CENTER FOR NUCLEAR WASTE REGULATORY ANALYSES

Goodluck Ofoegbu
(210) 522 6641

Table of Contents

Table of Contents Contd.

February 03 1999

GHO

Repository-Scale Thermal-Hydrological Model
of the Emplacement Area of the Proposed Repository at Yucca Mountain

Objective: To examine the effects of repository thermal loading on the distributions of water saturation, percolation flux, and flow patterns within the repository block from the ground surface to the emplacement horizon. The effects of site stratigraphy, orientation of stratigraphic interfaces, major faults such as the Ghost Dance Fault and the Solitario Canyon Fault, and infiltration rate will be examined.

The hydrologic characteristics of the fractured rock mass will be represented using the dual-continuum model (DCM). Analyses will be conducted using the METRA modules of MULTIFLOW (P.C. Lichtner and M.S. Seth, 1998. *MULTIFLOW User's Manual Version 1.2*β. San Antonio, TX: Center for Nuclear Waste Regulatory Analyses).

Both two-dimensional and three-dimensional models will be considered for the analyses. However, analyses will be performed first using a series of two-dimensional sections. The first of such sections consists of a vertical plane parallel to the emplacement drifts, along line AA' in the figure on page 4. The DD' portion of the section line corresponds approximately with the alignment of drift #64 (#1 is at the north end of the emplacement-drift array). The section will extend vertically from the ground surface to the water table and will be extracted from the Department of Energy's three-dimensional geological framework model of Yucca Mountain.

Investigator Goodluck I Ofoegbu

Signature

Initials GIO

GIO

Entries on page 3-9 by GIO 2/3/99

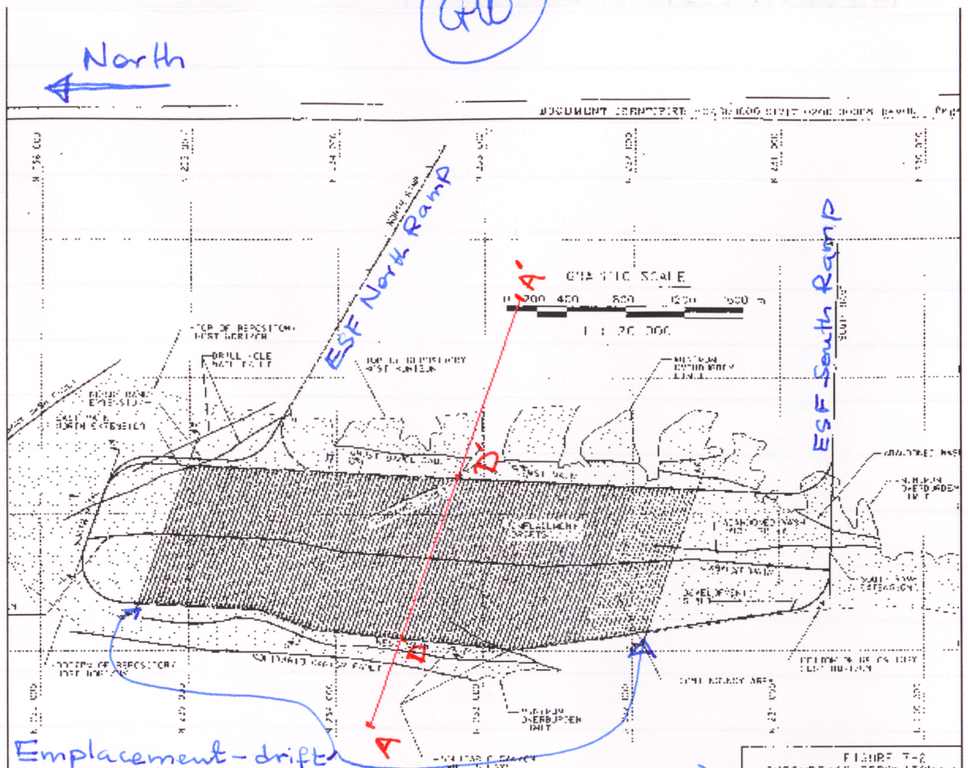


FIGURE 7-2
SUBSURFACE REPOSITORY L
W RESPECT TO GEOLOGIC BO

Coordinates: A (546,749E; 4,078,957N)
A' (549,758E; 4,077,970N)
D (547,213E; 4,078,798N)
D' (548,571E; 4,078,354N)

DD' corresponds approximately with the alignment of drift #64 from North end of the ~~emplacement~~ drift array.

The approximate location of the section line (AA') is also shown in the figure on p. 5, which is of better quality than the one on this page. The figure on page 5 was projected to a different coordinate system (by Ron Martin) to obtain the figure on this page.

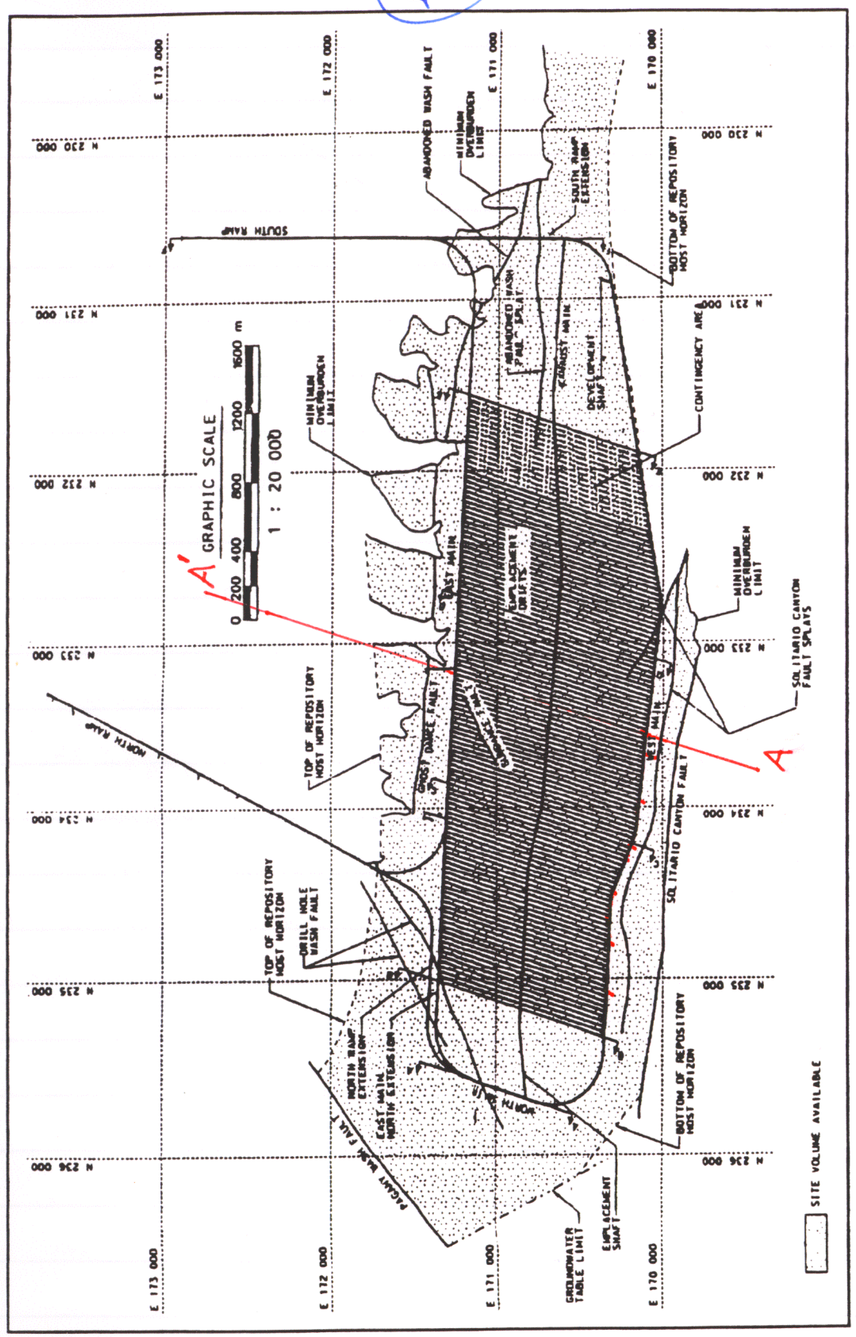


Figure 2. Proposed layout of the underground facility (Civilian Radioactive Waste Management System, Management and Operating Contractor, 1997a). The Exploratory Studies Facility loop consists of the "north ramp," "south ramp," and the portion of the "east main" between the two.

Temperature Boundary Conditions at the Water Table

The water table is located at a depth of 300 - 350 m below the emplacement horizon. The base of the model will be located at or near the water table. In a previous repository-scale model (Ofoegbu et al, 1999; reference cited on p. 7) the thermal boundary condition at the water table (base of model) was specified as fixed temperature. However, other thermal-hydrological models (Buscheck and Nitao, 1993; Haukwa and Wu, 1996) that include 500 - 1000 m of the saturated zone indicate significant increase in temperature at the water table (see refs on p. 7).

For the current project, a separate simpler model will be used to estimate the temperature history at the water table. The calculated temperature history will be applied as boundary condition in the more rigorous model.

The following assumptions are made in developing a model to estimate temperature history at the water table

2/3/4/~~5~~/6

(1) The temperature history is determined mainly by the thermal loading and heat conduction through the surrounding rock.

(2) Convective flow in the saturated zone is also likely to be important.

(3) Moisture and gas flow in the unsaturated zone have negligible effects on water-table temperature.

OFOEGBU, G.I., A.C. Bagtzoglou, R.T. Green, and M.A. Muller. 1999. Effects of perched water on thermally driven moisture flow at the Proposed Yucca Mountain repository for high-level waste. Nuclear Technology, 125: 235 - 253.

BUSCHECK, T.A. and J.J. Nitao. 1993. Repository-heat-driven hydrothermal flow at Yucca Mountain, Part I: Modeling and analysis. Nuclear Technology, 104: 418 - 448.

HAUKWA, C. and Y.S. Wu. 1996. Thermal loading studies using the unsaturated zone model. IN G.S. Bodvarsson and T.M. Bandurraga (ed), Site-scale unsaturated zone model of Yucca Mountain, Nevada. Berkeley, CA: Lawrence Berkeley National Laboratory.

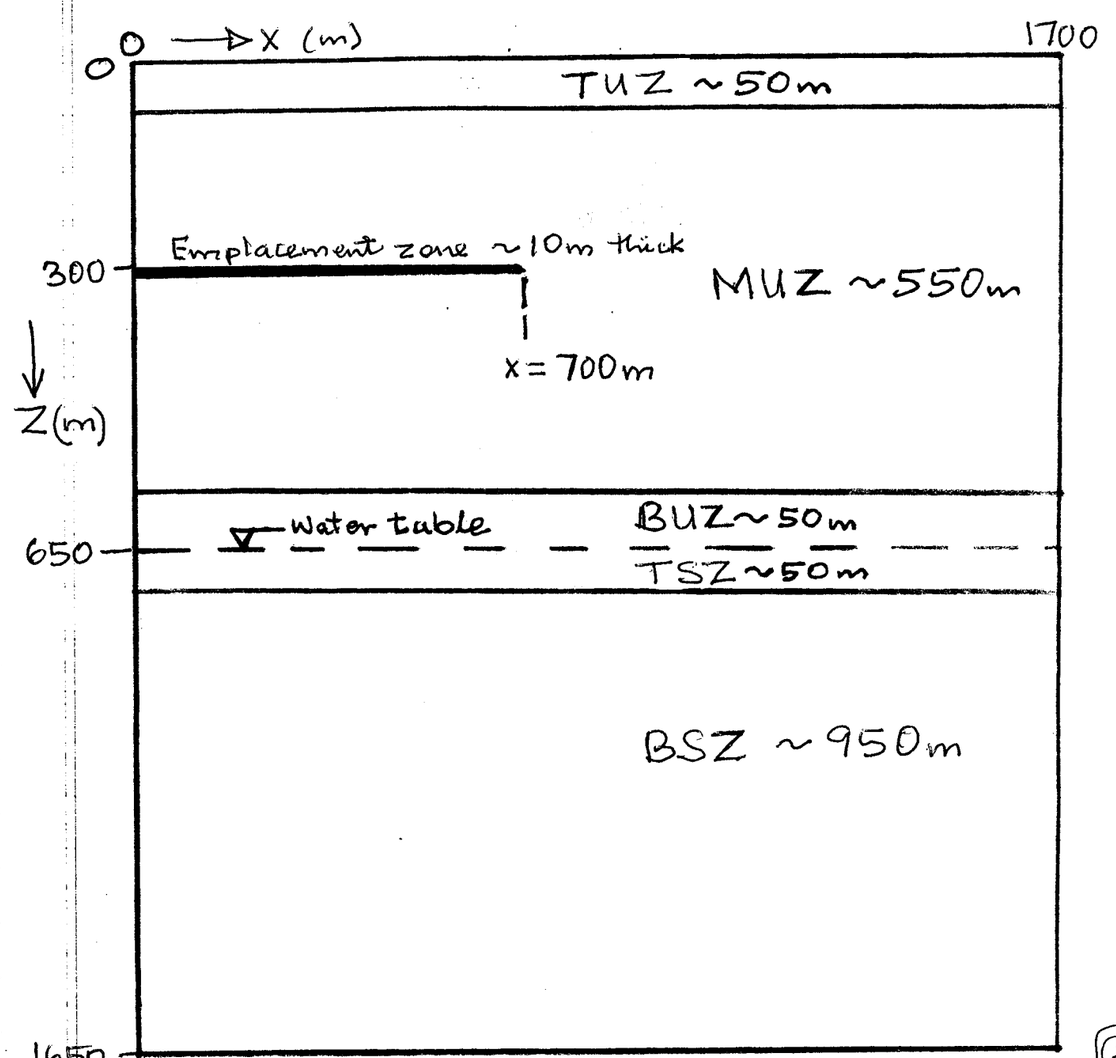
As a result, detailed hydrological modeling of the unsaturated zone is considered unnecessary for estimating the temperature history at the water table.

The model proposed for the analysis, (GWO) to estimate water-table temperature history (referred to hereafter as model WT01) consists of a rectangular vertical plane that extends to a depth of 1650 m below the ground surface (1000 m below the water table). The model is sketched on p. 9.

The unsaturated zone is represented by three rock layers: a 2/3/99 (GWO) 50-m thick layer at the top that is assigned properties representing the PTn (Paintbrush tuff, nonwelded) unit, a thick middle layer assigned TSw2 (Topopah Spring welded unit 2) representative property; and a 100-m zone at the base, which extends below the water table and is assigned CHn (Calico Hills nonwelded unit) properties.

The saturated zone below the CHn unit is represented as one rock layer assigned the properties of the Prow Pass unit.

(GWO)



(GWO)

Vertical line at X=0 is a symmetry plane (zero normal flux)

Vertical line at x=1700 treated as far field (zero normal flux)

Base: Fixed temperature, pressure and saturation

Top: Fixed temperature and mass flux

Entries on p. 3-9 by GWO 2/3/99

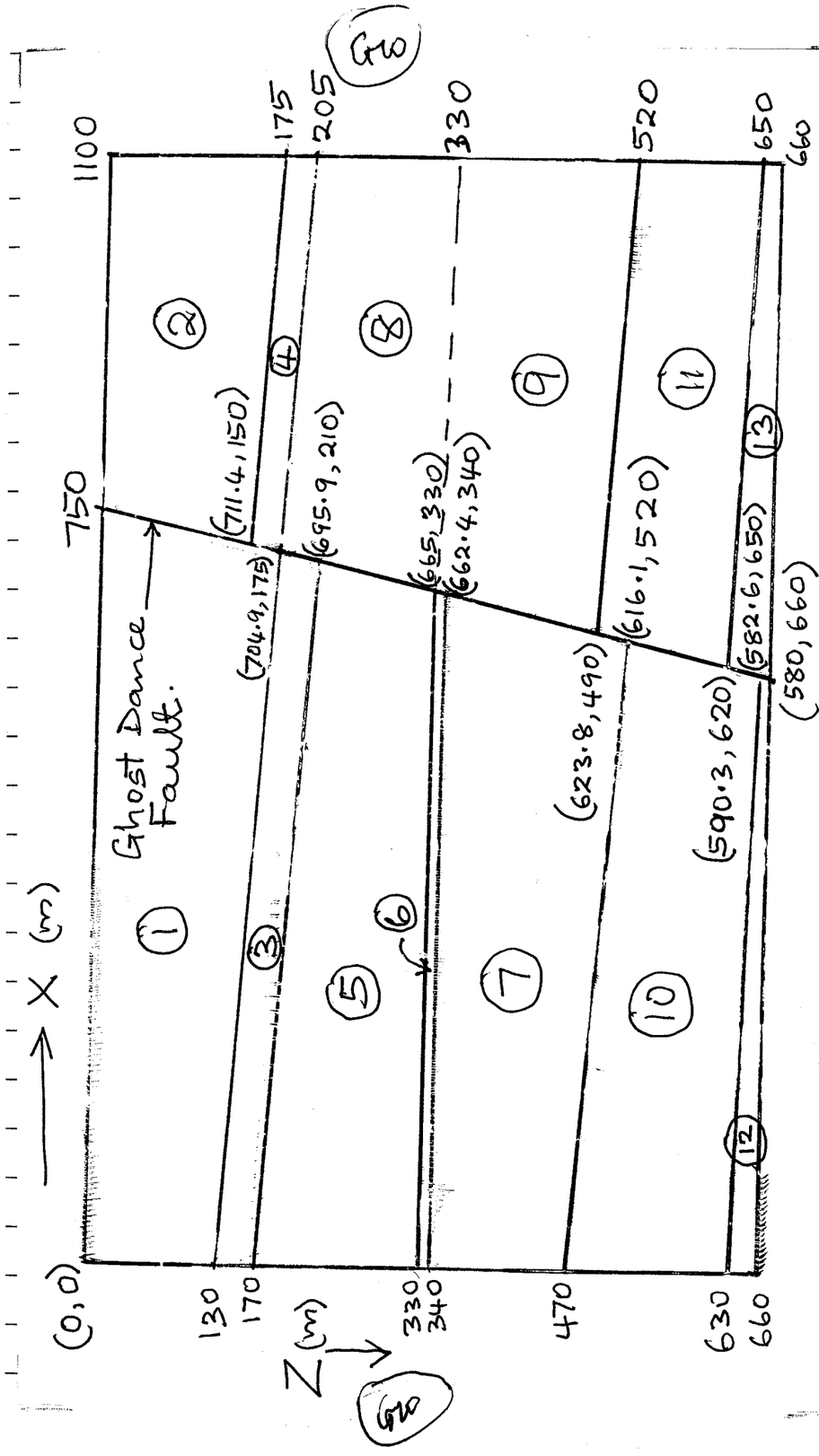
February 10 1999 Pages 10-12 GW entry begun 2/10/99

Development of Unstructured Grid Input ~~Files~~ Data Blocks for ~~METRA~~ METRA

GW 2/10/99

Further development of the simplified model described on p. 9 was suspended to work on the development of unstructured grid input for METRA. Unstructured grids (i.e., grids that don't necessarily follow a rectangular pattern) will be required in the repository-scale model ~~because~~ in order to represent the correct geometry of stratigraphic interfaces and faults that do not intersect at right angles. For example, the figure on p. 11, adapted from Foeglon et al (1999) [cited on p. 7] illustrates nonorthogonal intersections between stratigraphic interfaces and a fault. Such geometry will be encountered in the repository-scale model and will be best discretized using unstructured grids.

