dike & drift	$\mu_0 = 10\,kg\,m^{-1}\,s^{-1}, \alpha = 70\,\%$	20	1.139
Reference 1b	9	dike & drift	$\mu_0 = 10\,kg\,m^{-1}\,s^{-1}, \alpha = 70\,\%$	20	11.39
Fragmentation	14	dike & drift	$\mu_0 = 10\,kg\,m^{-1}\,s^{-1}, \alpha = 80\,\%$	20	11.39
Time-varying $A(\xi, t)$	15	dike & drift	$\mu_0 = 10\,kg\,m^{-1}\,s^{-1}, \alpha = 70\,\%$	20	11.39
Linear $A(\xi)$	16	dike & drift	$\mu_0 = 10\,kg\,m^{-1}\,s^{-1}, \alpha = 70\,\%$	20	11.39

Table 1: Summary of presented simulations in space-time plots.

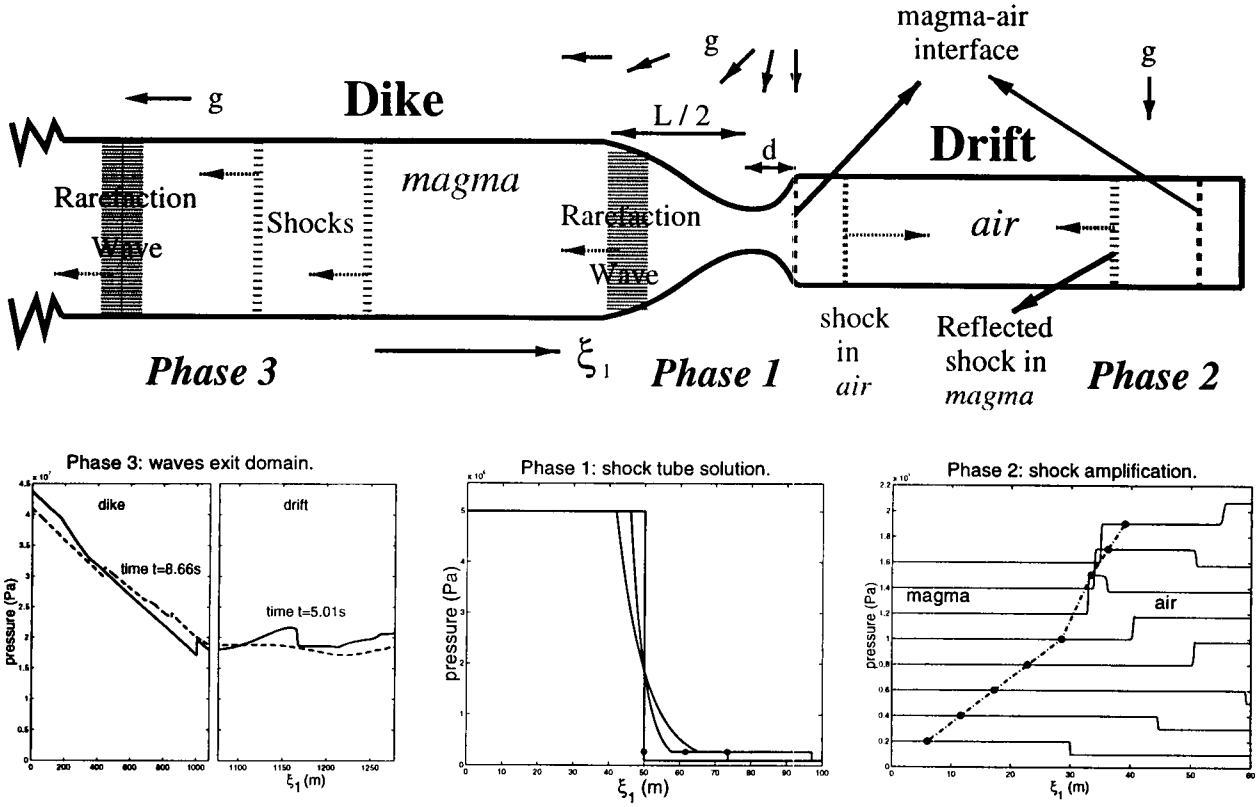


Figure 3: This schematic explains the three characteristic phases in time during magma-air interactions in a dike-drift system. Scales are exaggerated. The initial conditions in each of the depicted phases are different for explanatory reasons.

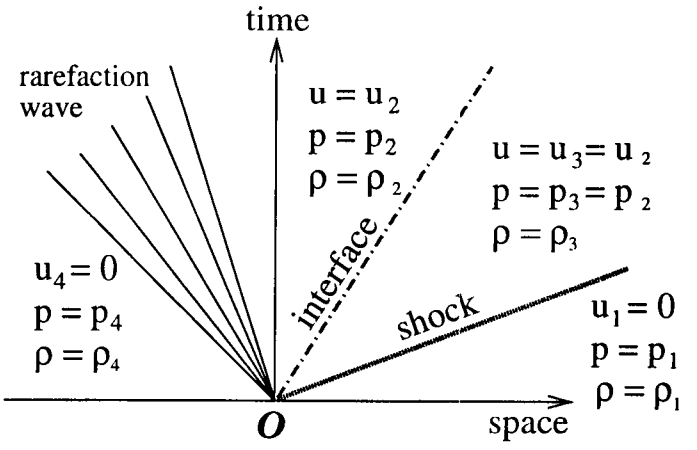


Figure 4: Four regions appear in this space-time sketch for a shock tube in magma, left of the interface, and air, right of the interface.