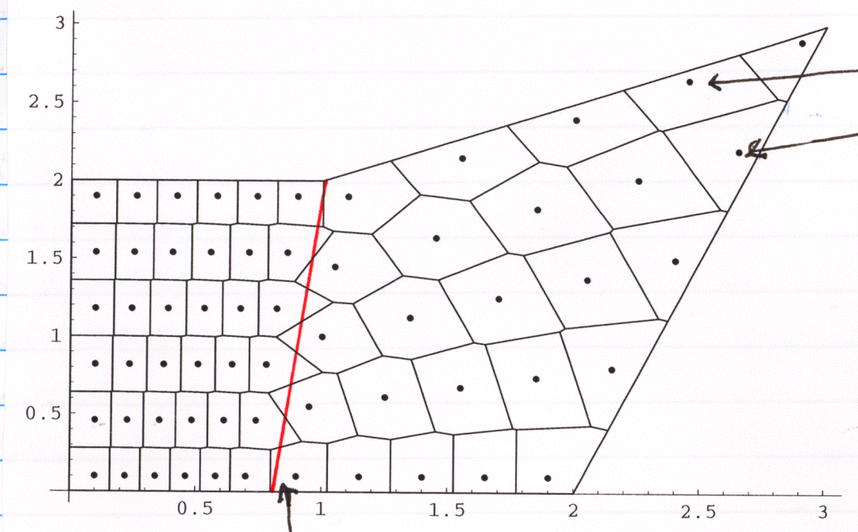
A computer code AMESH, obtained from Ernest Orlando Lawrence National Laboratory (Berkeley, California) will be used to generate the element volumes and connectivity data for the unstructured grid (or mesh).



Section from Foeglon et al (1999), cited on p. 7. Dark numbers are X-Z coordinates in meters. Red numbers are block units, i.e., strata blocks that will be discretized as continuous rock units.

AMESH requires the user to specify the nodal coordinates (cell locations or element locations) for the grid. ~~See ex~~ An example is shown below.

GW 2/10/99



User-specified nodal locations shown as points.

AMESH generates the cell volumes and connectivity data.

Nodes need to be placed close to interfaces to avoid a jagged interface like this one.

Tasks to be accomplished in order to use AMESH to generate unstructured grid for METRA

(1) Node placement code
(2) Code to translate AMESH output into METRA input.
(3) Code to plot mesh using AMESH output.

Pages 10-12 entered by GW 2/10/99

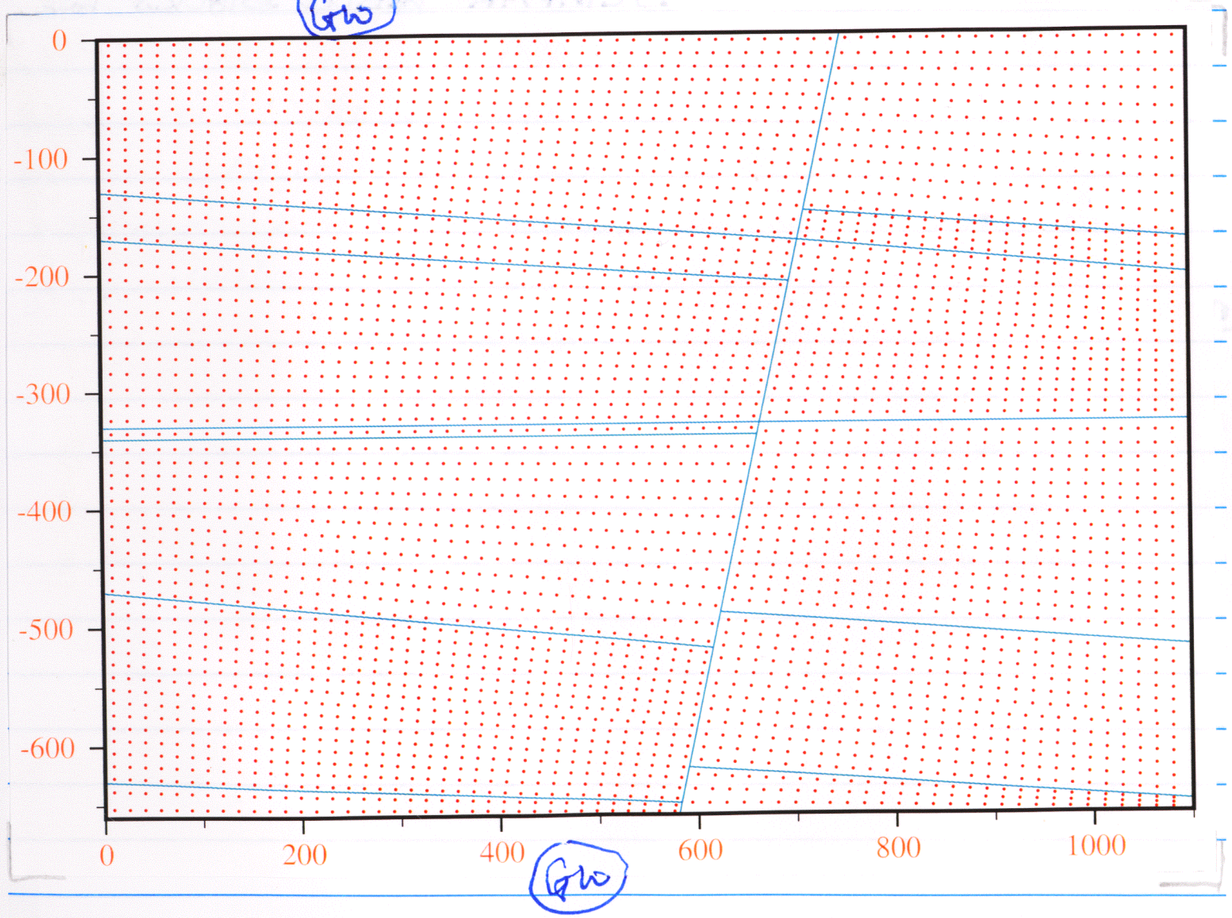
March ~~25~~ 1999 GW

Pages 13-25 entered by GW 3/25/99

The following codes have been developed to handle the tasks listed on p. 12.

(1) BlkToNodes Developes nodal placement. (example shown below for Section on p. 11)

GW



GW

(2) The output from code BlkToNodes is passed to AMESH to obtain element

volumes and connectivity.

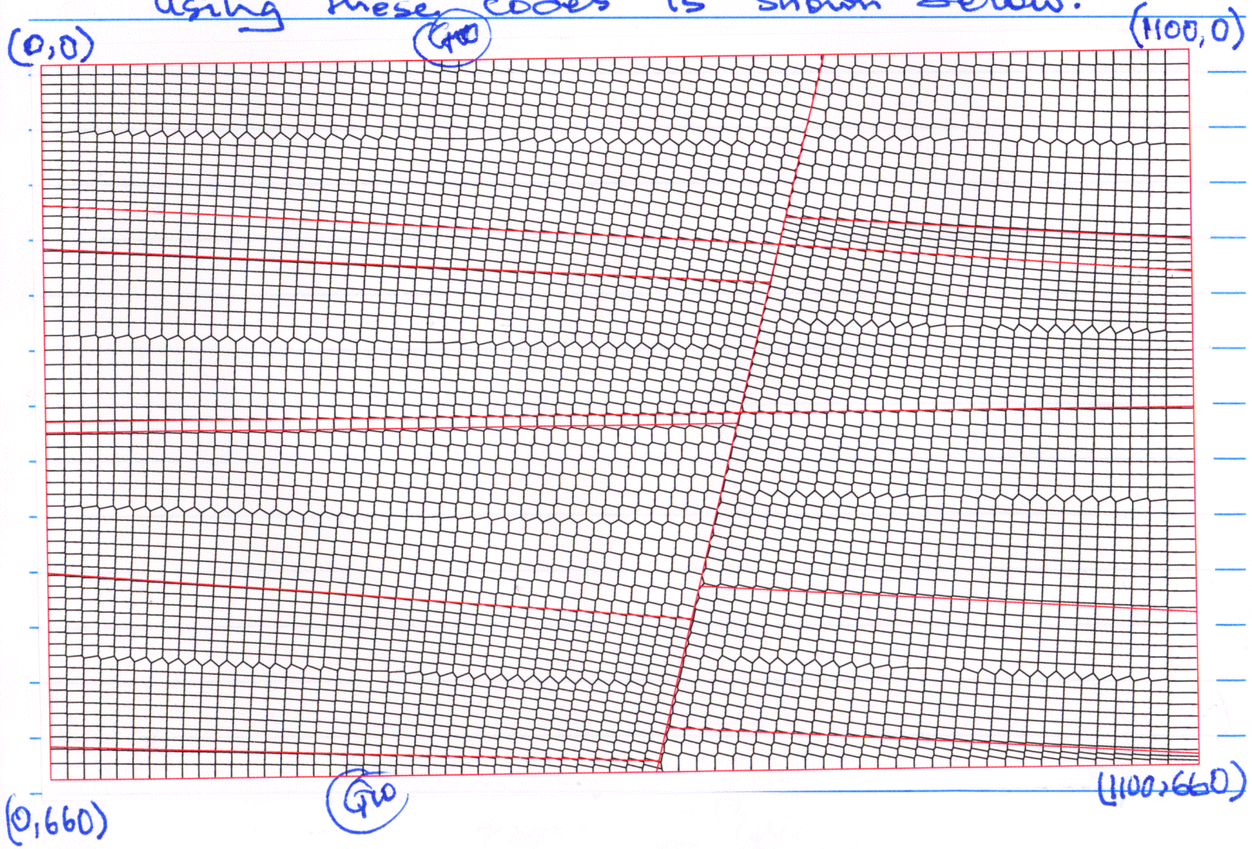
(3) AMESH output is processed using

MFLOMESH — To rewrite connectivity data to METRA format

BLKTOPROPS — To prepare PHIK and DCMPAR input blocks for METRA

BCON — To prepare BCON input block for METRA

A typical mesh (unstructured grid) developed using these codes is shown below.




```
/*****************************************************************************
blkToNodes

This code is part of an evolving preprocessor for MULTIFLOW. This module
generates nodal points for a domain and passes the nodal-point description
to the code AMESH through an input file. The nodal-point information is
also placed in a series of files to enable plotting with an x-y plotter such
as XPLOT.

The current version assumes that the problem domain consists of a series of
gently dipping strata that may be offset by one or more faults. The geometry
of each stratum is described using one or more blocks, each of which is
bounded either by faults or by a section of the domain boundary. The user
provides the geometry of the series of blocks through an input file

The format for the input file is as follows. A line that starts with the word
"Block" or "block" signals the start of input data for a rock-layer block

Comment lines                    <- (as many as needed)

Block                            <- Start of input data for a block
blkIndex  nRow nTopCol nBasCol   <- Node specification variables
x1 y1                            <- Corner coordinates for block, starting
x2 y2                            <- at the top-right corner and proceeding
x3 y3                            <- in counter-clockwise order through all
x4 y4                            <- four corners
beF1 beF2 beF3 beF4              <- Length fraction of near-boundary cells
                                 <- starting with the top boundary and
                                 <- proceeding in counter-clockwise order
                                 <- through all four boundaries

Comment lines may be placed between input lines, but the input data must be
given in the order specified.

Author:                          G. I. Ofoegbu
Date:                            February 24 1995
System:                          C++ Compiler
                                 (Code developed using Borland C++Builder 3,
*****************************************************************************/

#include <stdio.h>
#include <iostream.h>
#include <iomanip.h>
#include <fstream.h>
#include <strstrea.h>
#include <conio.h>
#include <stdlib.h>

#pragma hdrstop
#include <condefs.h>

char* outFile = "tm4Nodes.out";
char* inFile = "tm4Blk.inp";
char* modelName = "tm4";

char* messageBuffer;
void DumpAndQuit();

int ERROREND = 1;
int NORMALEND = 0;
const float zCoordinate = 0.0;
const float elemThickness = 1.0;

struct FBLayer{
  int nRow;
  int nTopCol;
  int nBasCol;
  int blkIndex;
  float xb[4];
  float yb[4];
  float beFrac[4];
};

int GetBlockData(FBLayer* theBlock, istream& Fin);
int NodeGen(FBLayer* theBlock, ostream& Fout, int& nodeIndex);
void BElemCoord(float* xBnd, float* yBnd, float a, float b, float& x,
  float& y);

//---------------------------------------------------------------------------
#pragma argsused
int main(int argc, char **argv)
{

// Allocate memory for possible error messages

  messageBuffer = new char [151];
  if (!messageBuffer){
    cerr << "Memory allocation error for messageBuffer\n";
    cerr << "Press any key to end: ";
    getch();
    return (1);
  }

// Open input file
  ifstream Fin(inFile);
  if (!Fin){
    sprintf(messageBuffer,"Unable to open file %s",inFile);
    DumpAndQuit();
  }

// Open output file (AMESH input file)
  ofstream Fout(outFile);
  if (!Fout){
    sprintf(messageBuffer,"Unable to open file %s",outFile);
    DumpAndQuit();
  }
  Fout << "locat" << endl;

// Read input data one block at a time.
// For each block read, invoke function NodeGen to generate nodes for the
// block and write the node data and block geometry to files

  int nodeIndex = 0;
  char buf[100];
  size_t nBuf = sizeof(buf);
  while (!Fin.eof()){
    Fin.getline(buf,nBuf);
    if (!strncmp(buf,"Block",5) || !strncmp(buf,"block",5)){
      FBLayer* theBlock = new FBLayer;
      if (!theBlock){
        sprintf(messageBuffer,"Memory allocation error on variable theBlock");
        DumpAndQuit();
      }
      if (GetBlockData(theBlock,Fin) == ERROREND)
        DumpAndQuit();
      if (NodeGen(theBlock,Fout,nodeIndex) == ERROREND)
        DumpAndQuit();
      delete theBlock;
    }
  }

// Done

  delete[] messageBuffer;
  cout << "Done ... Press any key to end: ";
  getch();
  return 0;
}

int GetBlockData(FBLayer* theBlock, istream& Fin)
```



```
{
  char buf[100];
  size_t nBuf = sizeof(buf);
  int blkIndex,nRow,nTopCol,nBasCol;
  float x,y;
  float f0,f1,f2,f3;

  int numsFound = 0;
  int coordsFound = 0;
  int fracsFound = 0;

  while (!Fin.eof()){
    Fin.getline(buf,nBuf);
    istrstream inpLine(buf,nBuf);
    if (!numsFound){
      inpLine >> blkIndex >> nRow >> nTopCol >> nBasCol;
      if (inpLine.good()){
        theBlock->blkIndex = blkIndex;
        theBlock->nRow = nRow;
        theBlock->nTopCol = nTopCol;
        theBlock->nBasCol = nBasCol;
        numsFound = 1;
      }
    }
    else if (coordsFound < 4){
      inpLine >> x >> y;
      if (inpLine.good()){
        theBlock->xb[coordsFound] = x;
        theBlock->yb[coordsFound] = y;
        coordsFound++;
      }
    }
    else if (!fracsFound){
      inpLine >> f0 >> f1 >> f2 >> f3;
      if (inpLine.good()){
        theBlock->beFrac[0] = f0;
        theBlock->beFrac[1] = f1;
        theBlock->beFrac[2] = f2;
        theBlock->beFrac[3] = f3;
        fracsFound = 1;
      }
    }
    if (fracsFound) break;
  }
  if (!numsFound){
    sprintf(messageBuffer,"Node specification not found for current block");
    return (ERROREND);
  }
  if (coordsFound < 4){
    sprintf(messageBuffer,"Only %d corner coordinates found for block",
      coordsFound);
    return (ERROREND);
  }
  if (!fracsFound){
    sprintf(messageBuffer,"BE fraction data missing for current block");
    return (ERROREND);
  }
  return (NORMALEND);
}

int NodeGen(FBLayer* theBlock, ostream& Fout, int& nodeIndex)
{
  char* geomFile = "D:\\HydrTherm\\GeomPlot\\bbbbbG99.out";
  char* nodeFile = "D:\\HydrTherm\\GeomPlot\\bbbbbN99.out";

  sprintf(geomFile,"D:\\HydrTherm\\GeomPlot\\%sG%02d.out",
    modelName,theBlock->blkIndex);
  sprintf(nodeFile,"D:\\HydrTherm\\GeomPlot\\%sN%02d.out",
    modelName,theBlock->blkIndex);

  ofstream Fg(geomFile);
  if (!Fg){
    sprintf(messageBuffer,"Unable to open file %s",geomFile);
    return (ERROREND);
  }
  ofstream Fn(nodeFile);
  if (!Fg){
    sprintf(messageBuffer,"Unable to open file %s",nodeFile);
    return (ERROREND);
  }

  int i,j;
  Fg << "Block " << (theBlock->blkIndex) << " corner coordinates" << endl;
  Fg << setw(12) << "xCoord"
     << setw(12) << "yCoord"
     << endl;
  for (i=0; i<4; i++)
    Fg << setiosflags(ios::fixed) << setprecision(2)
       << setw(12) << (theBlock->xb[i])
       << setw(12) << (theBlock->yb[i])
       << endl;
  Fg << setw(12) << theBlock->xb[0]
     << setw(12) << theBlock->yb[0]
     << endl;
  Fg.close();

  Fn << "Block " << (theBlock->blkIndex) << " nodal data" << endl;
  Fn << setw(12) << "NodeIndex"
     << setw(12) << "xCoord"
     << setw(12) << "yCoord"
     << endl;

  int nRow = theBlock->nRow;
  int nTopCol = theBlock->nTopCol;
  int nBasCol = theBlock->nBasCol;
  int midRow = nRow/2;
  int nCol;
  float xTR,xTL,xBL,xBR;
  float yTR,yTL,yBL,yBR;
  float xCoord,yCoord,alpha,beta;
  float xFirst,xLast,yFirst,yLast;
  float ratio;

  float fTop = theBlock->beFrac[0];
  float fLeft = theBlock->beFrac[1];
  float fBase = theBlock->beFrac[2];
  float fRight = theBlock->beFrac[3];

// Compute coordinates of near-corner nodes

// Top right near-corner node

  alpha = 1.0 - 2.0*fRight;
  beta = 1.0 - 2.0*fTop;
  BElemCoord(theBlock->xb,theBlock->yb,alpha,beta,xTR,yTR);

// Top left near-corner node

  alpha = 2.0*fLeft - 1.0;
  beta = 1.0 - 2.0*fTop;
  BElemCoord(theBlock->xb,theBlock->yb,alpha,beta,xTL,yTL);

// Bottom left near-corner node

  alpha = 2.0*fLeft - 1.0;
  beta = 2.0*fBase - 1.0;
  BElemCoord(theBlock->xb,theBlock->yb,alpha,beta,xBL,yBL);

// Bottom right near-corner node
```