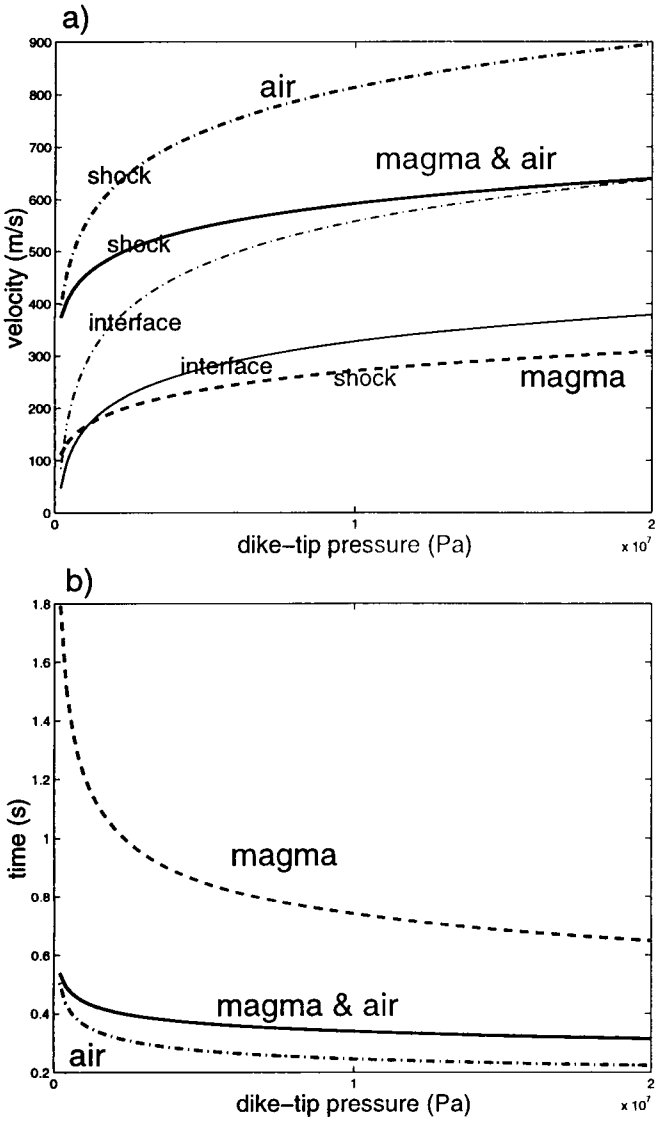


Figure 5: (a) The speed of the shock in air versus the dike-tip pressure P_t for the separated magma-air fluid (solid line), the shock in magma for a pure magma fluid (dashed line), and the shock in air only (dashed-dotted line). Speeds of pure air and magma-air fluid interfaces are thin versions of their shock lines. (b) The travel time of these shocks in a $200\,m$ long tunnel as function of P_t for the magma-air, magma and air fluids, respectively.

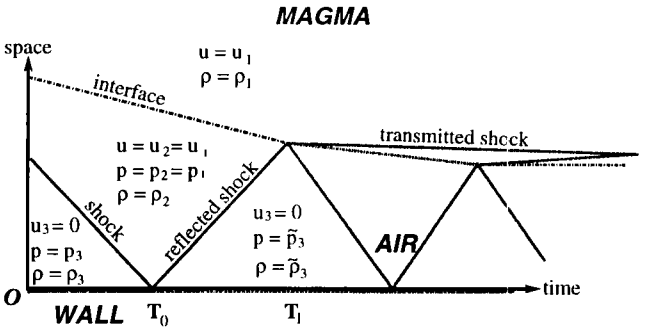


Figure 6: Space-time diagram of the resonating shock interactions between tunnel wall and melt-air interface. The vertical axis is space, the horizontal axis is time. The interface is denoted by a dashed-dotted line and shocks by a solid line. Initially, there is an upper state 1 in the basaltic fluid, an intermediate air state 2, and a lower air state 3 at rest. Effects of gravity and friction are absent.

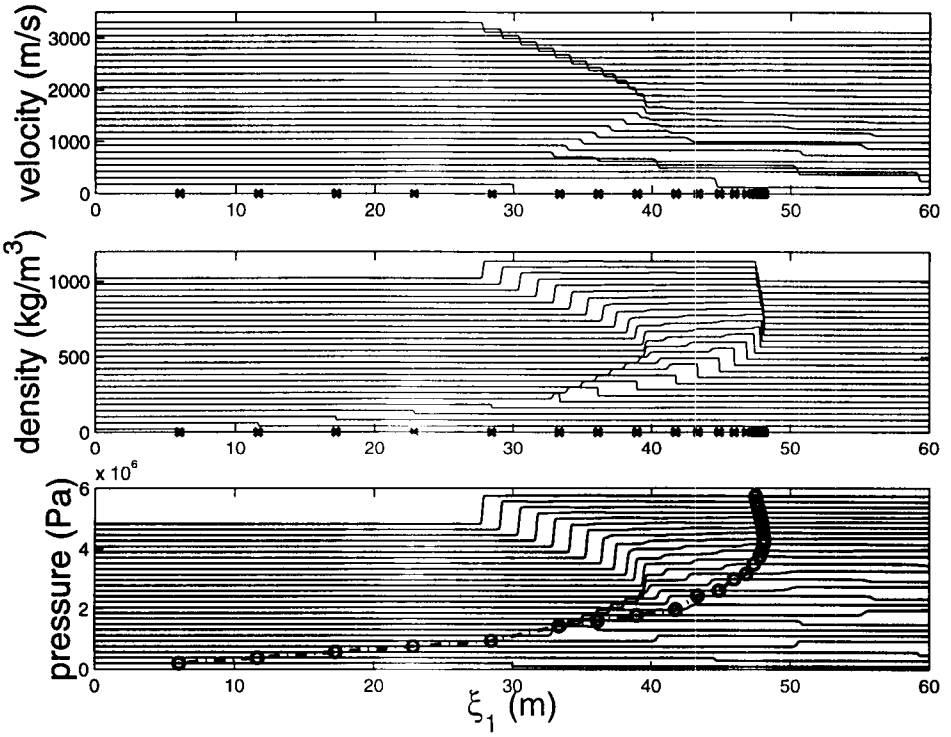


Figure 7: Velocity, density and pressure profiles are shown for a simulation of resonating shocks between a tunnel wall and magma-air interface. The wall of the horizontal tunnel is at the right; the boundary on the left is open and allows inflow (and outflow). The crosses at the ξ_1-axis indicate the forward moving positions of the interface, except in the pressure plot where circles indicate positions and pressures of the magma-air interface. Initial values of the pressures are $p_1 = 0.2\,MPa$ (magma), $p_2 = 0.2\,MPa$ (air) and $p_3 = 0.1\,MPa$ (air). Observe the strong compression of air as time advances. Gravity plays no role.

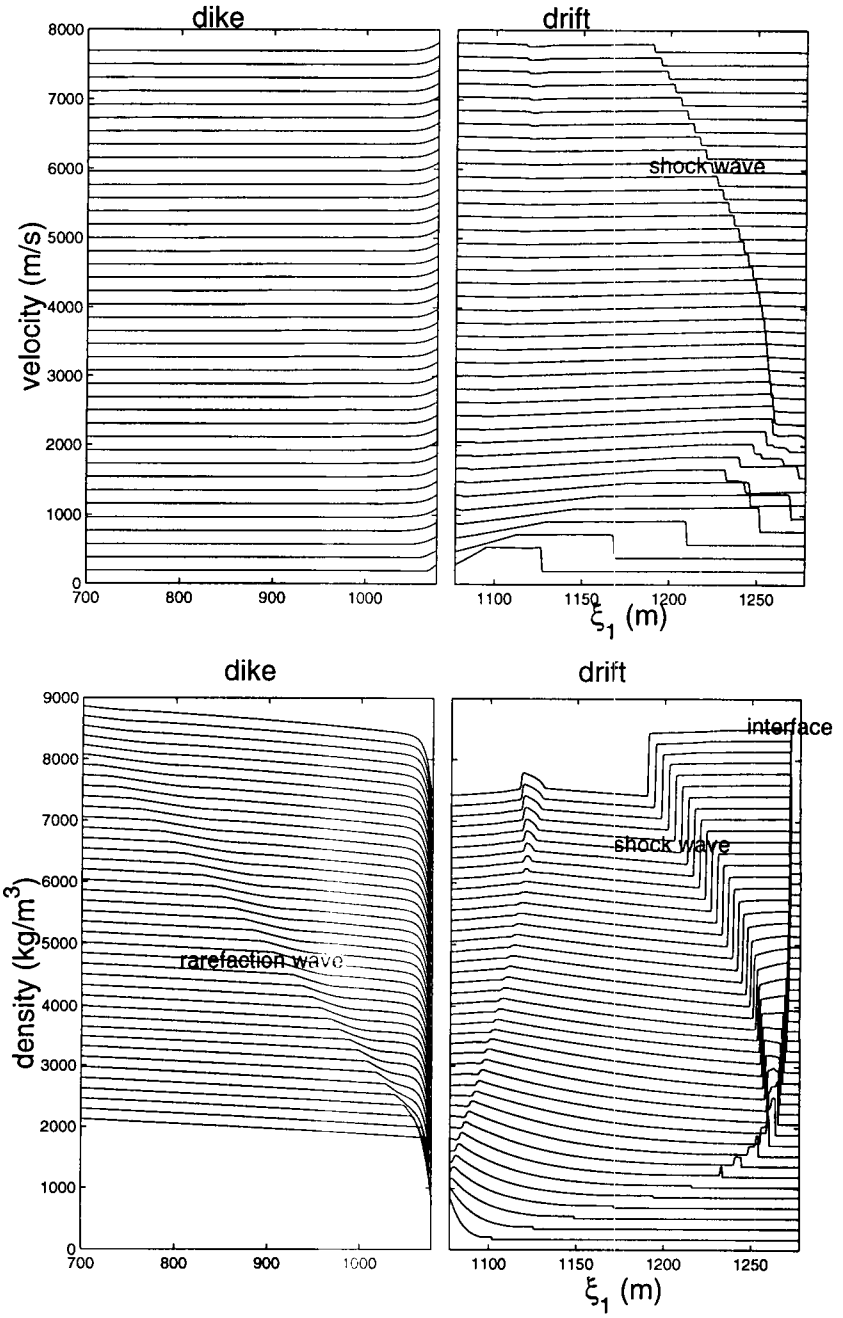


Figure 8: Velocity, density and pressure profiles are shown for magma-air interactions in a dike-drift system. The simulation encompasses 2.734 s and each of the 41 profiles is spaced 0.0683 s apart. Profiles in dike and drift have been separated to emphasize the different scaling in dike and drift. The rarefaction wave in the dike, the magma-air interface (only visible in the density profile), and the shock wave have been indicated. Short-time reference simulation 101.

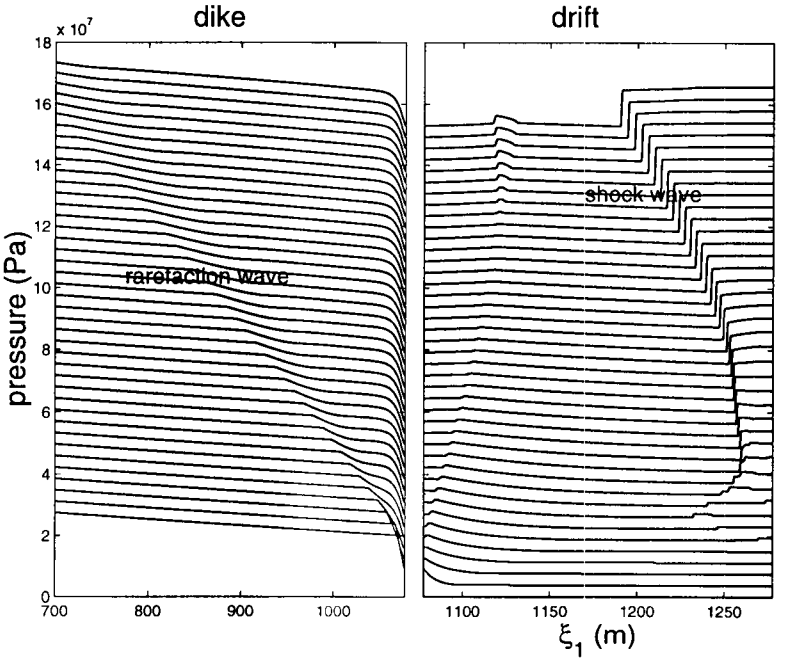


Figure 8: Continued, short-time reference simulation 101.

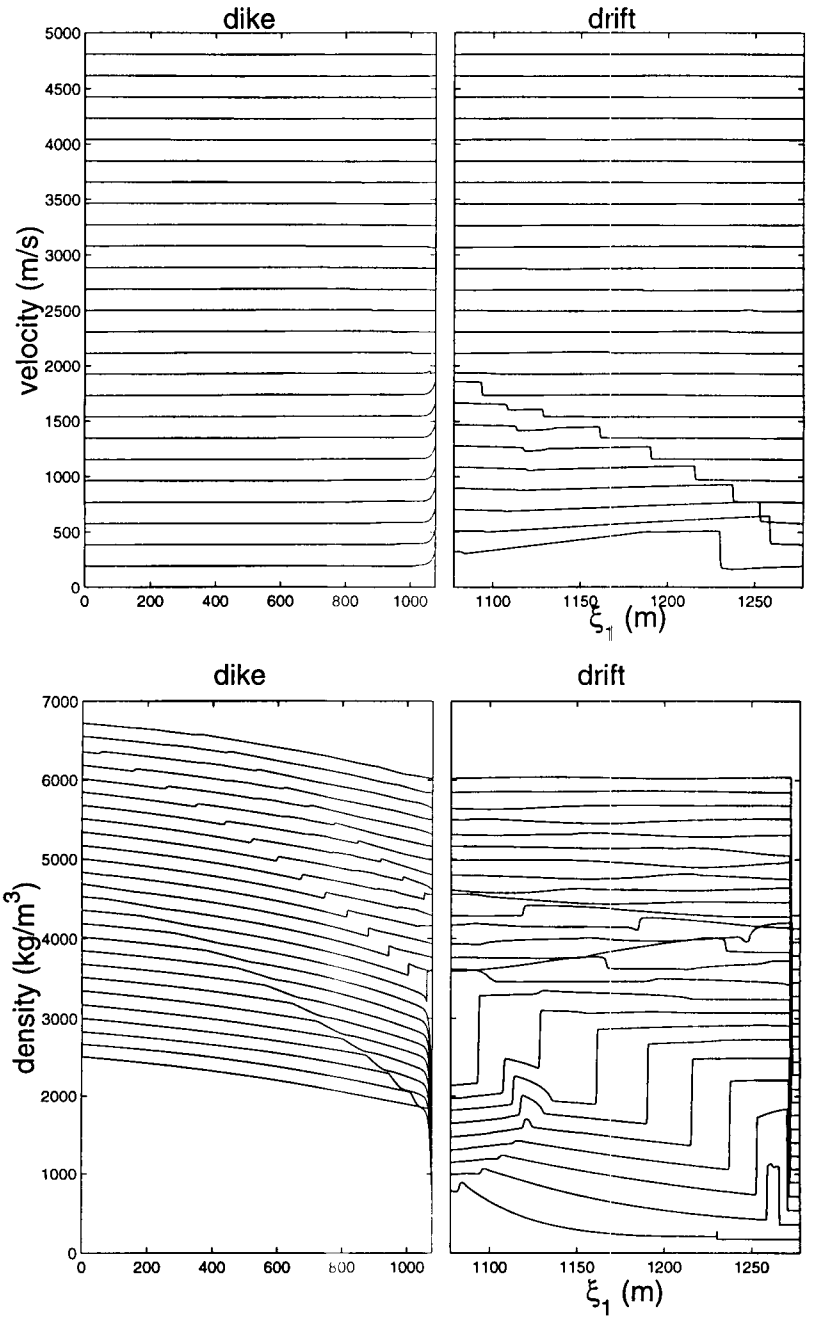


Figure 9: Velocity, density and pressure profiles are shown for magma-air interactions in a dike-drift system during 11.39 s in 26 profiles each spaced 0.456 s apart. Long-time reference simulation 060.