Page 3 of 3

blkToNodes.cpp

```
      alpha = 1.0 - 2.0*fRight;
      beta = 2.0*fBase - 1.0;
      BElemCoord(theBlock->xb,theBlock->yb,alpha,beta,xBR,yBR);

    // Generate nodes row by row, starting at the top-left corner

      for (j=1; j<=nRow; j++){
        ratio = 0.0;
        if (nRow > 1) ratio = ((float)(j-1))/((float)(nRow-1));
        xFirst = xTL + ratio*(xBL - xTL);
        xLast = xTR + ratio*(xBR - xTR);
        yFirst = yTL + ratio*(yBL - yTL);
        yLast = yTR + ratio*(yBR - yTR);

        nCol = nTopCol;
        if (j > midRow) nCol = nBasCol;

        for (i=1; i<=nCol; i++){
          ratio = 0.0;
          if (nCol > 1) ratio = ((float)(i-1))/((float)(nCol-1));
          xCoord = xFirst + ratio*(xLast - xFirst);
          yCoord = yFirst + ratio*(yLast - yFirst);

          Fn << setiosflags(ios::fixed) << setprecision(5)
             << setw(12) << (++nodeIndex)
             << setw(12) << xCoord
             << setw(12) << yCoord
             << endl;
          Fout << setiosflags(ios::fixed)
               << setw(4) << nodeIndex
//               << setw(4) << (theBlock->blkIndex)
               << setw(4) << "1"
               << setprecision(5)
               << setw(12) << xCoord
               << setw(12) << yCoord
               << setprecision(1)
               << setw(12) << zCoordinate
               << setw(12) << elemThickness
               << endl;
        }
      }
      return (NORMALEND);
    }

    void BElemCoord(float* xb, float* yb, float a, float b, float& x, float& y)
    {
      float f0,f1,f2,f3;

      f0 = 0.25*(1.0 + a)*(1.0 + b);
      f1 = 0.25*(1.0 - a)*(1.0 + b);
      f2 = 0.25*(1.0 - a)*(1.0 - b);
      f3 = 0.25*(1.0 + a)*(1.0 - b);

      x = f0*xb[0] + f1*xb[1] + f2*xb[2] + f3*xb[3];
      y = f0*yb[0] + f1*yb[1] + f2*yb[2] + f3*yb[3];
    }

    void DumpAndQuit()
    {
      cerr << "\n" << messageBuffer << "\n";
      delete[] messageBuffer;
      cerr << "Press any key to end: ";
      getch();
      exit(1);
    }
```

The input file tm4Blk.inp shown on the right column was used with code blkToNodes (shown on p. 15-16) to generate the node array on p. 13 for the section on p. 11.

D:\HydrTherm\Ppmflow\tm4Blk.inp

```
Example model No. 4
Same as No. 3, but depth-constant discretization within the PTn unit
Adopted from model reported in Ofoegbu et al. (1999)

Block No. 1
1  15  45  43
750.0      0.0
  0.0      0.0
  0.0   -130.0
704.9   -175.0
 0.035  0.015  0.025  0.01

Block No. 2
2  12  21  23
1100.0         0.0
 750.0         0.0
 711.4      -150.0
1100.0      -175.0
 0.035  0.02  0.025  0.04

Block No. 3
3  4   43  43
704.9     -175.0
  0.0     -130.0
  0.0     -170.0
695.9     -210.0
 0.08  0.015  0.08  0.01

Block No. 4
4  4  23  23
1100.0        -175.0
 711.4        -150.0
 704.9        -175.0
1100.0        -205.0
 0.08   0.02   0.12  0.04

Block No. 5
5  12  43  41
695.9         -210.0
  0.0         -170.0
  0.0         -330.0
665.0         -330.0
 0.035 0.015  0.04  0.01

Block No. 6
6  1   41  41
665.0         -330.0
  0.0         -330.0
  0.0         -340.0
662.4         -340.0
 0.5 0.015  0.5  0.01

Block No. 7
7  12  41  39
662.4         -340.0
  0.0         -340.0
  0.0         -470.0
616.1         -520.0
 0.06 0.015 0.025 0.01

Block No. 8
8  15  23  25
1100.0        -205.0
 704.9        -175.0
 665.0        -330.0
1100.0        -330.0
 0.025 0.02 0.05 0.04

Block No. 9
9  15 25  27
1100.0        -330.0
 665.0        -330.0
 623.8        -490.0
1100.0        -520.0
 0.05 0.02 0.064 0.04

Block No. 10
10 15 39  37
616.1         -520.0
  0.0         -470.0
  0.0         -630.0
582.6         -650.0
 0.035 0.015 0.025 0.01

Block No. 11
11 12 27  29
1100.0         -520.0
 623.8         -490.0
 590.3         -620.0
1100.0         -650.0
 0.05 0.02 0.05 0.04

Block No. 12
12 2 37  37
582.6         -650.0
  0.0         -630.0
  0.0         -660.0
580.0         -660.0
 0.25 0.015 0.25 0.01

Block No. 13
13 3 29  29
1100.0        -650.0
 590.3        -620.0
 580.0        -660.0
1100.0        -660.0
 0.15 0.02 0.25 0.04
```


Page 2 of 9

blkToProps.cpp

```
int ERROREND = 1;
int NORMALEND = 0;

enum BndType {ROCKDOMAIN, FAULT, BOUNDARY};

struct FBLayer{
    float fPorosity;
    float fPerm[3];
    float fracSpacm[3];
    float mPorosity;
    float mPerm;
    float fdrLeft;
    float fdrRight;
    float fAreaMod;
    int blkIndex;
    int nRow;
    int nTopCol;
    int nBasCol;
    int istFrac;
    int istMatrix;
    int iTherm;
    BndType top;
    BndType base;
    BndType left;
    BndType right;
};

int GetBlockData(FBLayer* theBlock, istream& Fin);
int UpdateFiles(FBLayer* theBlock, ostream& Fphk, ostream& Fdcm, ostream& Ftc,
  ostream& Flc, ostream& Fbc, ostream& Frc, istream& Fvol, int& nodeIndex);
int PhikAppendTo(FBLayer* theBlock, ostream& Fp, istream& Fvol,
    float fracDenRatio, int firstNode, int lastNode);
int GetElemVolume(istream& Fvol, float& elVol, int node);
void DcmAppendTo(FBLayer* theBlock, ostream& Fp, float fracDenRatio,
    int firstNode, int lastNode);
void BconAppendTo(FBLayer* theBlock, ostream& Fp, float fracDenRatio,
    int firstNode, int lastNode);
void DoProgressReport(int blkIndex);

//-------------------------------------------------------------------
#pragma argsused
int main(int argc, char **argv)
{

    char* phkFile = "tm4phik.out";
    char* dcmFile = "tm4DcmPar.out";
    char* tcFile = "tm4Tbcon.out";
    char* lcFile = "tm4Lbcon.out";
    char* bcFile = "tm4Bbcon.out";
    char* rcFile = "tm4Rbcon.out";
    char* inFile = "tm4Props.inp";
    char* volFile = "tm4Elem.out";

// Allocate memory for possible error messages

    messageBuffer = new char [1511];
    if (!messageBuffer){
      cerr << "Memory allocation error for messageBuffer\n";
      cerr << "Press any key to end: ";
      getch();
      return (1);
    }

// Open input files
    ifstream Fin(inFile);
    if (!Fin){
      sprintf(messageBuffer,"Unable to open file %s",inFile);
      DumpAndQuit();
    }
```

blkToProps.cpp

Page 1 of 9

```
/*****************************************************************
 blkToProps

 This code, as part of an evolving preprocessor for MULTIFLOW,
 generates nodal property input blocks PHIK, DCMPAR, and BCON.
 The BCON input block will be modified using output from a companion
 code MFLOMESH. Nodal volumes required for input block PHIK are
 read from file ELEME (an AMESH) output file.

 The current version assumes that the problem domain consists of a series of
 gently dipping strata that may be offset by one or more faults. Nodal
 locations and ID numbers generated by this module are the same as those
 generated by module BLKTONODES

 The geometry of each stratum is described using one or more blocks,
 each of which is bounded either by faults or by a section of the domain
 boundary. The user provides the boundary-condition and material-property
 definitions for the series of blocks through an input file.

 The format for the input file is as follows. A line that starts with the word
 "Block" or "block" signals the start of input data for a rock-layer block

 Comment lines                    <- (as many as needed)

 Block                            <- Start of input data for a block.
 blkIndex nRow nTopCol nBasCol    <- Block index number and node specifications
                                     User should ensure that these agree
                                     with corresponding values supplied to
                                     module BLKTONODES.
 top=Boundary or top=Domain       <- Boundary condition at top of block.
 left=Boundary or left=Fault
   or left=Domain                 <- Boundary condition at left side of block.
 base=Boundary or base=Domain     <- Boundary condition at base of block.
 right=Boundary or right=Fault
   or right=Domain                <- Boundary condition at right side of block.
 istFrac istMatrix iTherm         <- ID numbers for fracture-hydrologic,
                                     matrix-hydrologic, and thermal property
                                     tables.
 fPor fXPerm fYPerm fZPerm        <- Fracture porosity and x, y, and z
                                     components of fracture permeability.
 mPor mPerm                       <- Matrix porosity and permeability.
 areaMod dxFrac dyFrac dzFrac     <- AREAMOD parameter (see METRA input file)
                                     and fracture spacing in x, y, and z,
                                     directions.
 fdrLeft fdrRight                 <- Fracture-density ratio for fault zone at
                                     the left and right boundaries of block.

 Comment lines may be placed between input lines, but the input data must be
 given in the order specified.

 Author:                          G. I. Ofoegbu
 Date:                            March 18 1999
 System:                          C++ Compiler
                                  (Code developed using Borland C++Builder 3)
*****************************************************************/

#include <stdio.h>
#include <iostream.h>
#include <iomanip.h>
#include <fstream.h>
#include <strstrea.h>
#include <conio.h>
#include <stdlib.h>
#include <math.h>

#pragma hdrstop
#include <condefs.h>

char* messageBuffer;
void DumpAndQuit();
```

```
    ifstream Fvol(volFile);
    if (!Fvol){
      sprintf(messageBuffer,"Unable to open file %s",volFile);
      DumpAndQuit();
    }

  // Read and discard first line in volFile
    Fvol.getline(messageBuffer,100);

  // Open output files
    ofstream Fphk(phkFile);
    if (!Fphk){
      sprintf(messageBuffer,"Unable to open file %s",phkFile);
      DumpAndQuit();
    }
    ofstream Fdcm(dcmFile);
    if (!Fdcm){
      sprintf(messageBuffer,"Unable to open file %s",dcmFile);
      DumpAndQuit();
    }
    ofstream Ftc(tcFile);
    if (!Ftc){
      sprintf(messageBuffer,"Unable to open file %s",tcFile);
      DumpAndQuit();
    }
    ofstream Flc(lcFile);
    if (!Flc){
      sprintf(messageBuffer,"Unable to open file %s",lcFile);
      DumpAndQuit();
    }
    ofstream Fbc(bcFile);
    if (!Fbc){
      sprintf(messageBuffer,"Unable to open file %s",bcFile);
      DumpAndQuit();
    }
    ofstream Frc(rcFile);
    if (!Frc){
      sprintf(messageBuffer,"Unable to open file %s",rcFile);
      DumpAndQuit();
    }
    Fdcm << "DCMParameters" << endl;
    Fdcm << ":  M1   M2   INC    SIGMAF   AREAMOD   XLM   YLM   ZLM"
         << endl;
    Ftc << "BCON" << endl;
    Ftc << ":  M1   M2 INC  fPorosity       mPerm" << endl;
    Flc << "BCON" << endl;
    Flc << ":  M1   M2 INC  fPorosity       mPerm" << endl;
    Fbc << "BCON" << endl;
    Fbc << ":  M1   M2 INC  fPorosity       mPerm" << endl;
    Frc << "BCON" << endl;
    Frc << ":  M1   M2 INC  fPorosity       mPerm" << endl;

  // Read data and write output files one block at a time

    char buf[100];
    size_t nBuf = sizeof(buf);
    int nodeIndex = 0;
    while (!Fin.eof()){
      Fin.getline(buf,nBuf);
      if (!strncmp(buf,"Block",5) || !strncmp(buf,"block",5)){
        FBLayer* theBlock = new FBLayer;
        if (!theBlock){
          sprintf(messageBuffer,"Memory allocation error on variable theBlock");
          DumpAndQuit();
        }
        if (GetBlockData(theBlock,Fin) == ERROREND){
          delete theBlock;
          DumpAndQuit();
        }
        if (UpdateFiles(theBlock,Fphk,Fdcm,Ftc,Flc,Fbc,Frc,Fvol,
```



```
                        nodeIndex)==ERROREND){
          delete theBlock;
          DumpAndQuit();
        }
        DoProgressReport(theBlock->blkIndex);
        delete theBlock;
      }
    }

  // Done

    delete[] messageBuffer;
    cout << "Done ... Press any key to end: ";
    getch();
    return 0;
}

int GetBlockData(FBLayer* theBlock, istream& Fin)
{
    char buf[100];
    size_t nBuf = sizeof(buf);
    int linesFound = 0;

    while (!Fin.eof()){
      Fin.getline(buf,nBuf);
      istrstream inpLine(buf,nBuf);
      if (linesFound < 1){              // Input line 1
        int blkIndex,nRow,nTopCol,nBasCol;
        inpLine >> blkIndex >> nRow >> nTopCol >> nBasCol;
        if (inpLine.good()){
          theBlock->blkIndex = blkIndex;
          theBlock->nRow = nRow;
          theBlock->nTopCol = nTopCol;
          theBlock->nBasCol = nBasCol;
          linesFound++;
        }
      }
      else if (linesFound < 2){             // Input line 2
        if (!strncmp(buf,"top=",4))
          if ((buf[4] == 'B') || (buf[4] == 'b')){
            theBlock->top = BOUNDARY;
            linesFound++;
          }
          else if ((buf[4] == 'D') || (buf[4] == 'd')){
            theBlock->top = ROCKDOMAIN;
            linesFound++;
          }
          else{
            sprintf(messageBuffer,"Invalid input line: %s",buf);
            return (ERROREND);
          }
      }
      else if (linesFound < 3){             // Input line 3
        if (!strncmp(buf,"left=",5))
          if ((buf[5] == 'B') || (buf[5] == 'b')){
            theBlock->left = BOUNDARY;
            linesFound++;
          }
          else if ((buf[5] == 'F') || (buf[5] == 'f')){
            theBlock->left = FAULT;
            linesFound++;
          }
          else if ((buf[5] == 'D') || (buf[5] == 'd')){
            theBlock->left = ROCKDOMAIN;
            linesFound++;
          }
          else{
            sprintf(messageBuffer,"Invalid input line: %s",buf);
            return (ERROREND);
          }
```



```
      }
      else if (linesFound < 4){             // Input line 4
        if (!strncmp(buf,"base=",5))
          if ((buf[5] == 'B') || (buf[5] == 'b')){
            theBlock->base = BOUNDARY;
            linesFound++;
          }
          else if ((buf[5] == 'D') || (buf[5] == 'd')){
            theBlock->base = ROCKDOMAIN;
            linesFound++;
          }
          else{
            sprintf(messageBuffer,"Invalid input line: %s",buf);
            return (ERROREND);
          }
      }
      else if (linesFound < 5){             // Input line 5
        if (!strncmp(buf,"right=",6))
          if ((buf[6] == 'B') || (buf[6] == 'b')){
            theBlock->right = BOUNDARY;
            linesFound++;
          }
          else if ((buf[6] == 'F') || (buf[6] == 'f')){
            theBlock->right = FAULT;
            linesFound++;
          }
          else if ((buf[6] == 'D') || (buf[6] == 'd')){
            theBlock->right = ROCKDOMAIN;
            linesFound++;
          }
          else{
            sprintf(messageBuffer,"Invalid input line: %s",buf);
            return (ERROREND);
          }
      }
      else if (linesFound < 6){             // Input line 6
        int istFrac,istMatrix,iTherm;
        inpLine >> istFrac >> istMatrix >> iTherm;
        if (inpLine.good()){
          theBlock->istFrac = istFrac;
          theBlock->istMatrix = istMatrix;
          theBlock->iTherm = iTherm;
          linesFound++;
        }
      }
      else if (linesFound < 7){             // Input line 7
        float fPor,fxPerm,fyPerm,fzPerm;
        inpLine >> fPor >> fxPerm >> fyPerm >> fzPerm;
        if (inpLine.good()){
          theBlock->fPorosity = fPor;
          theBlock->fPerm[0] = fxPerm;
          theBlock->fPerm[1] = fyPerm;
          theBlock->fPerm[2] = fzPerm;
          linesFound++;
        }
      }
      else if (linesFound < 8){             // Input line 8
        float mPor,mPerm;
        inpLine >> mPor >> mPerm;
        if (inpLine.good()){
          theBlock->mPorosity = mPor;
          theBlock->mPerm = mPerm;
          linesFound++;
        }
      }
      else if (linesFound < 9){             // Input line 9
        float fAreaMod,dxFrac,dyFrac,dzFrac;
        inpLine >> fAreaMod >> dxFrac >> dyFrac >> dzFrac;
        if (inpLine.good()){
          theBlock->fAreaMod = fAreaMod;
```



```
          theBlock->fracSpacn[0] = dxFrac;
          theBlock->fracSpacn[1] = dyFrac;
          theBlock->fracSpacn[2] = dzFrac;
          linesFound++;
        }
      }
      else if (linesFound < 10){            // Input line 10
        float fdrLeft,fdrRight;
        inpLine >> fdrLeft >> fdrRight;
        if (inpLine.good()){
          if ((fdrLeft < 0.01) || (fdrRight < 0.01)){
            sprintf(messageBuffer,
              "Invalid values: fdrLeft=%.3f, fdrRight=%.3f: block No. %d",
              fdrLeft,fdrRight,theBlock->blkIndex);
            return (ERROREND);
          }
          theBlock->fdrLeft = fdrLeft;
          theBlock->fdrRight = fdrRight;
          linesFound++;
        }
      }
      if (linesFound == 10) break;
    }
    if (linesFound < 1){
      sprintf(messageBuffer,
        "Only %d of 10 input lines found for current block", linesFound);
      return (ERROREND);
    }
    else if (linesFound < 10){
      sprintf(messageBuffer,
        "Only %d of 10 input lines found: Block No. %d",
          linesFound,theBlock->blkIndex);
      return (ERROREND);
    }
    return (NORMALEND);
}

int UpdateFiles(FBLayer* theBlock, ostream& Fphk, ostream& Fdcm, ostream& Ftc,
  ostream& Flc, ostream& Fbc, ostream& Frc, istream& Fvol, int& nodeIndex)
{
    if (
      ((theBlock->fdrLeft)*(theBlock->fPorosity) > 0.99) ||
      ((theBlock->fdrRight)*(theBlock->fPorosity) > 0.99)
      ){
        sprintf(messageBuffer,
          "Faultzone fracture porosity out of range: Block No. %d",
          theBlock->blkIndex);
        return (ERROREND);
    }

    int nRow = theBlock->nRow;
    int nTopCol = theBlock->nTopCol;
    int nBasCol = theBlock->nBasCol;
    int midRow = nRow/2;

    for (int rowNum = 1; rowNum <= nRow; rowNum++){
      int m1 = nodeIndex + 1;
      int m2;
      if (rowNum <= midRow) nodeIndex += nTopCol;
      else nodeIndex += nBasCol;
      m2 = nodeIndex;

  //  Append to top-boundary list if necessary

      if ((rowNum == 1) && ((theBlock->top) == BOUNDARY)){
        int mm1 = m1;
        if ((theBlock->left) == FAULT){
          BconAppendTo(theBlock,Ftc,theBlock->fdrLeft,m1,m1);
          mm1 = m1 + 1;
```