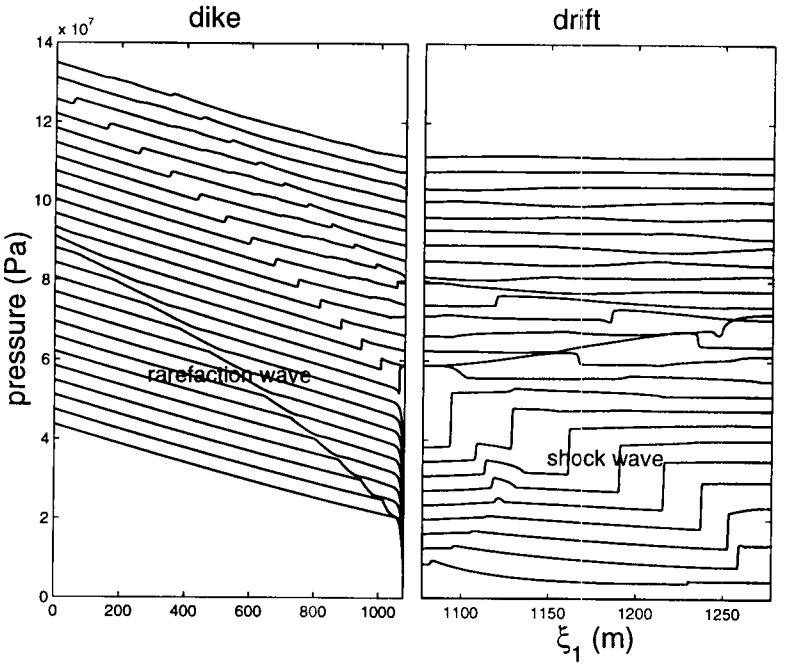


Figure 9: Continued, long-time reference simulation 060.

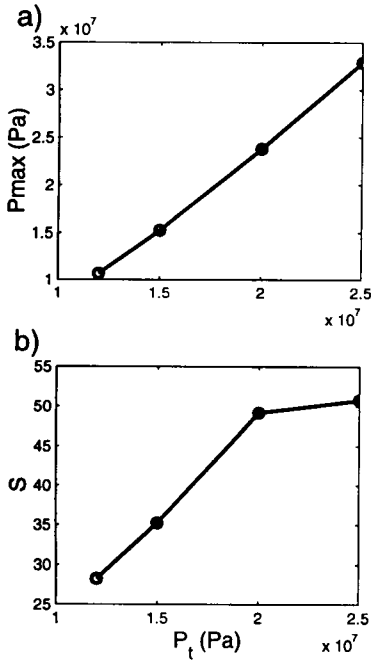


Figure 10: (a) Maximum pressure P_{max} in the tunnel, and (b) shock amplification S, the ratio of the maximum pressure drop in magma over the initial pressure drop across the shock in air, are shown versus initial dike-tip pressure P_t for runs: 064) $P_t = 25\,MPa$, 060) $P_t = 20\,MPa$, 061) $P_t = 15\,MPa$, and 063) $P_t = 12\,MPa$.

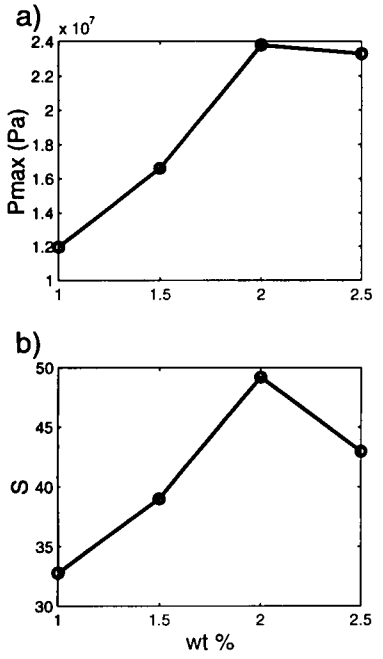


Figure 11: (a) Maximum pressure P_{max} in the tunnel, and (b) shock amplification S are shown versus volatile content n_0 for simulations (with different initial dike-tip pressures): 067) $n_0 = 2.5\,wt\%, P_t = 20\,MPa$; 060) $n_0 = 2\,wt\%, P_t = 20\,MPa$; 065) $n_0 = 1.5\,wt\%, P_t = 15\,MPa$; and 071) $n_0 = 1\,wt\%, P_t = 10\,MPa$.

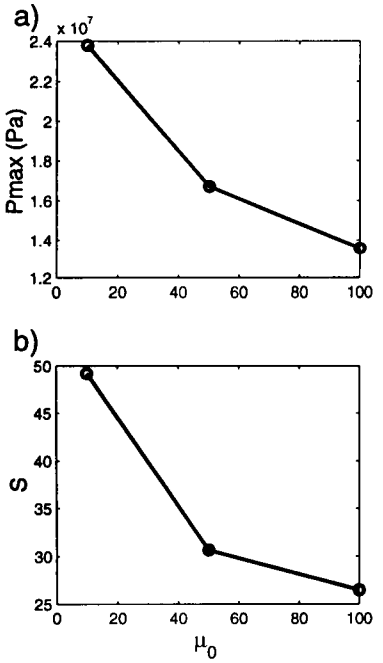


Figure 12: (a) Maximum pressure P_{max} in the tunnel, and (b) shock amplification S are shown versus frictional coefficient μ_0 for runs: 060) $\mu_0 = 10\,kg\,m^{-1}\,s^{-1}$, 072) $\mu_0 = 50\,kg\,m^{-1}\,s^{-1}$, and 073) $\mu_0 = 100\,kg\,m^{-1}\,s^{-1}$.

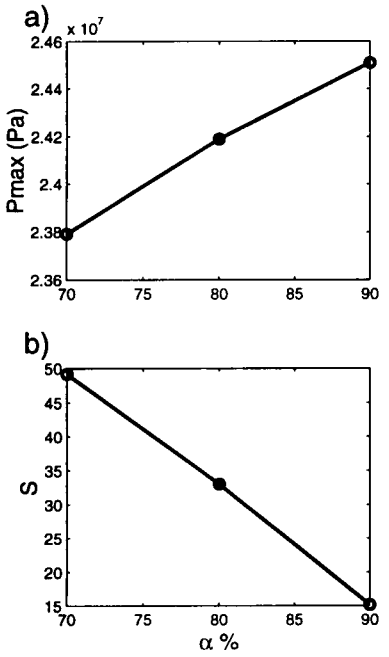


Figure 13: (a) Maximum pressure P_{max} in the tunnel, and (b) shock amplification S are shown versus fragmentation level α for runs: 060) $\alpha = 70\%$, 082) $\alpha = 80\%$, and 084) $\alpha = 90\%$.

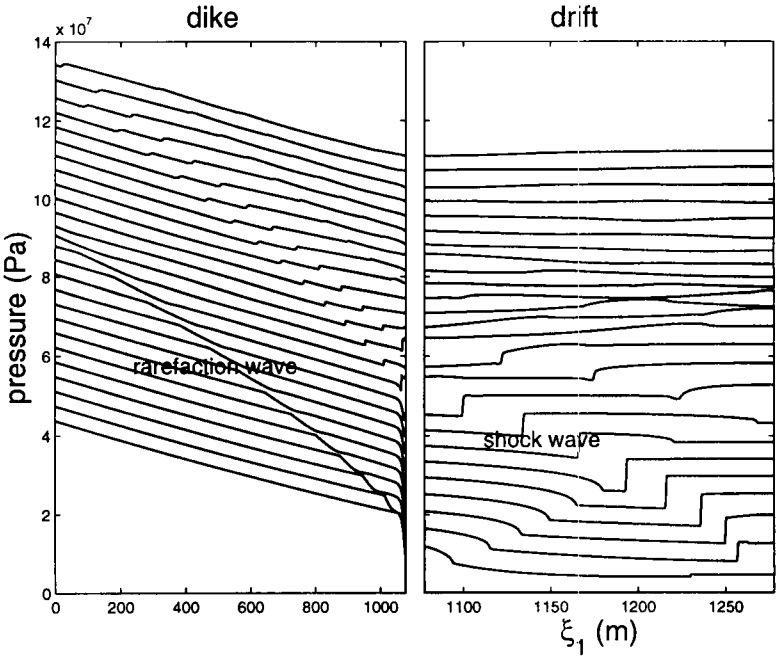


Figure 14: Pressure profiles are shown for magma-air interactions in a dike-drift system for simulation 082 with a fragmentation level of 80% instead of the 70% in the reference simulation. The scaling of the axes and time intervals is identical to the one in Fig. 9.

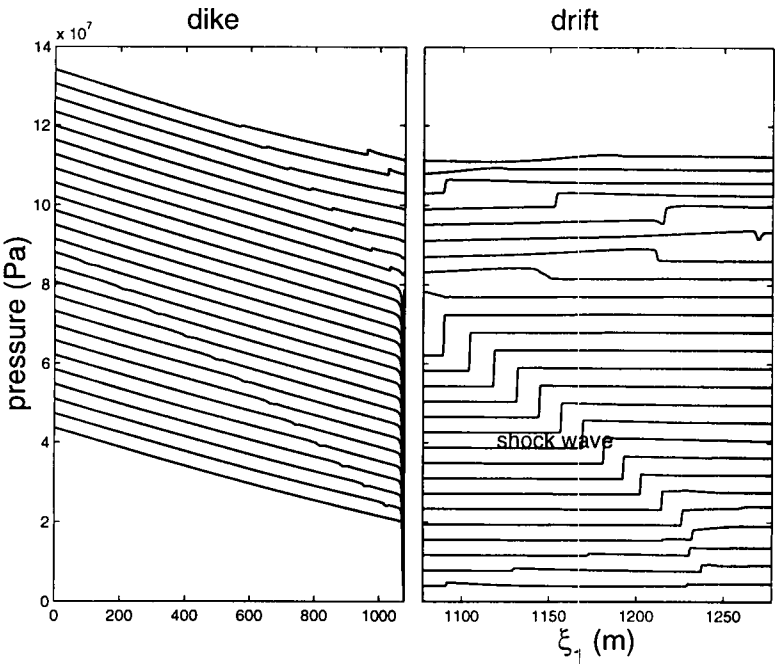


Figure 15: Pressure profiles are shown for magma-air interactions in a dike-drift system with a time-dependent cross section $A(\xi, t)$ increasing to its "reference" value during ten seconds. Simulation 097 ends at $t = 11.39\,s$. 26 profiles are shown separated $0.456\,s$ apart.

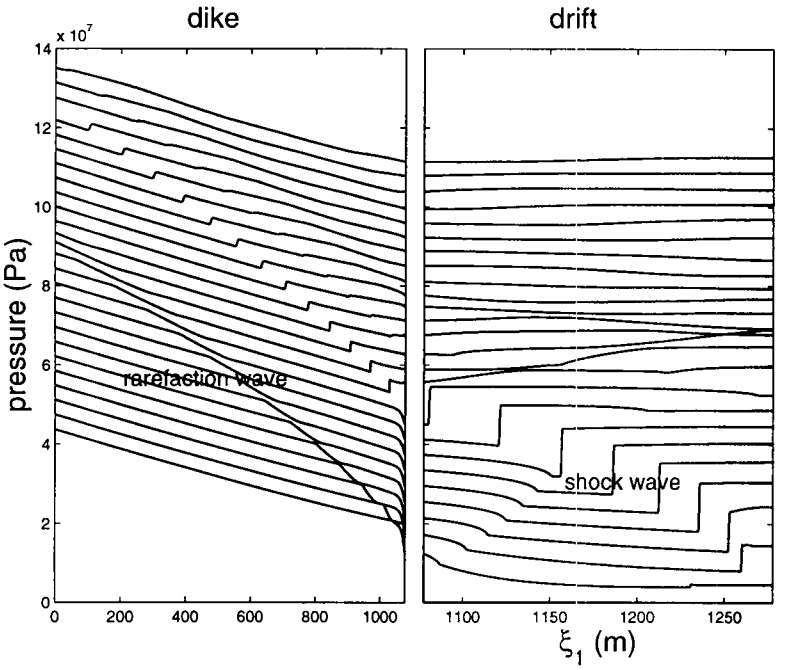


Figure 16: Pressure profiles are shown for magma-air interactions in a dike-drift system with a cross section A varying linearly between A_{dike} and A_{drift} in the transition zone. The scaling of axes and the offset in stack plots is the same as in Fig. 9. Simulation 076.

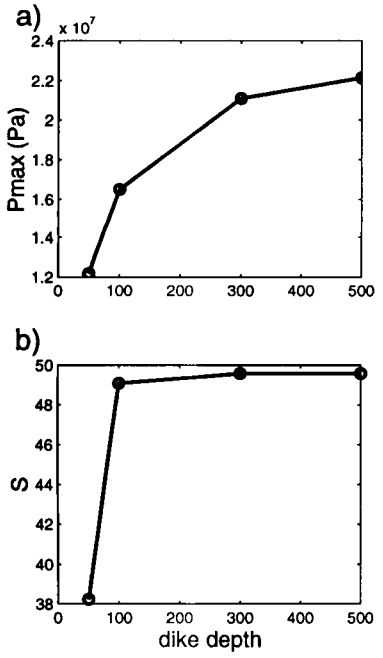


Figure 17: (a) Maximum pressure P_{max} in the tunnel, and (b) shock amplification S are shown versus the length of a finite-depth dike for runs: 102) $50\,m$, 103) $100\,m$, 104) $300\,m$, and 105) $500\,m$.