blkToProps.cpp

Page 7 of 9

```
        }
        if ((theBlock->right) == FAULT){
          BconAppendTo(theBlock,Ftc,1.0,mm1,m2-1);
          BconAppendTo(theBlock,Ftc,theBlock->fdrRight,m2,m2);
        }
        else BconAppendTo(theBlock,Ftc,1.0,mm1,m2);
      }

    //  Append to base-boundary list if necessary

      if ((rowNum == nRow) && ((theBlock->base) == BOUNDARY)){
        int mm1 = m1;
        if ((theBlock->left) == FAULT){
          BconAppendTo(theBlock,Fbc,theBlock->fdrLeft,m1,m1);
          mm1 = m1 + 1;
        }
        if ((theBlock->right) == FAULT){
          BconAppendTo(theBlock,Fbc,1.0,mm1,m2-1);
          BconAppendTo(theBlock,Fbc,theBlock->fdrRight,m2,m2);
        }
        else BconAppendTo(theBlock,Fbc,1.0,mm1,m2);
      }

    //  Append to left-boundary list if necessary

      if ((theBlock->left) == BOUNDARY)
        BconAppendTo(theBlock,Flc,1.0,m1,m1);

    //  If left-most element is in fault zone, update phik and dcmpar list
    //  with appropriate fault-zone modifier;
    //  then advance list to next element

      else if ((theBlock->left) == FAULT){
        if (PhikAppendTo(theBlock,Fphk,Fvol,theBlock->fdrLeft,m1,m1) == ERROREND)
          return (ERROREND);
        DcmAppendTo(theBlock,Fdcm,theBlock->fdrLeft,m1,m1);
        m1++;
      }

    //  Action taken for the rest of the elements in the current row depend on
    //  the location of the right-most element, on a boundary, fault zone, or
    //  inside the rock domain

      if ((theBlock->right) == BOUNDARY){ // right-most element on boundary
        if (PhikAppendTo(theBlock,Fphk,Fvol,1.0,m1,m2) == ERROREND)
          return (ERROREND);
        BconAppendTo(theBlock,Frc,1.0,m2,m2);
        DcmAppendTo(theBlock,Fdcm,1.0,m1,m2);
      }

      else if ((theBlock->right) == FAULT){ // right-most element in fault zone
        if (
          (PhikAppendTo(theBlock,Fphk,Fvol,1.0,m1,m2-1) == ERROREND) ||
          (PhikAppendTo(theBlock,Fphk,Fvol,theBlock->fdrRight,m2,m2) == ERROREND)
        ) return (ERROREND);
        DcmAppendTo(theBlock,Fdcm,1.0,m1,m2-1);
        DcmAppendTo(theBlock,Fdcm,theBlock->fdrRight,m2,m2);
      }

      else{                    // right-most element inside rock domain
        if (PhikAppendTo(theBlock,Fphk,Fvol,1.0,m1,m2) == ERROREND)
          return (ERROREND);
        DcmAppendTo(theBlock,Fdcm,1.0,m1,m2);
      }
    }
    return (NORMALEND);
  }
  void BconAppendTo(FBLayer* theBlock, ostream& Fp, float fdr,
    int firstNode, int lastNode)
  {
```

blkToProps.cpp

Page 8 of 9

```
    Fp << setw(5) << firstNode
       << setw(5) << lastNode
       << setw(3) << "1"
       << setiosflags(ios::scientific) << setprecision(3)
       << setw(12) << (theBlock->fPorosity)*fdr
       << setw(12) << (theBlock->mPerm)
       << resetiosflags(ios::scientific)
       << endl;
  }
  void DcmAppendTo(FBLayer* theBlock, ostream& Fp, float fdr,
    int firstNode, int lastNode)
  {
    float fsigma = (theBlock->fPorosity)*fdr;
    float ratio = pow(fdr,1.0/3.0);
    float xlm = (theBlock->fracSpacn[0])/ratio;
    float ylm = (theBlock->fracSpacn[1])/ratio;
    float zlm = (theBlock->fracSpacn[2])/ratio;

    Fp << setw(5) << firstNode
       << setw(5) << lastNode
       << setw(3) << "1"
       << setiosflags(ios::scientific) << setprecision(3)
       << setw(12) << fsigma
       << setw(12) << (theBlock->fAreaMod)
       << resetiosflags(ios::scientific)
       << setiosflags(ios::fixed) << setprecision(4)
       << setw(10) << xlm
       << setw(10) << ylm
       << setw(10) << zlm
       << resetiosflags(ios::fixed)
       << endl;
  }
  int PhikAppendTo(FBLayer* theBlock, ostream& Fp, istream& Fvol,
    float fdr, int firstNode, int lastNode)
  {
    float elVol;

    for (int node = firstNode; node <= lastNode; node++){
      if (GetElemVolume(Fvol,elVol,node) == ERROREND)
        return (ERROREND);
      float fPoros = (theBlock->fPorosity)*fdr;
      Fp << setw(5) << node
         << setw(5) << node
         << setw(3) << "1"
         << setw(5) << (theBlock->istFrac)
         << setw(5) << (theBlock->iTherm)
         << setiosflags(ios::scientific) << setprecision(3)
         << setw(12) << elVol
         << setw(5) << "1.0"
         << setw(12) << (theBlock->fPerm[0])/fPoros
         << setw(12) << (theBlock->fPerm[1])/fPoros
         << setw(12) << (theBlock->fPerm[2])/fPoros
         << resetiosflags(ios::scientific)
         << setiosflags(ios::fixed) << setprecision(3)
         << setw(10) << (theBlock->mPorosity)
         << resetiosflags(ios::fixed) << setiosflags(ios::scientific)
         << setw(12) << (theBlock->mPerm)
         << setw(5) << (theBlock->istMatrix)
         << setw(5) << (theBlock->iTherm)
         << endl;
    }
    return (NORMALEND);
  }
  int GetElemVolume(istream& Fvol, float& elVol, int node)
  {
    char buf[100];
    size_t nbuf = sizeof(buf);
    int elNum;

    Fvol.getline(buf,nbuf);
```


blkToProps.cpp

Page 9 of 9

```
    if (Fvol.bad()){
      sprintf(messageBuffer,"Error encountered in volume file for node %d",
        node);
      return (ERROREND);
    }
    istrstream nLine(buf,nbuf);
    nLine >> elNum;
    if (!nLine.good()){
      sprintf(messageBuffer,"Invalid input found for node %d in volume file",
        node);
      return (ERROREND);
    }
    if (elNum != node){
      sprintf(messageBuffer,"Node %d found in volume file instead of %d",
        elNum,node);
      return (ERROREND);
    }
    istrstream vLine(buf+20,nbuf-20);
    vLine >> elVol;
    if (!vLine.good()){
      sprintf(messageBuffer,"Invalid input found for volume of node %d",
        node);
      return (ERROREND);
    }
    return (NORMALEND);
  }
  void DumpAndQuit()
  {
    cerr << "\n" << messageBuffer << "\n";
    delete[] messageBuffer;
    cerr << "Press any key to end: ";
    getch();
    exit(1);
  }
  void DoProgressReport(int blkIndex)
  {
    cout << "Working ... Processed block number "
         << blkIndex
         << endl;
  }
```

The input file tm4Props.inp, which is shown on the second and third columns and below was used with the code blkToProps on p.17-21 to generate METRA input files for the example mesh on p.13 & 14.

D:\HydrTherm\Ppmflow\tm4Props.inp

Page 1 of 3

```
Nodal property specifications for Example model No. 4
Adopted from model reported in Ofoegbu et al. (1999)

Block No. 1
1  15  45  43
top=Boundary
left=Boundary
base=Domain
right=Fault
2  1  1
2.33e-4  6.03e-12  6.03e-12  2.29e-11
0.066    5.40e-18
4.90e-4  0.5       0.5       0.5
1.0      1.0

Block No. 2
2  12  21  23
top=Boundary
left=Fault
base=Domain
right=Boundary
2  1  1
2.33e-4  6.03e-12  6.03e-12  2.29e-11
0.066    5.40e-18
4.90e-4  0.5       0.5       0.5
1.0      1.0

Block No. 3
3  4   43  43
top=Domain
left=Boundary
base=Domain
right=Fault
4  3  2
6.94e-5   6.17e-14   6.17e-14   6.17e-14
0.469    8.80e-14
0.479    0.5   0.5   0.5
1.0      1.0

Block No. 4
4  4  23  23
top=Domain
left=Fault
base=Domain
right=Boundary
4  3  2
6.94e-5   6.17e-14   6.17e-14   6.17e-14
0.469    8.80e-14
0.479    0.5   0.5   0.5
1.0      1.0

Block No. 5
5  12  43  41
top=Domain
left=Boundary
base=Domain
right=Fault
6  5  3
1.24e-4  4.27e-13  4.27e-13  6.76e-12
0.089    6.75e-18
1.55e-4  0.5   0.5   0.5
1.0  1.0

Block No. 6
6  1   41  41
top=Domain
left=Boundary
base=Domain
right=Fault
6  5  3
```

D:\HydrTherm\Ppmflow\tm4Props.inp

Page 2 of 3

```
1.24e-4  4.27e-13  4.27e-13  6.76e-12
0.089    6.75e-18
1.55e-4  0.5   0.5   0.5
1.0  1.0

Block No. 7
7  12  41  39
top=Domain
left=Boundary
base=Domain
right=Fault
6  5  3
1.24e-4  4.27e-13  4.27e-13  6.76e-12
0.089    6.75e-18
1.55e-4  0.5   0.5   0.5
1.0  1.0

Block No. 8
8  15  23  25
top=Domain
left=Fault
base=Domain
right=Boundary
6  5  3
1.24e-4  4.27e-13  4.27e-13  6.76e-12
0.089    6.75e-18
1.55e-4  0.5   0.5   0.5
1.0  1.0

Block No. 9
9  15 25  27
top=Domain
left=Fault
base=Domain
right=Boundary
6  5  3
1.24e-4  4.27e-13  4.27e-13  6.76e-12
0.089    6.75e-18
1.55e-4  0.5   0.5   0.5
1.0  1.0

Block No. 10
10 15 39  37
top=Domain
left=Boundary
base=Domain
right=Fault
8  7  4
7.14e-5  2.88e-13  2.88e-13  2.88e-13
0.321    5.50e-14
0.489    0.5   0.5   0.5
1.0  1.0

Block No. 11
11 12 27  29
top=Domain
left=Fault
base=Domain
right=Boundary
8  7  4
7.14e-5  2.88e-13  2.88e-13  2.88e-13
0.321    5.50e-14
0.489    0.5   0.5   0.5
1.0  1.0

Block No. 12
12 2 37  37
top=Domain
left=Boundary
base=Boundary
```

D:\HydrTherm\Ppmflow\tm4Props.inp

Page 3 of 3

```
right=Fault
10  9  5
7.14e-5  6.92e-13  6.92e-13  6.92e-13
0.274  1.24e-15
5.13e-4  0.5  0.5  0.5
1.0  1.0

Block No. 13
13 3 29  29
top=Domain
left=Fault
base=Boundary
right=Boundary
10  9  5
7.14e-5  6.92e-13  6.92e-13  6.92e-13
0.274  1.24e-15
5.13e-4  0.5  0.5  0.5
1.0  1.0
```

Page 1 of 8

```
/*****************************************************************************
Bcon

This code, as part of an evolving preprocessor for MULTIFLOW,
generates the boundary condition input block BCON using information
previously generated by two companion codes: MFLOMESH and BLKTOPROPS.
The following METRA boundary-condition types are assumed:

Top boundary            Type 5  (Specified liquid flux and temperature)
Base boundary           Type 1  (Specified temperature and pressure)
Left boundary           Zero flux
Right boundary          Zero flux

Author:                    G. I. Ofoegbu
Date:                      March 18 1999
System:                    C++ Compiler
                           (Code developed using Borland C++Builder 3)
*****************************************************************************/

#include <stdio.h>
#include <iostream.h>
#include <iomanip.h>
#include <fstream.h>
#include <strstrea.h>
#include <conio.h>
#include <stdlib.h>
#include <math.h>

#pragma hdrstop
#include <condefs.h>


char* messageBuffer;
void DumpAndQuit();

int ERROREND = 1;
int NORMALEND = 0;

char* mf2BcFile = "mf2Bcon.tm4";
char* nodeDumpFile = "testBcon.out";
char* outFile = "tm4All.bc";
char* tcFile = "tm4Tbcon.out";
//char* lcFile = "tm4Lbcon.out";
char* bcFile = "tm4Bbcon.out";
//char* rcFile = "tm4Rbcon.out";

const float topInfilRate = 10.0;     // Steady infiltration rate (mm/yr)
const float topAtmos = 93361.0;      // Atmospheric pressure at top (Pa)
const float basAtmos = 101325.0;     // Atmospheric pressure at base (Pa)
const float topTemp = 20.0;          // Temperature at top (Celcius)
const float basTemp = 30.0;          // Temperature at base (Celcius)
const float denWater = 998.2;        // Density of water (kg/m/m/m) at 20 C
const float visWater = 1.002e-03;    // Viscosity ------------------------
const float gravity = 9.8;           // Gravitational acceleration (m/s/s)
const float secPerYr = 365.25*24.0*60.0*60.0; // Seconds per yr
const float meter = 1000.0;          // mm per m

struct BndNode{
  int flowDir;
  int index;
  float bDist;
  float bArea;
  float bDelh;
  BndNode* next;
  BndNode(int dir, int nd, float r, float a, float dh){
    flowDir = dir;
    index = nd;
    bDist = r;
    bArea = a;
    bDelh = dh;
```