Scientific Notebook
20-1402-461
Inititals: CC

Chuck Connor
Sept 14, 2000

Entries into scientific notebook #115E for the period July - August, 2000 have been made by Dr. Chuck Connor. Entries into scientific notebook #115E for the pages section 1 page 1 to section 4 page 53 have been made by Dr. Chuck Connor. No original text entered into this scientific notebook have been removed.

Signature: R B Connor

Name: Dr. Chuck Connor

Date: Sept 14, 2000

I have reviewed scientific notebook 115 E and find it in compliance with QAP-001. There is sufficient information regarding procedure used for conducting the research and acquiring and analyzing the data so that another qualified scientist could repeat the activity or activities recorded in this scientific notebook

H. Lawrence McKague 9/27/00

H. Lawrence McKague
GLGP Element Manager

17 of 17

SCIENTIFIC NOTEBOOK 115E

CHUCK CONNOR

IGNEOUS ACTIVITY
20.01402.461

2000–2001

SCIENTIFIC NOTEBOOK 115E

CHUCK CONNOR

IGNEOUS ACTIVITY
20.01402.461

2000–2001

SOUTHWEST RESEARCH INSTITUTE
CENTER FOR NUCLEAR WASTE REGULATORY ANALYSES
SAN ANTONIO, TEXAS

SCIENTIFIC NOTEBOOK 115E

CHUCK CONNOR

PROJECT

IGNEOUS ACTIVITY
20-1402-461

2000-2001

PARTS IN THIS NOTEBOOK:

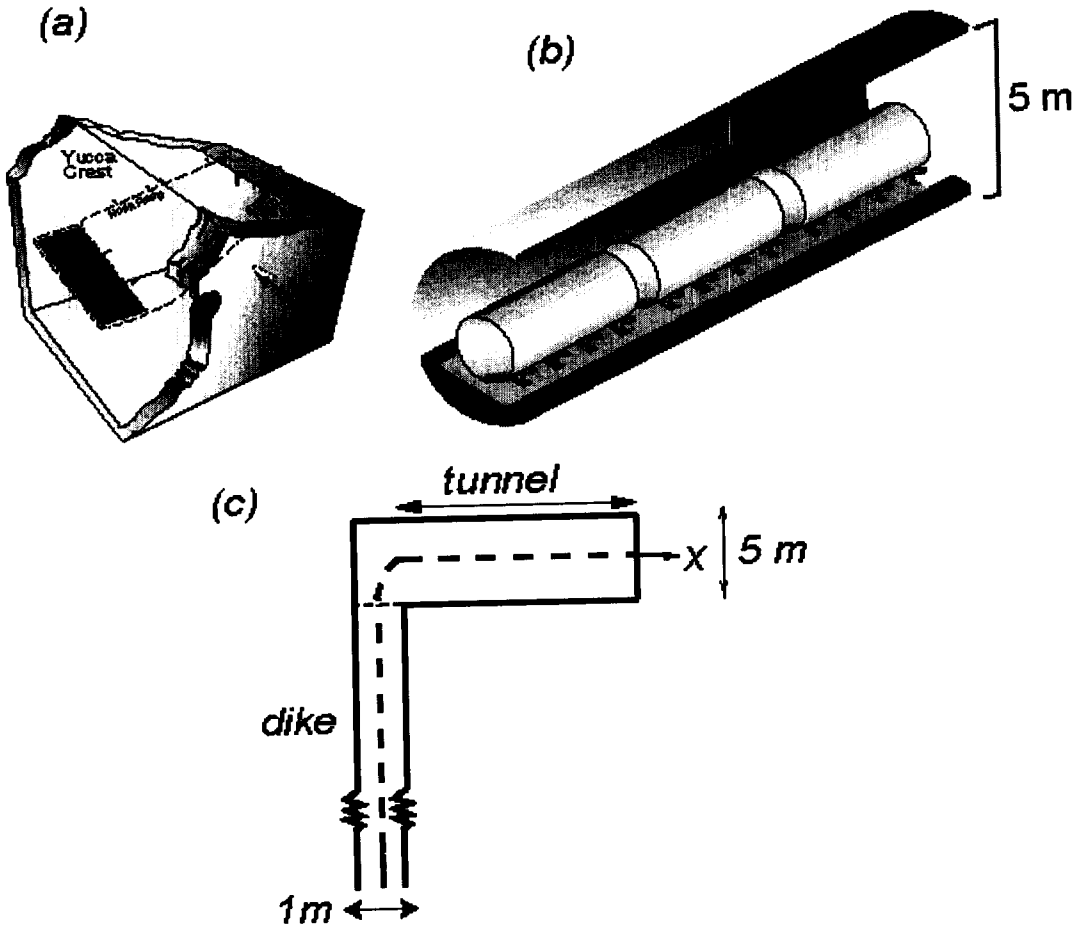
Section 1: development of steady-state code for magma flow
Section 2: draft paper on steady-state magma flow

Scientific Notebook #115-(cc)
20-1402-461
Inititals: CC

Chuck Connor
April 1, 2001

The following is a draft paper prepared on modeling magma repository interactions