Page 2 of 8

```
    next = 0;
  };
};

int GetAllBndNodes(BndNode** firstNode, int& totNumBc);
int PrintNodes(BndNode* firstNode);
int TopBconOut(BndNode* firstNode);
//int LeftBconOut(BndNode* firstNode);
int BaseBconOut(BndNode* firstNode);
//int RightBconOut(BndNode* firstNode);
int GetBconneData(BndNode* firstNode, int node, int flowDir,
  float& bDist,float& bArea,float& bDelh);
void DeleteNodes(BndNode** firstNode);
//---------------------------------------------------------------------------
#pragma argsused
int main(int argc, char **argv)
{

// Allocate memory for possible error messages
  messageBuffer = new char [151];
  if (!messageBuffer){
    cerr << "Memory allocation error for messageBuffer\n";
    cerr << "Press any key to end: ";
    getch();
    return (1);
  }

// Initialize output file (erase file if it already exists)
  ofstream Fout(outFile);
  if (!Fout){
    sprintf(messageBuffer,"Unable to open file %s in append mode",outFile);
    DumpAndQuit();
  }

  BndNode* firstNode = 0;
  int totNumBc = 0;
  if (GetAllBndNodes(&firstNode,totNumBc) == ERROREND){
    DeleteNodes(&firstNode);
    DumpAndQuit();
  }
  Fout << ": Total number of BCON definitions = "
       << totNumBc
       << endl;
  Fout.close();

  if (TopBconOut(firstNode) == ERROREND){
    DeleteNodes(&firstNode);
    DumpAndQuit();
  }
/*
  if (LeftBconOut(firstNode) == ERROREND){
    DeleteNodes(&firstNode);
    DumpAndQuit();
  }
*/
  if (BaseBconOut(firstNode) == ERROREND){
    DeleteNodes(&firstNode);
    DumpAndQuit();
  }
/*
  if (RightBconOut(firstNode) == ERROREND){
    DeleteNodes(&firstNode);
    DumpAndQuit();
  }
*/

// Done

  if (PrintNodes(firstNode) == ERROREND){
    DeleteNodes(&firstNode);
```


Page 3 of 8

```
    DumpAndQuit();
  }

  DeleteNodes(&firstNode);
  delete[] messageBuffer;
  cout << "Done ... Press any key to end: ";
  getch();
  return 0;
}
int GetAllBndNodes(BndNode** firstNode, int& numNodes)
{
  ifstream Fin(mf2BcFile);
  if (!Fin){
    sprintf(messageBuffer,"Unable to open file %s",mf2BcFile);
    return (ERROREND);
  }

  BndNode* preNode = 0;
  *firstNode = preNode;
  int n1,n2,n3;
  numNodes = 0;
  int dir;
  int index;
  float bDist;
  float bArea;
  float bDelh;
  char buf[100];
  size_t nBuf = sizeof(buf);

  while (!Fin.eof()){
    Fin.getline(buf,nBuf);
    istrstream inpLine(buf,nBuf);
    inpLine >> n1 >> dir >> index >> n2 >> n3 >> bDist >> bArea >> bDelh;
    if (inpLine.good() && (dir == 3)){
      numNodes++;
      BndNode* curNode = new BndNode(dir,index,bDist,bArea,bDelh);
      if (!curNode){
        sprintf(messageBuffer,"Memory allocation error for %d(th) node",
          numNodes);
        return (ERROREND);
      }
      if (numNodes == 1) *firstNode = curNode;
      else preNode->next = curNode;
      preNode = curNode;
    }
  }
  return (NORMALEND);
}
int TopBconOut(BndNode* firstNode)
{
  ofstream Fout(outFile,ios::app);
  if (!Fout){
    sprintf(messageBuffer,"Unable to open file %s in append mode",outFile);
    return (ERROREND);
  }
  ifstream Fin(tcFile);
  if (!Fin){
    sprintf(messageBuffer,"Unable to open file %s",tcFile);
    return (ERROREND);
  }
  float bDist;
  float bArea;
  float bDelh;
  float fPorosity;
  float mPerm;
  float qTotal = topInfilRate*denWater/(secPerYr*meter);
  int flowDir = 3;
  int bcType = 5;
  int m1,m2,inc;
  char buf[100];
```


Page 4 of 8

```
  size_t nBuf = sizeof(buf);

  Fout << ": Boundary condition type 5 at top" << endl;
  while (!Fin.eof()){
    Fin.getline(buf,nBuf);
    istrstream inpLine(buf,nBuf);
    inpLine >> m1 >> m2 >> inc >> fPorosity >> mPerm;
    if (inpLine.good()){
      float mFlux = (denWater*denWater*gravity/visWater)*(0.9*mPerm);
      float fFlux = (qTotal - mFlux)/fPorosity;
      for (int node = m1; node <= m2; node++){
        if (GetBconneData(firstNode,node,flowDir,bDist,bArea,bDelh)
          == ERROREND) return (ERROREND);
        Fout << setw(3) << bcType
             << setw(3) << flowDir
             << setw(5) << node
             << setw(5) << node
             << setw(3) << "1"
             << setiosflags(ios::fixed) << setprecision(3)
             << setw(10) << bDist
             << setw(10) << bArea
             << setw(10) << bDelh
             << resetiosflags(ios::fixed)
             << endl;
        Fout << setw(5) << "0.0"
             << setiosflags(ios::scientific)
             << setw(12) << mFlux
             << resetiosflags(ios::scientific) << setiosflags(ios::fixed)
             << setprecision(1)
             << setw(10) << topAtmos
             << setw(6) << topTemp
             << setw(5) << "0.95"
             << setw(5) << "0.0" << setw(5) << "0.0"
             << resetiosflags(ios::fixed)
             << endl;
        Fout << setw(5) << "0.0"
             << setiosflags(ios::scientific) << setprecision(3)
             << setw(12) << fFlux
             << resetiosflags(ios::scientific) << setiosflags(ios::fixed)
             << setprecision(1)
             << setw(10) << topAtmos
             << setw(6) << topTemp
             << setw(5) << "0.95"
             << setw(5) << "0.0" << setw(5) << "0.0"
             << endl;
        Fout << "/" << endl;
      }
    }
  }
  return (NORMALEND);
}
/*
int LeftBconOut(BndNode* firstNode)
{
  ofstream Fout(outFile,ios::app);
  if (!Fout){
    sprintf(messageBuffer,"Unable to open file %s in append mode",outFile);
    return (ERROREND);
  }
  ifstream Fin(lcFile);
  if (!Fin){
    sprintf(messageBuffer,"Unable to open file %s",lcFile);
    return (ERROREND);
  }
  float bDist;
  float bArea;
  float bDelh;
  float fPorosity;
  float mPerm;
  int flowDir = 1;
```

Bcon.cpp

Page 5 of 8

```
    int bcType = 2;
    int m1,m2,inc;
    char buf[100];
    size_t nBuf = sizeof(buf);

    Fout << ": Boundary condition type 2 on the left side" << endl;
    while (!Fin.eof()){
      Fin.getline(buf,nBuf);
      istrstream inpLine(buf,nBuf);
      inpLine >> m1 >> m2 >> inc >> fPorosity >> mPerm;
      if (inpLine.good()){
        for (int node = m1; node <= m2; node++){
          if (GetBconneData(firstNode,node,flowDir,bDist,bArea,bDelh)
            == ERROREND) return (ERROREND);
          Fout << setw(3) << bcType
               << setw(3) << flowDir
               << setw(5) << node
               << setw(5) << node
               << setw(3) << "1"
               << setiosflags(ios::fixed) << setprecision(3)
               << setw(10) << bDist
               << setw(10) << bArea
               << setw(10) << bDelh
               << resetiosflags(ios::fixed)
               << endl;
          Fout << setw(5) << "0.0"
               << setw(5) << "0.0"
               << setw(5) << "0.0"
               << setw(5) << "0.0"
               << setw(5) << "0.0"
               << setw(5) << "0.0"
               << setw(5) << "0.0"
               << endl;
          Fout << setw(5) << "0.0"
               << setw(5) << "0.0"
               << setw(5) << "0.0"
               << setw(5) << "0.0"
               << setw(5) << "0.0"
               << setw(5) << "0.0"
               << setw(5) << "0.0"
               << endl;
          Fout << "/" << endl;
        }
      }
    }
    return (NORMALEND);
  }
  */
int BaseBconOut(BndNode* firstNode)
{
  ofstream Fout(outFile,ios::app);
  if (!Fout){
    sprintf(messageBuffer,"Unable to open file %s in append mode",outFile);
    return (ERROREND);
  }
  ifstream Fin(bcFile);
  if (!Fin){
    sprintf(messageBuffer,"Unable to open file %s",bcFile);
    return (ERROREND);
  }
  float bDist;
  float bArea;
  float bDelh;
  float fPorosity;
  float mPerm;
  int flowDir = 3;
  int bcType = 1;
  int m1,m2,inc;
  char buf[100];
```

Bcon.cpp

Page 6 of 8


```
  size_t nBuf = sizeof(buf);

  Fout << ": Boundary condition type 1 at base" << endl;
  while (!Fin.eof()){
    Fin.getline(buf,nBuf);
    istrstream inpLine(buf,nBuf);
    inpLine >> m1 >> m2 >> inc >> fPorosity >> mPerm;
    if (inpLine.good()){
      for (int node = m1; node <= m2; node++){
        if (GetBconneData(firstNode,node,flowDir,bDist,bArea,bDelh)
          == ERROREND) return (ERROREND);
        Fout << setw(3) << bcType
             << setw(3) << flowDir
             << setw(5) << node
             << setw(5) << node
             << setw(3) << "1"
             << setiosflags(ios::fixed) << setprecision(3)
             << setw(10) << bDist
             << setw(10) << bArea
             << setw(10) << bDelh
             << resetiosflags(ios::fixed)
             << endl;
        Fout << setw(5) << "0.0"
             << setw(5) << "0.0"
             << setiosflags(ios::fixed)
             << setprecision(1)
             << setw(10) << basAtmos
             << setw(6) << basTemp
             << setw(5) << "0.0"
             << setw(5) << "0.0"
             << setw(5) << "0.0"
             << endl;
        Fout << setw(5) << "0.0"
             << setw(5) << "0.0"
             << setw(10) << basAtmos
             << setw(6) << basTemp
             << setw(5) << "0.0"
             << setw(5) << "0.0"
             << setw(5) << "0.0"
             << endl;
        Fout << "/" << endl;
      }
    }
  }
  return (NORMALEND);
}
/*
int RightBconOut(BndNode* firstNode)
{
  ofstream Fout(outFile,ios::app);
  if (!Fout){
    sprintf(messageBuffer,"Unable to open file %s in append mode",outFile);
    return (ERROREND);
  }
  ifstream Fin(rcFile);
  if (!Fin){
    sprintf(messageBuffer,"Unable to open file %s",rcFile);
    return (ERROREND);
  }
  float bDist;
  float bArea;
  float bDelh;
  float fPorosity;
  float mPerm;
  int flowDir = 1;
  int bcType = 2;
  int m1,m2,inc;
  char buf[100];
  size_t nBuf = sizeof(buf);
```

Bcon.cpp

Page 7 of 8

```
    Fout << ": Boundary condition type 2 on the right side" << endl;
    while (!Fin.eof()){
      Fin.getline(buf,nBuf);
      istrstream inpLine(buf,nBuf);
      inpLine >> m1 >> m2 >> inc >> fPorosity >> mPerm;
      if (inpLine.good()){
        for (int node = m1; node <= m2; node++){
          if (GetBconneData(firstNode,node,flowDir,bDist,bArea,bDelh)
            == ERROREND) return (ERROREND);
          Fout << setw(3) << bcType
               << setw(3) << flowDir
               << setw(5) << node
               << setw(5) << node
               << setw(3) << "1"
               << setiosflags(ios::fixed) << setprecision(3)
               << setw(10) << bDist
               << setw(10) << bArea
               << setw(10) << bDelh
               << resetiosflags(ios::fixed)
               << endl;
          Fout << setw(5) << "0.0"
               << setw(5) << "0.0"
               << setw(5) << "0.0"
               << setw(5) << "0.0"
               << setw(5) << "0.0"
               << setw(5) << "0.0"
               << setw(5) << "0.0"
               << endl;
          Fout << setw(5) << "0.0"
               << setw(5) << "0.0"
               << setw(5) << "0.0"
               << setw(5) << "0.0"
               << setw(5) << "0.0"
               << setw(5) << "0.0"
               << setw(5) << "0.0"
               << endl;
          Fout << "/" << endl;
        }
      }
    }
    return (NORMALEND);
  }
  */

int GetBconneData(BndNode* first, int node, int flowDir,
  float& bDist,float& bArea,float& bDelh)
{
  while (first){
    if (((first->flowDir)==flowDir) && ((first->index)==node)){
      bDist = first->bDist;
      bArea = first->bArea;
      bDelh = first->bDelh;
      return (NORMALEND);
    }
    first = first->next;
  }
  sprintf(messageBuffer,"Conne data not found for node %d with flowDir %d",
    node,flowDir);
  return (ERROREND);
}
void DeleteNodes(BndNode** firstNode)
{
  BndNode* first = *firstNode;
  BndNode* theNode = first;
  while (first){
    theNode = first;
    first = first->next;
    delete theNode;
  }
}
```

Bcon.cpp

Page 8 of 8


```
int PrintNodes(BndNode* theNode)
{
  ofstream Fout(nodeDumpFile);
  if (!Fout){
    sprintf(messageBuffer,"Unable to open dump file for Bcon testing",
      nodeDumpFile);
    return (ERROREND);
  }
  Fout << setw(5) << "pDir"
       << setw(5) << "Node"
       << setw(10) << "bDist"
       << setw(10) << "bArea"
       << setw(10) << "bDelh"
       << endl;
  while (theNode){
    Fout << setw(5) << (theNode->flowDir)
         << setw(5) << (theNode->index)
         << setiosflags(ios::fixed) << setprecision(4)
         << setw(10) << (theNode->bDist)
         << setw(10) << (theNode->bArea)
         << setw(10) << (theNode->bDelh)
         << endl;
    theNode = theNode->next;
  }
  return (NORMALEND);
}
void DumpAndQuit()
{
  cerr << "\n" << messageBuffer << "\n";
  delete[] messageBuffer;
  cerr << "Press any key to end: ";
  getch();
  exit(1);
}
```

Pages 1 3 — 25 Printed 3/25/99

May 7 1999

Pages 26-27 entries by GW 5/7/99

Analysis of model introduced on p. 13 was conducted using METRA. The input files were developed using procedure documented on p. 13--25. The model was presented to METRA through the input file tm4.dat, a portion of which is reproduced below:

GW 5/7/99

GW

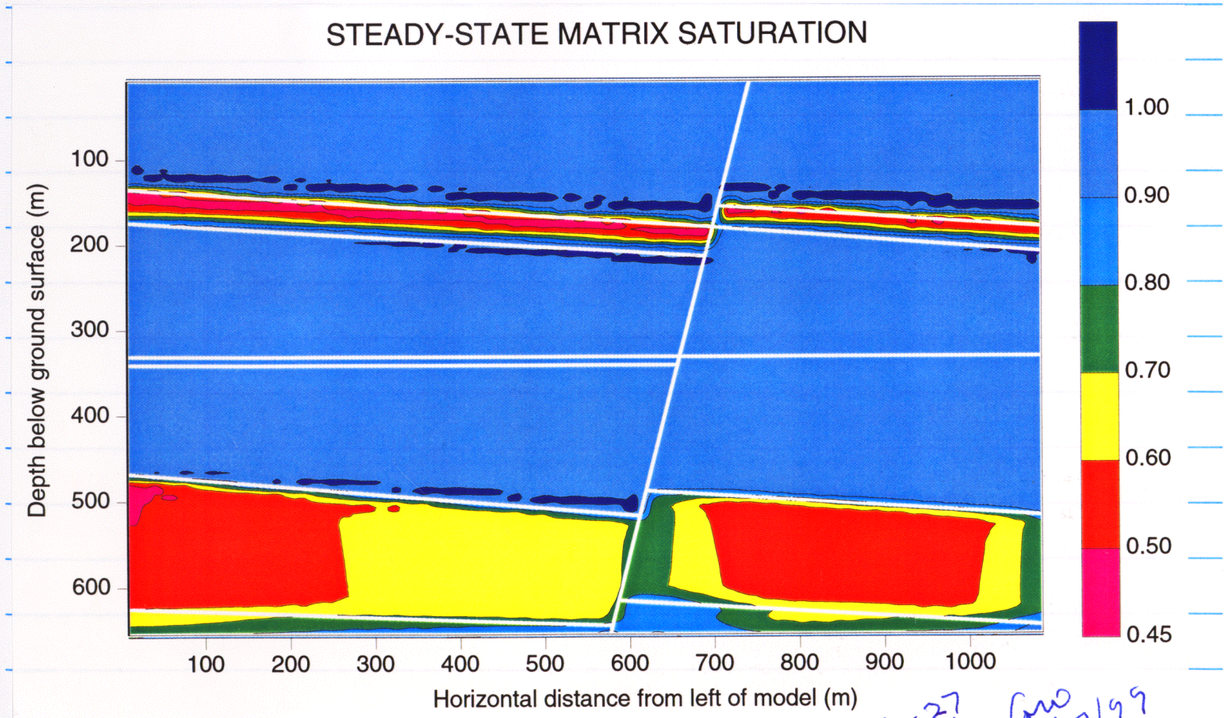
```
..\RepoScale\TestCase\tm4.dat

      Test Model tm4 - Unstructured grid with DCM test case
      Adopted from Ofoegbu et al (1999) model
:
: Infiltration rate = 10.0 mm/yr
:
RSTART  0
:
:grid  geometry  nx ny   nz ivplwr ipvcal  iout  gravity  pref tref  href
Grid   DCMUNSTRUCT     0  0   0   1        1     1        0       0    0     0
Pckr                              :relative perm and pc keyword
:    i  type-curv     swirm    rpm(lamda)  alpha       swext     sgc     iecm
     1  Van-Gen       0.13    0.232       1.15e-6     -1.0e+8   0.0       0 : (TCw12 - matrix)
     2  Van-Gen       0.01    0.492       2.95e-4     -0.0e+8   0.0       0 : (TCw12 - fracture)
     3  Van-Gen       0.10    0.326       4.30e-5     -0.0e+8   0.0       0 : (PTn24 - matrix)
     4  Van-Gen       0.01    0.492       9.33e-4     -0.0e+8   0.0       0 : (PTn24 - fracture)
     5  Van-Gen       0.18    0.323       1.140e-6    -0.0e+8   0.0       0 : (TSw34 - matrix)
     6  Van-Gen       0.01    0.492       9.77e-5     -0.0e+8   0.0       0 : (TSw34 - fracture)
     7  Van-Gen       0.06    0.222       8.65e-5     -1.0e+8   0.0       0 : (CH2vc - matrix)
     8  Van-Gen       0.01    0.492       1.17e-3     -0.0e+8   0.0       0 : (CH2vc - fracture)
     9  Van-Gen       0.07    0.310       1.81e-5     -0.0e+8   0.0       0 : (PP3vp - matrix)
    10  Van-Gen       0.01    0.492       1.41e-3     -0.0e+8   0.0       0 : (PP3vp - fracture)
0
Debug  1
0
: ********************
:  Thermal properties
: ********************
: no = sequential number of data set
: rho = rock density (kg/m^3)
: cpr = rock specific heat (J/kg-K)
: ckdry = thermal conductivity of dry rock (J/s/m-K)
: cksat = thermal conductivity of liquid saturated rock (J/s/m-K)
: crp = pore compressibility with pressure at constant T (1/Pa)
: crt = absolute value of pore compressibility with pressure at constant T (1/Pa)
: tau = tortuosity for binary diffusion
: cdiff = vapor-dir diffusion coefficient, (m^2/s)
: cexp = exponent for binary diffusion
: enbd = enhanced binary diffusion coefficient
:
Thermal-prop
: no rho         cpr     ckdry  cksat    crp  crt   tau  cdiff    cexp  enbd
  1  2.510e+03 837.0    1.28   1.88      0     0    .5  2.13e-5  1.8   0.   : (TCw12 - matrix)
  2  2.260e+03 1330.0   0.35   0.82      0     0    .5  2.13e-5  1.8   0.   : (PTn24 - matrix)
  3  2.530e+03 948.0    1.56   2.33      0     0    .5  2.13e-5  1.8   0.   : (TSw34 - matrix)
  4  2.240e+03 1200.0   0.58   1.17      0     0    .5  2.13e-5  1.8   0.   : (CH2vc - matrix)
  5  2.580e+03 841.0    0.66   1.26      0     0    .5  2.13e-5  1.8   0.   : (PP3vp - matrix)
0
:
conn tm4 :  Read nodal-connections data from file tm4.con
:
PHIK tm4    ! Read nodal-property assignments from file tm4.phk
:
Init
:
:   M1    M2  INC       P        T     SG     XA     PM          TM     SGM    XAM
    1    45  1      9.3E+04 20.0  0.99    0.0 9.3E+04      20.0   0.75   0.0   : Top boundary
   46   659  1      9.3E+04 25.0  0.99    0.0 9.3E+04      25.0   0.75   0.0
  660   680  1      9.3E+04 20.0  0.99    0.0 9.3E+04      20.0   0.75   0.0   : Top boundary
  681  3906  1      9.3E+04 25.0  0.99    0.0 9.3E+04      25.0   0.75   0.0
 3907  3943  1 1.01289E+05 30.0  0.99    0.0 1.0129E+05   30.0   0.09   0.0   : Base boundary
 3944  4001  1      9.3E+04 25.0  0.99    0.0 9.3E+04      25.0   0.75   0.0
 4002  4030  1 1.01289E+05 30.0  0.99    0.0 1.0129E+05   30.0   0.09   0.0   : Base boundary
/
```

GW

The following auxiliary input files are included through commands in tm4.dat:

tm4.con	Node connectivity data
tm4.phk	PHIK input block
tm4.bc	BCON input block

The analysis was started from an arbitrary initial state (p. 26) and consists of transient analysis (with no thermal load) to the steady state. The resulting steady-state saturation for the rock matrix is shown below (white lines are block boundaries, shown as green lines on p. 13).



Pages 26-27 entries by GW 5/7/99

Pages 28-31 entries by GW 5/25/99

May 25 1999 First Site-Scale Model EW01

A vertical section along the East-West line segment AB (below) was selected for the first site-scale model.

from "Viability Assessment of a Repository at Yucca Mountain",

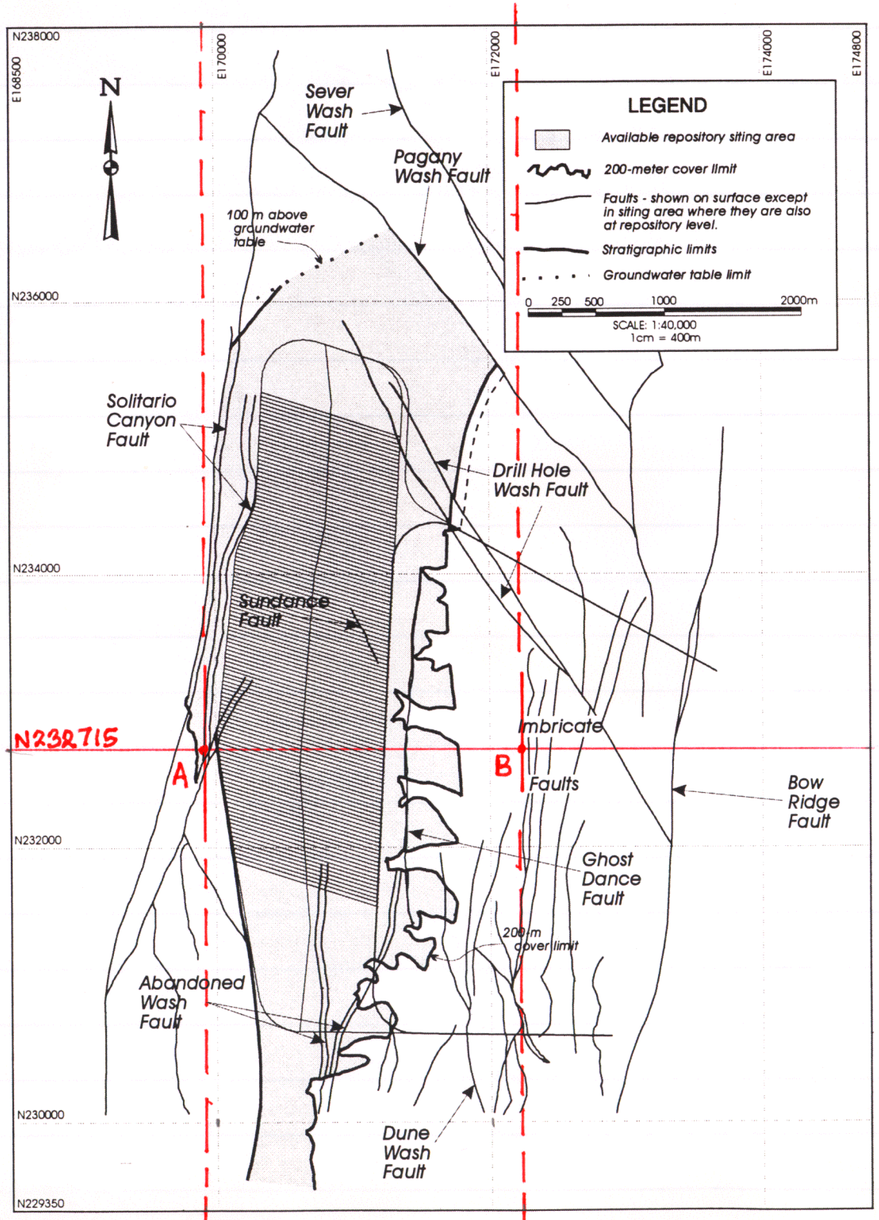
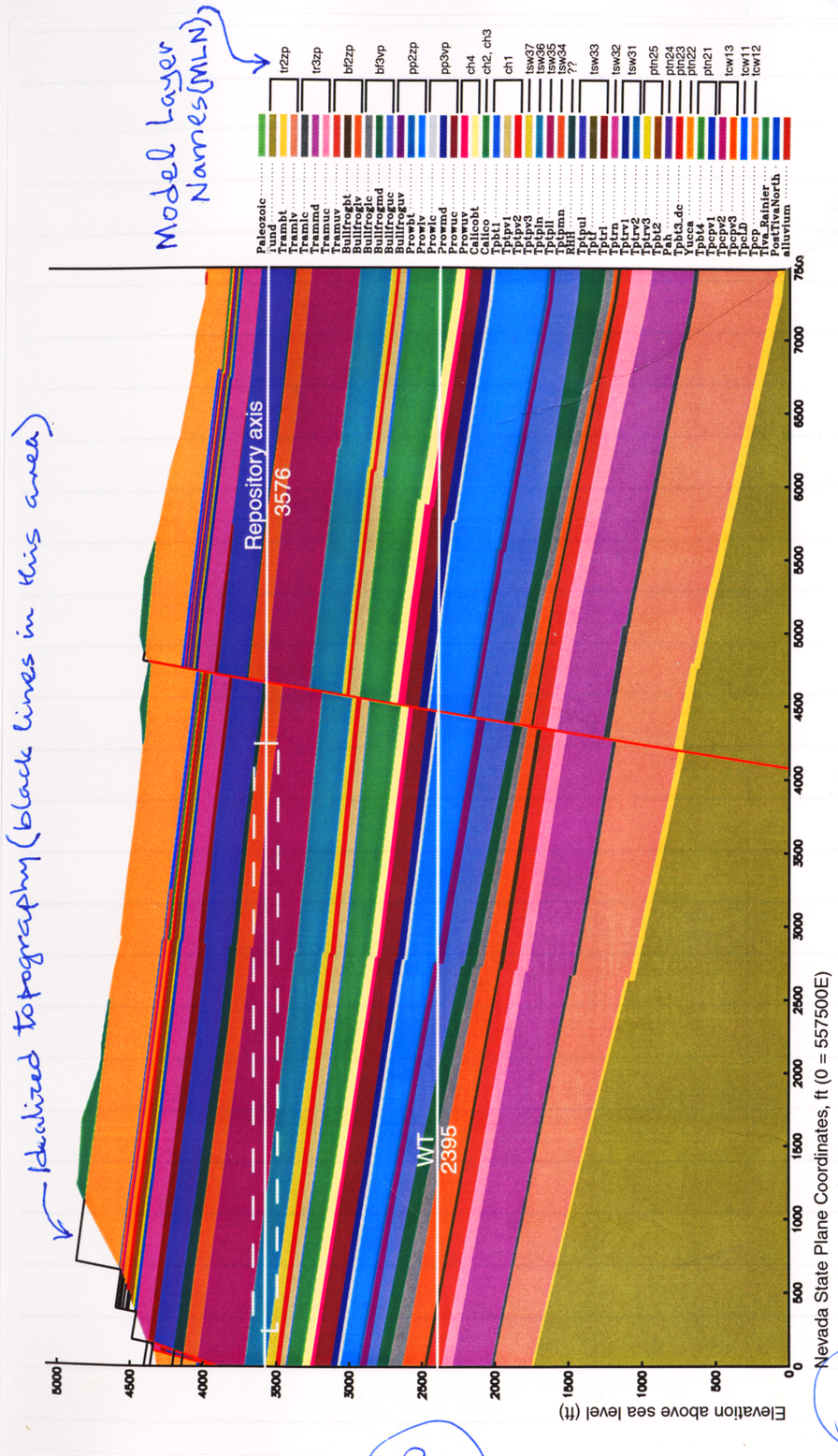


Figure 4-20. Repository Siting Limits



Stratigraphic section along section line AB (p. 28) - Section was extracted from DOE's three-dimensional Earth Vision model of the Yucca Mountain area. The stratigraphic units are related to model layer names (MLN), shown on the right side, through Table 3.4-1 of Bodvarsson et al. (1997). See p. 30.

Table 3.4-1
Relationship between Model Hydrogeological Units and Geological Formations, LBNL Geological Framework Model

Geological Unit	Welding Intensity/Formation Name (Buesch et al., 1995)	Model Layer Name	Hydrogeological Unit
PAINTBRUSH GROUP			
Tiva Canyon Tuff	M,D[1] (Tpcoooo)	tcw11 tcw12	Tiva Canyon (TCw)
	D-Basal vitrophyre (Tpcpv3) M (Tpcpv2)	tcw13	
	N,P (Tpcpv1)		Paintbrush (PTn)
Bedded Tuff	N (Tpbt4)	ptn21	
Yucca Mountain Tuff	N,P,M (Tpy)	ptn22	
Bedded Tuff	N (Tpbt3)	ptn23	
Pah Canyon Tuff	N,P,M (Tpp)	ptn24	
Bedded Tuff	N (Tpbt2)	ptn25	
Topopah Spring Tuff	N,P (Tptrv3)		
	M (Tptrv2) D-Upper vitrophyre (Tptrv1)	tsw31	Topopah Spring (TSw)
	M,D (Tptrn)	tsw32	
	M,D,L[2] (Tptrl) M,D,L (Tptpul)	tsw33	
	D (Tptpmn)	tsw34	
	M,D,L (Tptpll)	tsw35	
	D (Tptpln)	tsw36	
	D-Basal vitrophyre (Tptpv3)	tsw37	
	N,P,M; may be altered (Tptpv1, Tptpv2)	ch1(vc or zc)	Calico Hills (CHn)
Bedded Tuff	N; may be altered (Tpbt1)		
Calico Hills Formation	N; unaltered (Tac-vitric)	ch2(vc or zc)	
	N; altered (Tac-zeolitic)	ch3(vc or zc)	
Bedded Tuff	N; may be altered (Thtbt)	ch4(vc or zc)	
CRATER FLAT GROUP			
Prow Pass Tuff	N; may be altered (Tcp) Unit 4[3]		
	P,M Unit 3	pp3vp	
	N,P; generally altered Units 2,1	pp2zp	
Bedded Tuff	N; generally altered (Tcpbt)		
Upper Bullfrog Tuff	N,P; generally altered (Tcb)		
Middle Bullfrog Tuff	P,M	bf3vp	

From The Site-Scale Unsaturated Zone Model of Yucca Mountain, Nevada, For The Viability Assessment, edited by G.S. Bodvarsson, T.M. Bandurraga, and Y.S. Wu. LBNL-40376 UC-814, 1997.

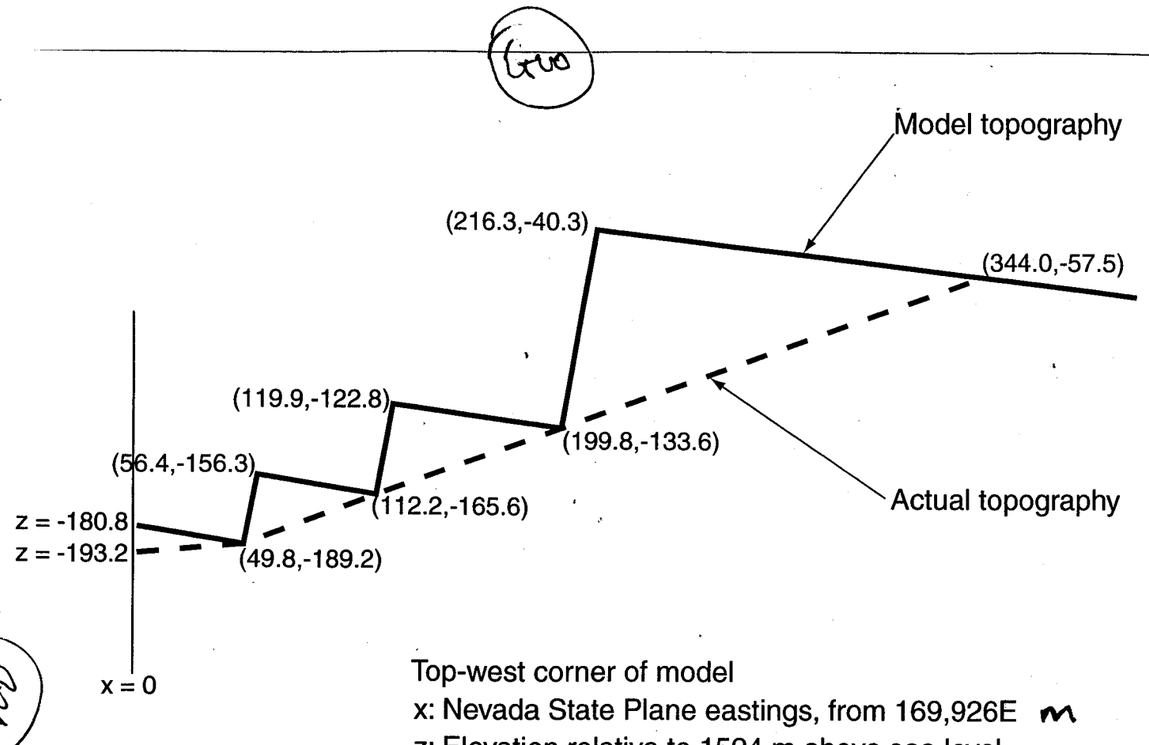
The unit labeled RHH (p. 29) is not found in the table on p. 30. Examination of other DOE reports led to the conclusion that the RHH label on p. 29 is most likely a mistake. Therefore the RHH unit was merged into the tptpmn (middle nonlithophysal or tsw34) unit.

Pages 28–31 entries by GW 5/25/99

May 27 1999 Model EWD1 extends from the ground surface (see figure on p. 29) to the water table at Elevation 2395 ft (730 m) above sea level. The water table elevation was not obtained from the DOE earth vision model ~~it~~ GW 5/27/99 but was estimated using information in the reference Wu et al. GW 5/27/99 cited on p. 30. Personal communication GW 5/25/99 with CNWRA hydrologists (e.g. Randy Fedors) confirms the water table elevation. Alluvial deposits (green color above tcw12) was not included in the model. The GW 5/27/99 ground-surface topography at the top-west corner of the model (along the Solitario Canyon slope) was idealized as shown by dark lines in the figure to facilitate model discretization in that area. The idealized topography at the top-west corner is shown in detail in the figure on p. 32. GW

Pages 31–41 entries by GW 5/27/99

GW



The model was discretized using the four codes listed on pages 13 and 14, as follows:

(1) BLKTONODES (Version 2.0) Develop placement of nodes

(2) AMESH Develop network of cells based on the nodes calculated in (1)

(3) BLKTOPROPS (Version 2.0) Develop ~~in~~ METRA input blocks PHIK, and DCMParams and boundary-node lists using output from (1) and (2)

(4) BCON (Version 2.0) Develop METRA input

block BCON using output from (2) and (3).

The codes BLKTONODES, BLKTOPROPS, and BCON have been modified slightly from the version documented on p. 15-25.

(1) The input file~~s~~ for BLKTONODES (Version 2.0) for model ew01 is

ew01Blk.inp.