Woods et al., Figure 2



schematic of magma intersecting drift

Chuck Connor
April 1, 2001

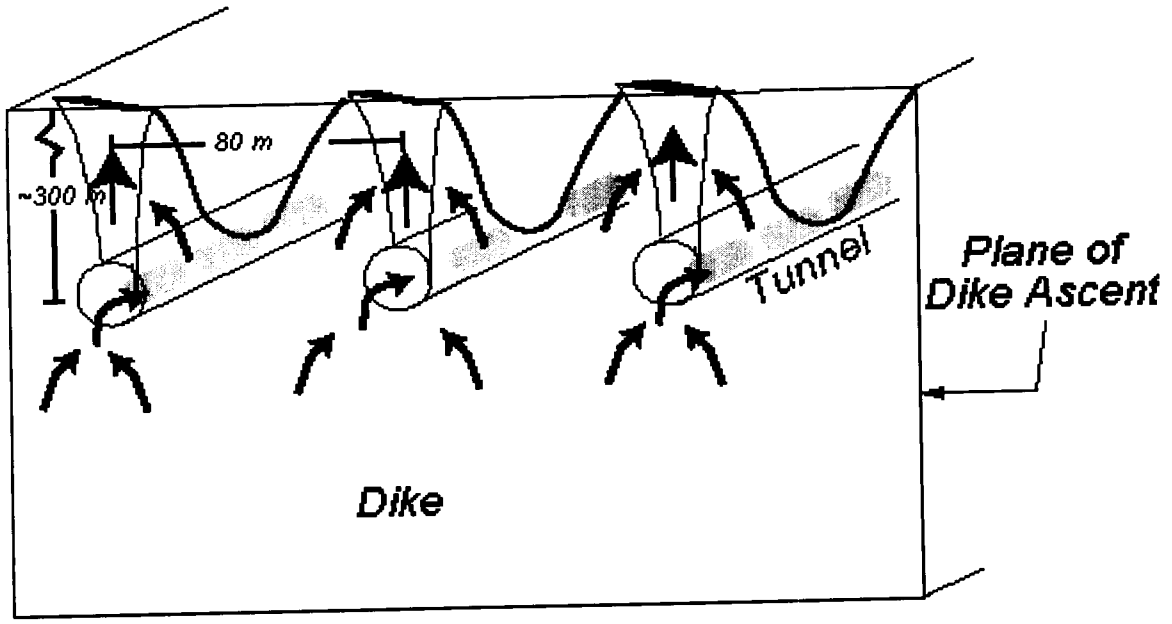


Figure 4a

Schematic of magma developing steady-state flow above drift

A central problem is the development of steady-state magma flows in drifts. This effects magma-waste package interaction (time available for magma to transport waste, thermal response of waste packages, etc) and the number os waste packages likely to be transported to the surface (or their contents) in a single volcanic event.

Previous work by Bokove and Woods (see earlier volumes of 115E) show that initial shock waves may develop. Here, simple calculations are done to show the steady-state flow conditions in the magma. The following code is annotated to show how this is done.

Scientific Notebook #445 (CC)
20-1402-461
Inititals: CC

Chuck Connor
April 1, 2001

The following code was developed to model steady-state flow in the drift:

```
/* Program Name: tunnel_flow.c
 * Written in Fortran by: Andy Woods, Nov. 11, 00
 * Translated to C and revised by: Chuck Connor, Dec 28, 00
 *
 * Purpose: This code estimates the steady-state flow conditions
 * of magma ascending from depth in a dike, intersecting a repository
 * drift and subsequently ascending to the surface. Flow character is
 * described on output by pressure, volume fraction gas, and flow velocity
 * as a function of depth, and distance along flowpath. Depth and distance
 * along flowpath differ because flow is assumed to occur over some
 * specified distance in the horizontal repository drift (tunnel).

 * Flow character is determined based on iterative solution of the
 * flux and pressure conditions along the flowpath, using a finite
 * difference approximation.

 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>


/*************************************************************************/
//define the following as constants

#define PI 3.141592654
#define R 462.0   //gas constant J/(kg degK)
#define G 10.0  //gravity (m/s*s)
#define S 0.000003  //solubility const for Henry's Law (1/sqrt(Pa))

 main () {

  /*************************************************************************/
  //variable assignments
```

```
int isolve, inogas; //flags for convergence and gas fraction
double af, a, ah, alst, anow, w; //area of the fracture or conduit or tunnel,and conduit width

double v;  // magma viscosity
double t;  // magma temperature
double go; // magma gas fraction, initially
double fragmentation; //volume fraction gas at fragmentation
double delp; // overpressure term
double q, qmin,qmax,qav; //magma flux
double gas, den, vol, vel; //mixure gas fraction, density, volume fraction gas, velocity
double denbas; //basalt magma density
double surface_denlith, denlith;  //rock density at the surface and deeper
double c;  //drag coefficient
double prt, iprt; // control print statements
double p, pmin, pmax,po,p1,p2, c1, c2,c3,plith; //pressure terms
long double alhs, arhs;
double x,dx,y; //distance along flowpath, increment, and depth
double origin_depth;  //depth of origin of basalt flow
double dp,dc2;
double truex, truey;
double dike_flow_area, dike_width;
double tunnel_depth, tunnel_flow_area, tunnel_width;
double tunnel_flow_length;
/*****************************************************************/
// the following variable assignments are
//commonly changed for successive runs

//set basalt density (kg/m*m*m)
denbas = 2600.0;

// set lithologic density (kg/m*m*m)
surface_denlith = 2400.0;

//set temperature (deg C)
t = 1200.0;


//initial gas content (wt fraction)
go = 0.02;
```

```
//set fragmentation value (volume fraction gas)
fragmentation = 0.7;

//initial c (drag coefficient - dimensionless - typically 0.001-0.01)
c = 0.01;

//set the viscosity (Pa-s)
v = 10.0;

//set the overpressure (Pa)
delp = 0.0;

//set the step size (m)
dx = 1.0;

//set origin depth for basalt (m)
origin_depth = 30000.0;

//set tunnel depth
tunnel_depth = 300.0;

//set tunnel cross section area
tunnel_flow_area = 50.0;

//set tunnel 'width'
tunnel_width = 5.0;

// set the tunnel length (m) - this is the distance
// magma flows in the tunnel, rather than the
// true length of the tunnel
tunnel_flow_length =500.0;

//set the area of the dike available to flow
// into the tunnel - equal to the drift spacing
// assuming that all of the dike volume flows
//into drifts within the repository foot print
dike_flow_area = 80.0;

//set the dike width  (above and below tunnel)
```

```
dike_width = 1.0;

// set the fissure aperture (area through which
 // flow occurs between tunnel and  surface)
 af = 10.0;


 /*********************************************************/
 //initialize the flux
 qmin = 1000.0;
 qmax = 10.0e7;

 //initialize flags
 isolve = 0;
 inogas = 0;
 prt = 1000.0;

 /*********************************************************/
 //outer loop iterates  till convergence on flux
 while (isolve == 0) {

  //check if convergence has occurred
  // if little change in flux between steps then
  // isolve set to 1
  if (qmax-qmin > 0.0001*qmin) {
   qav = (qmin+qmax)/2.0;
   q =  qav;
   isolve = 0;
  }
  else{
   isolve = 1;
  }

  //fracture aperture terms
  a = af;
  alst = a;

  //initialize pressure solver
  pmin = 100000.0;
```

```
pmax = pmin*1000.0;


// find the pressure for the given flux
while ((pmax - pmin) > pmin*0.00001) {
 p = (pmin+pmax)/2.0;

 alhs = p*p+q*q*0.5*S*R*t*sqrt(p)/(2.0*a*a);
 arhs = q*q*(go*R*t+S*pow(p,1.5)/(2.0*denbas))/(a*a);
 if (alhs > arhs) pmax=p;
 if (arhs > alhs) pmin = p;
}
//verify that the pressure solved with print statement
printf("%f %f %f\n", pmin,pmax,p);


c2 = (S*sqrt(p)/denbas + go*R*t/p + S*R*t/sqrt(p))/(a*a);
c2 = q*q*c2 + p;

//atmosphere pres. in Pa
//initialize plith to this value
po = 100000.0;
plith = po;

inogas = 0;
if (p < po) {
p=po;
c2 = c1*p+c3/p;
}

//initialize the distance
// x is the distance along the flowpath
//y is the depth - they vary with tunnel length
x = 0.0;
y = 0.0;

iprt = 100000.0;

/*****************************************************/
```

```
//loop to calculate pressure variation along flowpath
//loop from the surface down to total depth
while(y < origin_depth) {

 //Geometry changes with depth
 //fracture
 // pressure change with depth in the fracture
 // so ah = H(x) = 1.0
 ah = 1.0;

 // area of the condiut
 a = af;
 anow = a;

 //WIDTH OF THE CONDUIT
 w = dike_width;
 //how does depth (y) increment with distance along
 //flow path x
 y=x;

 // tunnel
 if (x > tunnel_depth) {
     ah = 0.0;
     w = tunnel_width;
     a = tunnel_flow_area;
     anow = a;
     y = tunnel_depth;
 }

 //dike

 if (x > (tunnel_depth + tunnel_flow_length)) {
     ah=1.0;
     w = dike_width;
     a = dike_flow_area;
     anow=a;
     y = x-tunnel_flow_length;
 }
```

```
/*****************************************************************/
// calculate the flow properties as a function of depth
// for the current estimate of the flux, q

// if the volume fraction gas is zero then the properties are:
if (inogas == 1) {
    dp = -v*q / (12.0*w*w*a*denbas) - c*q*q/(denbas*a*a*w);
    dp = dp -denbas*G*ah;
    p = p-dp*dx;
    den=denbas;
    gas = 0.0;

}

//if the volume fraction gas is greater than zero:
else {

    gas = go-S*sqrt(p);
    den = 1.0/ ((gas*R*t/p) + (1.0-gas)/denbas);
    vol = (gas*R*t/p) * den;


    if (vol > fragmentation) v=0.0;
    dc2 = -v*q/(12.0*den*w*w*a)-c*q*q/(den*a*a*w);
    dc2 = dc2 - den*G*ah;
    c2 = c2 - dc2*dx;

    if (x > tunnel_depth-5.0  && x < tunnel_depth+5.0)
      c2 = c2+q*q*(1.0/(anow*anow) - 1.0/(alst*alst))/(2.0*den);

    if ((x > tunnel_depth-5.0+ tunnel_flow_length) && (x < tunnel_depth+5.0+
tunnel_flow_length))
      c2 = c2+q*q*(1.0/(anow*anow) - 1.0/(alst*alst))/(2.0*den);

    p1 = p;
    p2 = 10.0*p;

    while (p2-p1 > 0.0001*p1) {
```

```
        p = (p1+p2)/2.0;
        c1 = p+q*q*go*R*t/(p*a*a);
        c3 = q*q*S*R*t/(a*a*sqrt(p)) + q*q*S*sqrt(p)/(a*a*denbas);

        if (c1 > c2+c3) p2=p;
        if (c1 < c2+c3) p1 = p;
      }
    }

    //assume that the density of the crust veries linearly with depth
    denlith = surface_denlith + y*6.0/300.0;
    plith = plith+denlith*G*ah*dx;

    // fact was calculated in the original fortran code
    // but not used
    //if (x < 850.0) iprt+=10.0;
    //  if (x>400.0 && x < 415.0) fact = q*q/(den*a*a*11000.0 * G * 5.0);

    //change the print frequency at depth
    prt += 10.0;
    if (y > 2000.0) prt = prt - 9.9;

    //write the output
    if (isolve == 1 && prt > 100.0) {
      vol = (gas*R*t/p)/((gas*R*t/p)+(1.0-gas)/denbas);
      vel = q/(den*a);

      // pressure already incremented so adjust x and y for output
      truex = x+1;
      truey = y+1;
      printf("%f %f %f %f %f %f\n", truex,truey, p, plith, vol, vel);
      prt = 0;
    }

    //determine if the pressure is sufficinetly
    //high to cause complete dissolution of the
    //gas phase
    if (sqrt(p) > go/S) inogas = 1;
```

```
    alst = a;

    //increment x
    x+=dx;

  } // end while (y < 30000.0);
  printf ("q = %f\n",q);

  //if the flux has converged then
  //now is the time to leave the
  //isolve (flux) loop
  if (isolve == 1) break;

  // reset the gas flag
  inogas = 0;

  //adjust the flux until dike and lithostatic
  //pressure match at origin_depth
  //can adjust for overpressure delp

  if (p < plith+delp) qmin = qav;
  if (p > plith+delp) qmax = qav;

 } // end of isolve
} //end main
```