BLKTONODES produced the following output files:

ew01Nodes.dat	Input to AMESH. Also input to SURFER, which plots the nodes on an x-y plane.
ew01Blks.bna	Input to SURFER for plotting block geometries.
ew01N__.out (__ = 01, 02, .., 52)	Input to BLKTOPROPS (52 files, one for each rock-layer block).

(2) The input file to AMESH is the file ew01Nodes.dat modified as follows.

in.ew01 → Locate ← statement added to top of file
→ ew01Nodes.dat Main body of input
→ additional information to define model boundaries and tolerances.

The modified file, named in.ew01, was copied to a file named in that is located in the AMESH directory. The output from AMESH is run through a code MFLOMESH (developed by Scot Painter). The output from MFLOMESH was broken into two files as follows (using a text editor):

output from MFLOMESH	ew01.con
	mf2Bcon.ew01

ew01.con is the CONNE input block for METRA.

mf2Bcon.ew01 contains boundary connections that need to be processed further using code BCON (Version 2.0) to obtain the BCON input block for METRA.

(3) The input to BLKTOPROPS (Version 2.0) consists of the following files:

ew01Prop.inp — Associates block numbers with model layer names (p. 35).

set3RkProp.dat — Associates the model layer names with intrinsic hydrologic properties such as porosity and permeability (p. 36).

ew01Vol.inp — Same as the output file ELEME from AMESH, which gives the

D:\HydrTherm\RepoScale\EastWest01\ew01Prop.inp

blkToProp (Version 2.0) input data for model ew01

```
**********************************************************************************************
blk:          Block Number
rkName:       Hydrostratigraphic model layer name
istM:         Associated Van-Gen curve number for matrix in METRA input file
istF:         Associated Van-Gen curve number for fracture in METRA input file
iTh:          Associated thermal-property curve number for matrix in METRA input file
**********************************************************************************************
```

blk	rkName	istM	istF	iTh
1	tcw12	1	2	1
2	tcw13	3	4	2
3	tcw13	3	4	2
4	ptn21	5	6	3
5	ptn22	7	8	4
6	ptn23	9	10	5
7	ptn24	11	12	6
8	ptn25	13	14	7
9	tsw31	15	16	8
10	tsw32	17	18	9
11	tsw32	17	18	9
12	tsw33	19	20	10
13	tsw33	19	20	10
14	tsw33	19	20	10
15	tsw34	21	22	11
16	tsw35	23	24	12
17	tsw36	25	26	13
18	tsw37	27	28	14
19	ch1vc	29	30	15
20	ch2vc	31	32	16
21	ch4vc	33	34	17
22	pp3vp	35	36	18
23	pp2zp	37	38	19
24	bf3vp	39	40	20
25	bf2zp	41	42	21
26	bf2zp	41	42	21
27	bf3vp	39	40	20
28	pp2zp	37	38	19
29	tcw12	1	2	1
30	tcw13	3	4	2
31	ptn21	5	6	3
32	ptn22	7	8	4
33	ptn23	9	10	5
34	ptn24	11	12	6
35	ptn25	13	14	7
36	tsw31	15	16	8
37	tsw32	17	18	9
38	tsw33	19	20	10
39	tsw34	21	22	11
40	tsw35	23	24	12
41	tsw36	25	26	13
42	tsw37	27	28	14
43	ch1vc	29	30	15
44	ch2vc	31	32	16
45	ch2vc	31	32	16
46	ch4vc	33	34	17
47	pp3vp	35	36	18
48	pp2zp	37	38	19
49	pp2zp	37	38	19
50	pp3vp	35	36	18
51	ch4vc	33	34	17
52	ch2vc	31	32	16

D:\HydrTherm\RepoScale\set3RkProp.dat

Yucca Mountain Project Hydrological Rock Properties
Property Set 3: From Tables 3-10 and 3-11 of Near-Field/Altered-Zone Models Report UCRL-ID-129179

rkName: Hydrostratigraphic model layer name
fPor: Fracture porosity (volume fraction of fracture space relative to bulk space)
fKh: Horizontal fracture permeability (m^2)
fKv: Vertical fracture permeability (m^2)
mPor: Matrix porosity
mK: Matrix permeability (m^2)
aFM: Fracture-Matrix interaction factor for DCM (AREAMODF in METRA)
blkLen: Fracture spacing

Fracture-permeability values in this table were based on average flux normal to bulk area;
for DCM models, they are divided by fPor to obtain intrinsic fracture permeability.

rkName	fPor	fKh	fKv	mPor	mK	aFM	blkLen
tcw11	2.33E-04	6.03E-12	2.29E-11	0.066	5.40E-18	5.00E-04	0.5
tcw12	2.99E-04	6.03E-12	1.38E-11	0.066	5.40E-18	5.00E-04	0.5
tcw13	7.05E-05	2.40E-13	2.82E-12	0.14	5.69E-17	5.00E-04	0.5
ptn21	4.84E-05	5.25E-13	5.25E-13	0.369	1.61E-14	5.02E-01	0.5
ptn22	4.83E-05	1.95E-13	1.95E-13	0.234	3.30E-15	5.00E-01	0.5
ptn23	1.30E-04	2.57E-13	2.57E-13	0.353	5.40E-14	5.00E-01	0.5
ptn24	6.94E-05	6.17E-14	6.17E-14	0.469	8.80E-14	5.00E-01	0.5
ptn25	3.86E-05	7.76E-14	7.76E-14	0.464	3.18E-13	5.00E-01	0.5
tsw31	8.92E-05	1.00E-12	1.07E-11	0.042	7.76E-17	4.68E-01	0.5
tsw32	1.29E-04	7.08E-13	1.51E-11	0.146	1.82E-16	5.00E-04	0.5
tsw33	1.05E-04	8.91E-13	2.63E-11	0.135	2.04E-17	5.00E-04	0.5
tsw34	1.24E-04	4.27E-13	6.76E-12	0.089	4.08E-18	1.23E-03	0.5
tsw35	3.29E-04	9.12E-13	3.80E-12	0.115	2.22E-17	5.00E-04	0.5
tsw36	3.99E-04	1.20E-12	1.20E-12	0.092	8.70E-18	5.00E-04	0.5
tsw37	4.92E-04	1.20E-12	1.20E-12	0.02	8.39E-18	5.00E-04	0.5
ch1zc	1.10E-05	2.40E-14	2.51E-14	0.193	1.36E-17	5.00E-01	0.5
ch2zc	1.10E-05	1.17E-14	2.51E-14	0.24	2.50E-18	9.22E-01	0.5
ch3zc	1.10E-05	1.17E-14	2.51E-14	0.24	2.50E-18	9.22E-01	0.5
ch4zc	1.10E-05	1.55E-14	2.51E-14	0.169	5.49E-18	5.00E-01	0.5
ch1vc	7.14E-05	1.74E-13	1.74E-13	0.265	1.60E-12	5.00E-01	0.5
ch2vc	7.14E-05	2.88E-13	2.88E-13	0.321	5.50E-14	5.00E-01	0.5
ch3vc	7.14E-05	2.88E-13	2.88E-13	0.321	5.50E-14	5.00E-01	0.5
ch4vc	7.14E-05	2.88E-13	2.88E-13	0.321	5.50E-14	5.00E-01	0.5
pp3vp	7.14E-05	6.92E-13	7.08E-13	0.274	1.91E-15	5.00E-04	0.5
bf3vp	7.14E-05	6.92E-13	7.08E-13	0.274	1.91E-15	5.00E-04	0.5
tm3vt	7.14E-05	6.92E-13	7.08E-13	0.274	1.91E-15	5.00E-04	0.5
pp2zp	1.10E-05	6.46E-14	2.51E-14	0.197	1.75E-17	5.00E-01	0.5
bf2zp	1.10E-05	6.46E-14	2.51E-14	0.197	1.75E-17	5.00E-01	0.5

Cell volumes for each node.

ew01N__.out (-- = 01, 02, ..., 52) Give the node numbers and coordinates for each block. The files were output from BLKTONODES (Version 2.0).

The first file, ew01Prop.inp (p. 35), also associates the model layer names (tcw11, etc) with Van-Genuchten and thermal-property curve numbers for the fractures and matrix.

The curve numbers are assigned in the METRA input blocks PCKR and THERM, both of which are reproduced on this page (p. 37) and on p. 38.

D:\HydrTherm\RepoScale\thPropCurves.dat

```
:
:  METRA-DCM, Thermal and Hydrologic Properties:
:  From Near-Field/Altered-Zone Models Report UCRL-ID-129179
: ********************************************************************
: Hydrologic properties from Tables 3-10 and 3-11 of UCRL-ID-129179
: ********************************************************************
: i = sequential number of material types
: type = the characteristic curves (Van-Gen, Linear, tabular, and corey)
: swirm = irreducible liquid saturation for the matrix
: rpmm = Van Genuchten parameter for matrix
: alpham = Van Genuchten parameter for matrix
: swext = liquid saturation below which the capillary pressure
: is calculated based on the slope dPcw/dSw evaluated at SWEXT.
: sgc = residual (immobile) gas saturation, fraction
: iecm = Equivalent Continuum Model (ECM) formulation
:    (0 do not invoke, 1 invoke, 2 ECM with tables)
: swirf = residual liquid saturation for fracture, fraction
: alphaf = parameter in Van-Genuchten equation for fracture (1/Pa)
: phim = matrix porosity (fraction)
: phif = fracture porosity (fraction)
: permm = intinsic matrix permeability (m^2)
: permf = intrinsic fracture permeability
:
Pckr                              :relative perm and pc keyword
:   i  type-curv   swirm   rpmm(lamda) alpham     swext   sgc     iecm
    1  Van-Gen     0.13    0.2447      2.01e-6    -1.0e+8  0.0     0  : (TCw12 - matrix)
    2  Van-Gen     0.01     0.669      2.37e-3    -0.0e+8  0.0     0  : (TCw12 - fracture)
    3  Van-Gen     0.33    0.4548      3.74e-6    -0.0e+8  0.0     0  : (TCw13 - matrix)
    4  Van-Gen     0.01     0.669      9.12e-4    -0.0e+8  0.0     0  : (TCw13 - fracture)
    5  Van-Gen     0.10    0.2531      3.98e-5    -1.0e+8  0.0     0  : (PTn21 - matrix)
    6  Van-Gen     0.01     0.669      1.10e-3    -0.0e+8  0.0     0  : (PTn21 - fracture)
    7  Van-Gen     0.14    0.4925      7.94e-6    -0.0e+8  0.0     0  : (PTn22 - matrix)
    8  Van-Gen     0.01     0.669      1.85e-3    -0.0e+8  0.0     0  : (PTn22 - fracture)
    9  Van-Gen     0.17    0.3002      5.44e-5    -0.0e+8  0.0     0  : (PTn23 - matrix)
   10  Van-Gen     0.01     0.667      3.45e-3    -0.0e+8  0.0     0  : (PTn23 - fracture)
   11  Van-Gen     0.10    0.3859      3.43e-5    -0.0e+8  0.0     0  : (PTn24 - matrix)
   12  Van-Gen     0.01     0.667      9.13e-4    -0.0e+8  0.0     0  : (PTn24 - fracture)
   13  Van-Gen     0.10    0.3195      1.81e-4    -0.0e+8  0.0     0  : (PTn25 - matrix)
   14  Van-Gen     0.10     0.320      1.81e-4    -0.0e+8  0.0     0  : (PTn25 - fracture)
   15  Van-Gen     0.11    0.2304      5.84e-5    -1.0e+8  0.0     0  : (TSw31 - matrix)
   16  Van-Gen     0.01     0.566      1.44e-4    -0.0e+8  0.0     0  : (TSw31 - fracture)
   17  Van-Gen     0.04    0.2861      2.00e-5    -1.0e+8  0.0     0  : (TSw32 - matrix)
   18  Van-Gen     0.01     0.667      1.42e-3    -1.0e+8  0.0     0  : (TSw32 - fracture)
   19  Van-Gen     0.06    0.2479      6.21e-6    -1.0e+8  0.0     0  : (TSw33 - matrix)
   20  Van-Gen     0.01     0.667      1.73e-3    -0.0e+8  0.0     0  : (TSw33 - fracture)
   21  Van-Gen     0.18    0.3212      1.19e-6    -1.0e+8  0.0     0  : (TSw34 - matrix)
   22  Van-Gen     0.01     0.643      9.34e-4    -1.0e+8  0.0     0  : (TSw34 - fracture)
   23  Van-Gen     0.08    0.1983      4.01e-6    -1.0e+8  0.0     0  : (TSw35 - matrix)
   24  Van-Gen     0.01     0.667      1.26e-3    -0.0e+8  0.0     0  : (TSw35 - fracture)
   25  Van-Gen     0.18    0.5138      8.08e-7    -0.0e+8  0.0     0  : (TSw36 - matrix)
   26  Van-Gen     0.01     0.667      1.32e-3    -0.0e+8  0.0     0  : (TSw36 - fracture)
   27  Van-Gen     0.50    0.3709      5.30e-7    -0.0e+8  0.0     0  : (TSw37 - matrix)
   28  Van-Gen     0.01     0.659      1.19e-3    -0.0e+8  0.0     0  : (TSw37 - fracture)
   29  Van-Gen     0.04    0.1592      7.60e-5    -1.0e+8  0.0     0  : (CH1vc - matrix)
   30  Van-Gen     0.01     0.669      1.18e-3    -0.0e+8  0.0     0  : (CH1vc - fracture)
   31  Van-Gen     0.06    0.2291      4.12e-5    -1.0e+8  0.0     0  : (CH2/3vc - matrix)
   32  Van-Gen     0.01     0.667      1.18e-3    -0.0e+8  0.0     0  : (CH2/3vc - fracture)
   33  Van-Gen     0.33    0.2291      4.12e-5    -0.0e+8  0.0     0  : (CH4vc - matrix)
   34  Van-Gen     0.01     0.667      1.18e-3    -0.0e+8  0.0     0  : (CH4vc - fracture)
   35  Van-Gen     0.07    0.3142      1.66e-5    -0.0e+8  0.0     0  : (PP3vp - matrix)
   36  Van-Gen     0.01     0.667      1.42e-3    -0.0e+8  0.0     0  : (PP3vp - fracture)
   37  Van-Gen     0.18    0.3568      8.39e-6    -0.0e+8  0.0     0  : (PP2zp - matrix)
   38  Van-Gen     0.01     0.667      1.14e-3    -0.0e+8  0.0     0  : (PP2zp - fracture)
   39  Van-Gen     0.07    0.3142      1.66e-5    -0.0e+8  0.0     0  : (BF3vp - matrix)
   40  Van-Gen     0.01     0.667      1.42e-3    -0.0e+8  0.0     0  : (Bf3vb - fracture)
   41  Van-Gen     0.18    0.3568      8.39e-6    -1.0e+8  0.0     0  : (Bf2zp - matrix)
   42  Van-Gen     0.01     0.667      1.14e-3    -1.0e+8  0.0     0  : (Bf2zp - fracture)
0
```

```
: *****************************************************
: Thermal properties from Table 3-5 of UCRL-ID-129179
: *****************************************************
: no = sequential number of data set
: rho = rock density (kg/m^3)
: cpr = rock specific heat (J/kg-K)
: ckdry = thermal conductivity of dry rock (J/s/m-K)
: cksat = thermal conductivity of liquid saturated rock (J/s/m-K)
: crp = pore compressibility with pressure at constant T (1/Pa)
: crt = absolute value of pore compressibility with pressure at constant T (1/Pa)
: tau = tortuosity for binary diffusion
: cdiff = vapor-dir diffusion coefficient, (m^2/s)
: cexp = exponent for binary diffusion
: enbd = enhanced binary diffusion coefficient
:
Thermal-prop
: no rho       cpr     ckdry   cksat    crp   crt    tau   cdiff     cexp  enbd
  1  2510.0    837.    1.28    1.88     0     0     .5   2.13e-5   1.8   0.    : (TCw12 - matrix)
  2  2470.0    857.    0.54    0.98     0     0     .5   2.13e-5   1.8   0.    : (TCw13 - matrix)
  3  2340.0   1080.    0.35    0.50     0     0     .5   2.13e-5   1.8   0.    : (PTn21 - matrix)
  4  2400.0    849.    0.44    0.97     0     0     .5   2.13e-5   1.8   0.    : (PTn22 - matrix)
  5  2370.0   1020.    0.46    1.02     0     0     .5   2.13e-5   1.8   0.    : (PTn23 - matrix)
  6  2260.0   1330.    0.35    0.82     0     0     .5   2.13e-5   1.8   0.    : (PTn24 - matrix)

  7  2370.0   1220.    0.23    0.67     0     0     .5   2.13e-5   1.8   0.    : (PTn25 - matrix)
  8  2510.0    834.    0.37    1.00     0     0     .5   2.13e-5   1.8   0.    : (TSw31 - matrix)
  9  2550.0    866.    1.06    1.62     0     0     .5   2.13e-5   1.8   0.    : (TSw32 - matrix)
 10  2510.0    883.    0.71    1.80     0     0     .5   2.13e-5   1.8   0.    : (TSw33 - matrix)
 11  2530.0    948.    1.56    2.33     0     0     .5   2.13e-5   1.8   0.    : (TSw34 - matrix)
 12  2540.0    900.    1.20    2.02     0     0     .5   2.13e-5   1.8   0.    : (TSw35 - matrix)
 13  2560.0    865.    1.42    1.84     0     0     .5   2.13e-5   1.8   0.    : (TSw36 - matrix)
 14  2360.0    984.    1.69    2.08     0     0     .5   2.13e-5   1.8   0.    : (TSw37 - matrix)
 15  2310.0   1060.    0.70    1.31     0     0     .5   2.13e-5   1.8   0.    : (CH1vc - matrix)
 16  2240.0   1200.    0.58    1.17     0     0     .5   2.13e-5   1.8   0.    : (CH2/3vc - matrix)
 17  2440.0   1200.    0.58    1.17     0     0     .5   2.13e-5   1.8   0.    : (CH4vc - matrix)
 18  2580.0    841.    0.66    1.26     0     0     .5   2.13e-5   1.8   0.    : (PP3vp - matrix)
 19  2510.0    644.    0.74    1.35     0     0     .5   2.13e-5   1.8   0.    : (PP2zp - matrix)
 20  2580.0    841.    0.66    1.26     0     0     .5   2.13e-5   1.8   0.    : (BF3vp - matrix)
 21  2510.0    644.    0.74    1.35     0     0     .5   2.13e-5   1.8   0.    : (BF2zp - matrix)
0
```

The output from BLKTOPROPS (Version 2.0) are:

ew01.phk	PHIK input block
ew01.dcm	DCMPar input block
bcTop.out } bcBase.out }	List of nodes for top- and base- boundary conditions. These files also give the coordinates and values of fracture porosity and matrix permeability for each boundary node.