Scientific Notebook #115 (CC)
20-1402-461
Inititals: CC

Chuck Connor
April 1, 2001

The following is a draft paper prepared on modeling magma repository interactions

Modeling an Explosive Eruption of Basaltic Magma into the Proposed High-Level Radioactive Waste Repository at Yucca Mountain, Nevada, USA

Andrew W. Woods[1], Steve Sparks[2], Onno Bokhove[3], Anne-Marie LeJeune[2],

Charles B. Connor[4], Brittain E. Hill[4]

1. BP Institute, University of Cambridge, Cambridge, CB3 OEZ, UK

2. Centre for Environmental and Geophysical Flows, University of Bristol, Bristol, UK

3. Faculty of Mathematical Sciences, University of Twente, Enschede, The Netherlands

4. Center for Nuclear Waste Regulatory Analyses, Southwest Research Institute, San Antonio, Texas, USA

March 29, 2001

Information potentially subject to copyright protection was redacted from Section 2, pages 1 through 23. The redacted material is from the following reference: Woods, et al., Title listed above. DOI 10.1029/2002GL014665. U.S.A. Geophysics Research Letter. 2002.

Scientific Notebook #115(E)
20-1402-461
Inititals: CC

Chuck Connor
April 16,2001

Entries into scientific notebook #115E for the period July - August, 2000 have been made by Dr. Chuck Connor. Entries into scientific notebook #115E for the pages section 1 page 1 to section 2 page 23 have been made by Dr. Chuck Connor. No original text entered into this scientific notebook have been removed.

Signature: Chuck B Connor

Name: Dr. Chuck Connor

Date: April 16, 2001 July 5, 2001

I have reviewed scientific notebook 115E and find it in compliance with QAP-001. There is sufficient information regarding procedure used for conducting the research and acquiring and analyzing the data so that another qualified scientist could repeat the activity or activities recorded in this scientific notebook

H. L McKague 4/17/01

H. Lawrence McKague
GLGP Element Manager

SCIENTIFIC NOTEBOOK 115E IS TO BE ARCHIVED EFFECTIVE 7/11/01

H. Lawrence McKague 7/11/01

ADDITIONAL INFORMATION FOR SCIENTIFIC NOTEBOOK #: 115E
(17 of 17)

Document Date:	04/01/2001
Availability:	Southwest Research Institute® Center for Nuclear Waste Regulatory Analyses 6220 Culebra Road San Antonio, Texas 78228
Contact:	Southwest Research Institute® Center for Nuclear Waste Regulatory Analyses 6220 Culebra Road San Antonio, Texas 78228 Attn.: Director of Administration 210.522.5054
Data Sensitivity:	■"Non-Sensitive" ▫ Sensitive ▫"Non-Sensitive - Copyright" ☐ Sensitive - Copyright
Date Generated:	07/01/2001
Operating System: (including version number)	Windows
Application Used: (including version number)	WordPerfect
Media Type: (CDs, 3 1/2, 5 1/4 disks, etc.)	1 CD
File Types: (.exe, .bat, .zip, etc.)	wpd
Remarks: (computer runs, etc.)	Media contains: WordPerfect files relating to igneous activity for this notebook.