(4) The inputs for BCON (Version 2.0) are:

bcTop.out } bcBase.out }	Output from BLKTOPROPS (see p. 38).
Mf2Bcon.ew01	Output from AMESH (see p. 34).

The output from BCON (Version 2.0) is the file ew01.bc, which gives the BCON input block for METRA.

The finished model, 5/28/99 the input to code METRA, consists of the following files:

ew01.dat	METRA input file

which incorporates the following files by reference:

ew01.con	CONNE input block
ew01.bc	BCON input block
ew01.phk	PHIK input block
ew01.dcm	DCMPar input block

The finished model is also shown as a mesh plot on p. 40-41.

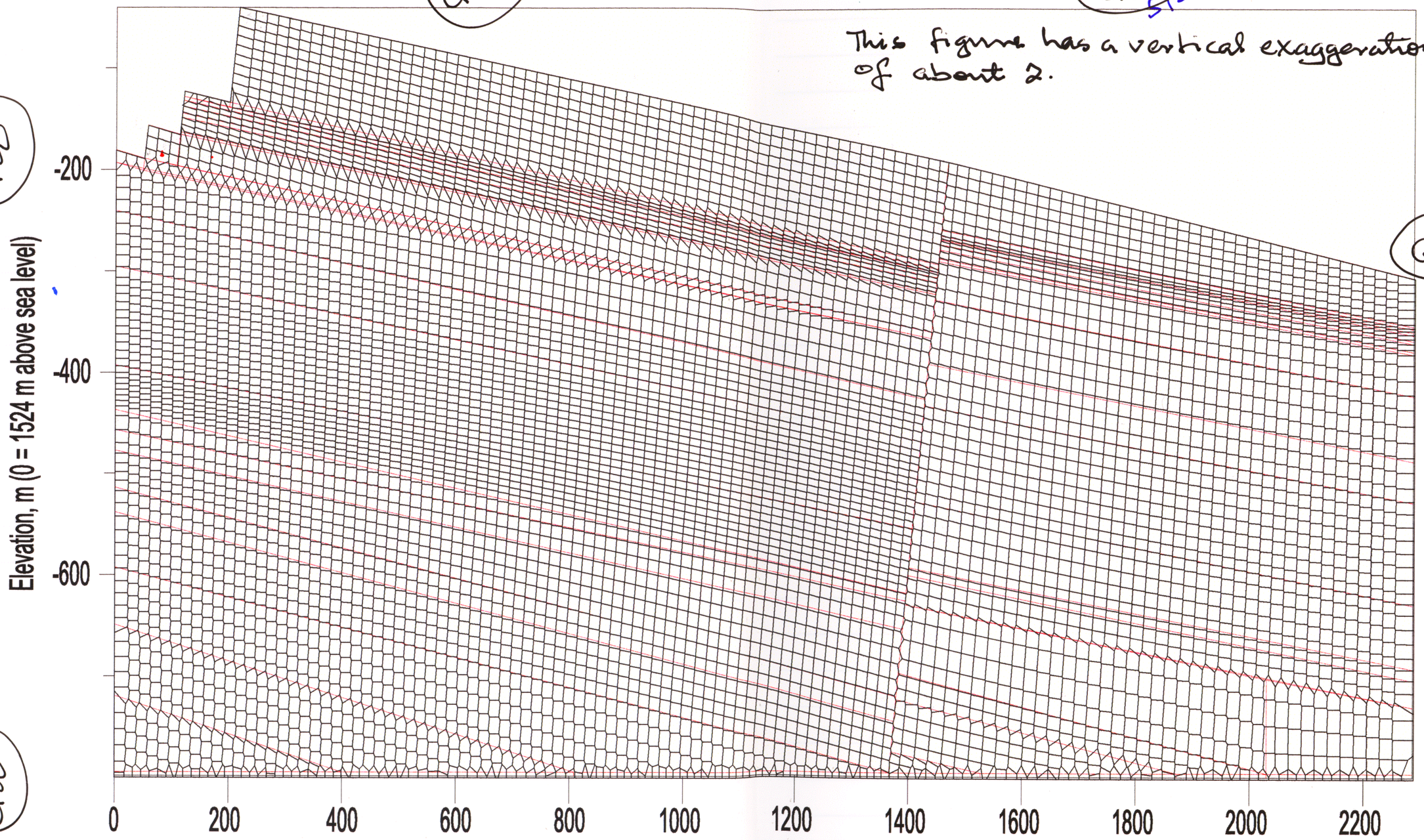
GW

GW 5/27/99

This figure has a vertical exaggeration of about 2.

GW

GW 5/27/99



GW

Pages 31-41 entered GW 5/27/99

July 7 1999

Infiltration Rate

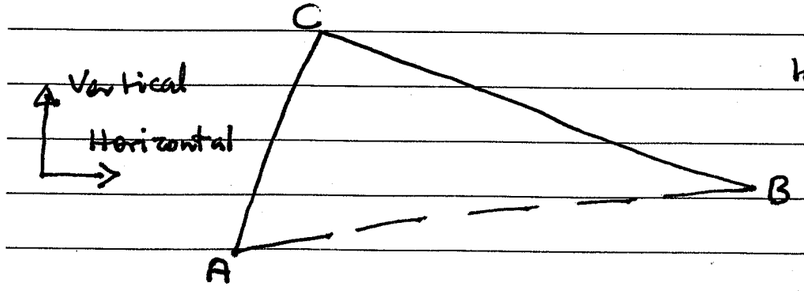
Pages 42-47 by entries GW 7/7/99

Model ew01 is setup initially for a net infiltration rate of 10 mm/yr. Because the ground surface is not horizontal (figure on p. 40-41), the input infiltration rate was modified as follows to ensure that an ~~equivalent~~ infiltration rate equivalent (GW 7/7/99) to 10 mm/yr is applied over the top of the model:

$$Q_{in} = R_f Q_o$$

Q_o = 10 mm/yr
Q_{in} = Input infiltration rate
R_f = modification factor.

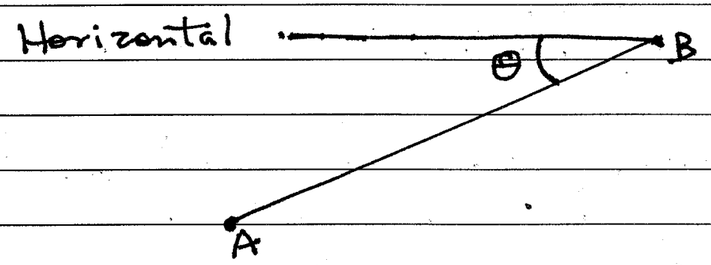
The factor R_f is calculated to account for the slope of the actual topography and the difference between the actual and model topographies if the two differ, e.g., p. 32.



Infiltration incident on AB (actual topography) was applied over the surface CB (model topography). No-flow condition was applied on the surface AC. Surfaces AB and CB are coincident over most of the model (except near the top-west corner as shown on p. 29 and 32). The ratio R_f is given by

$$R_f = \left(\frac{l_o}{l}\right) \cos\theta$$

l = length of CB (p. 42)
l_o = length of AB (p. 42)
θ = Inclination of AB from the horizontal



R_f was evaluated as a function of the horizontal coordinate, x (m), as shown in the table here. The evaluation of the infiltration input to MMETRA is implemented (GW 7/7/99) in the code BCON.

GW 7/7/99

Section	l_o (m)	l (m)	$\cos\theta$	R_f
$x \le 49.8$	49.9604	50.5035	0.9968	0.9861
$49.8 < x \le 112.2$	66.7137	56.5697	0.9353	1.1030
$112.2 < x \le 199.8$	93.2618	80.6266	0.9393	1.0865
$199.8 < x \le 344.0$	163.0486	128.8531	0.8844	1.1191
$x > 344$	1957.74	1957.74	0.9920	0.9920

GW

Other Boundary Conditions

(1) Top of Model

In addition to the infiltration rate (specified as a mass flux), both temperature and atmospheric pressure were fixed at the top of the model. The top-surface temperature θ_t was evaluated as a function of elevation Z_e using the following empirical equation (S. Stothoff, personal communication):

$$\theta_t\,(^{\circ}C) = 21.667 - 0.006063(Z_e)$$

where Z_e = surface elevation in m above sea level.

The atmospheric pressure, P_{atm}, was also evaluated as a function of Z_e using the following empirical equation (S. Stothoff, personal communication):

$$P_{atm} = 29372 - 2.8494(Z_e)$$

where P_{atm} is in 10^{-3} inches of mercury, which can be converted to P_a (N/m^2) by multiplication with the following factor, f_{hgi}

$$f_{hgi} = (2.54 \times 10^{-5})(13.6\,\rho_w\,g)$$

where ρ_w is the density of water (kg/m^3) and g is gravitational acceleration ($9.81\ m/s^2$).

The infiltration flux is specified as a mass flux and is split between fracture and matrix as follows: The input infiltration (p. 42) is converted to a mass flux using the equation

$$Q_{in} = R_f\,Q_0\,\rho_w$$

which gives Q_{in} in $kg/(m^2 \cdot s)$ if Q_0 is in m/s and ρ_w is in kg/m^3. The maximum flux through the matrix is estimated as (GW 7/7/99)

$$Q_m = 0.9\,k_m\left(\frac{g\,\rho_w^2}{\eta_w}\right)$$

where k_m is the matrix permeability (m^2), and η_w is the viscosity of water ($Pa \cdot s$). If ($Q_{in} < Q_m$), then Q_m is reset to (GW 7/7/99)

$$Q_m = (1 - \phi_f)\,Q_{in}$$

where ϕ_f is the fracture porosity (fracture volume per unit bulk volume). Whether Q_m is given by the first (GW 7/7/99)

or second equation, the fracture flux Q_f is calculated as

$$Q_f = (Q_{in} - Q_m) / \phi_f$$

Both f_w and η_w were evaluated as functions of temperature using the following empirical data:

Temperature (°C)	Density (kg/m^3)	Viscosity ($Pa \cdot s$)
5	1000.0	1.514×10^{-3}
10	999.7	1.304×10^{-3}
15	999.1	1.137×10^{-3}
20	998.2	1.002×10^{-3}
25	997.1	8.91×10^{-4}
30	995.7	7.98×10^{-4}
35	994.1	7.20×10^{-4}

Source: Bejan, A. 1984. Convection Heat Transfer. John Wiley and Sons, New York. ~~(Tables~~ (p. 462-463). GW 7/7/99

(2) Base of Model.

The base of the 7/7/99 model (ew01) is the water table. ~~and it Below~~ (GW) Fixed temperature, pressure and saturation were specified

at the base. The base temperature was set to 30°C, which is about the same as the average water table temperature [compare with Wu, Y.S., C Hankwa, and G.S Bodvarsson, 1999. A site-scale model for fluid and heat flow in the unsaturated zone of Yucca Mountain, Nevada. Journal of Contaminant Hydrology, 38: 185-215]. The ~~atm~~ gas pressure at the water table was calculated by substituting ~~Ze = 730~~ Ze = 730 m in the equation for atmospheric pressure (p. 44). (GW 7/7/99) Water saturation was set to 1.0 at the model base.

(3) The two vertical boundaries of the model were treated as no-flow boundaries (for both mass and energy).

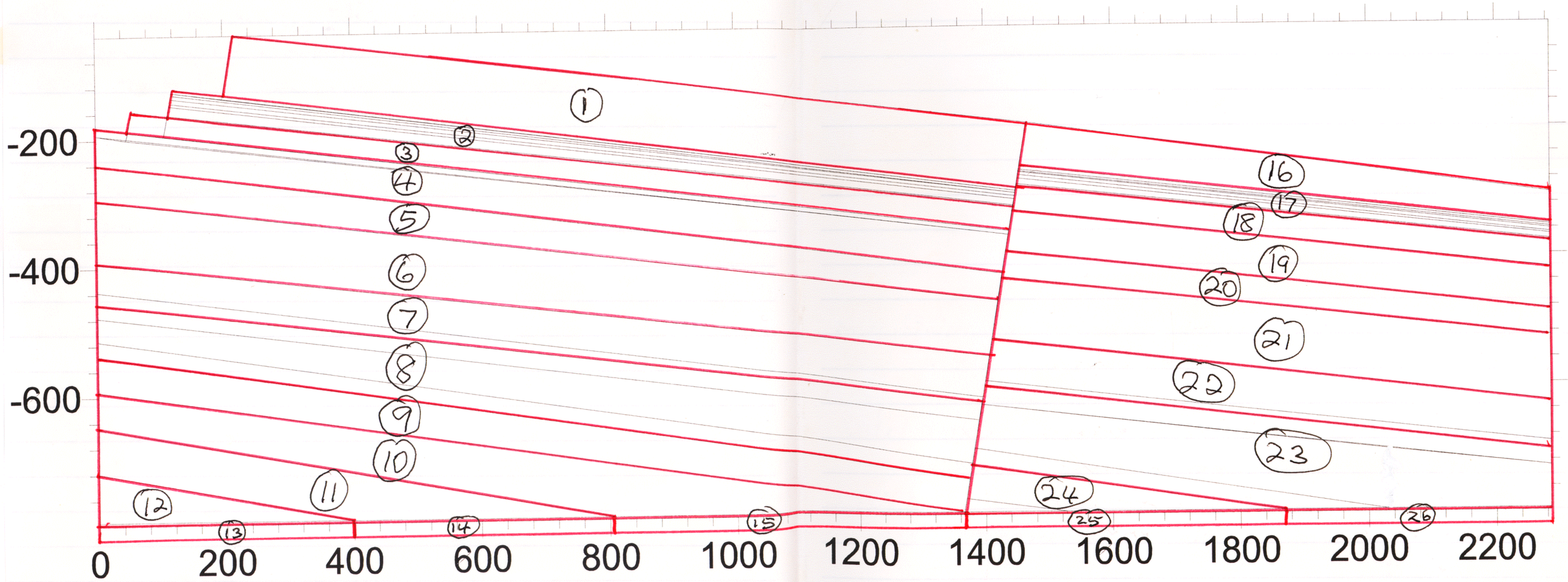
Pages 42-47 by GW 7/7/99 entries

Pages 48-53 entries by CW 10/25/99

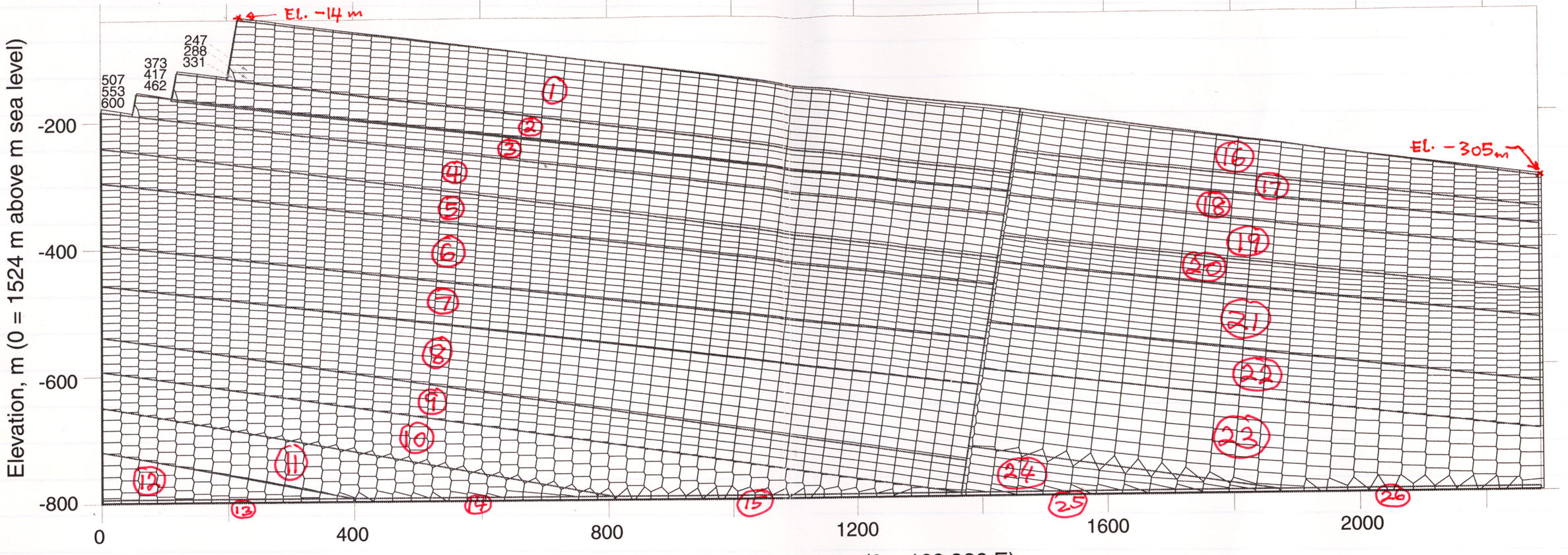
October 25 1999 New Model m20

Several unsuccessful attempts have been made to complete a transient analysis to steady state (without thermal loading) using model ew01. In all attempts the analysis proceeded extremely slowly with time increments less than order of 10^{-4} yr. Therefore, the model was modified by lumping some of the rock layers (especially the PTn layers) together as shown in the figure below (dark lines are ew01 interfaces and red lines are the interfaces ~~of the~~ used in the modified model).

The new model (m20) is shown on p. 50-51



Model m20



The input definitions for model m20 are contained in the following files.

m20.dat	METRA basic input file
m20.con	Connections data
m20.phk	Rock properties definition
m20.dcm	DCMPARAMS input block
m20.bc	Boundary conditions

The block-property assignments and the hydrologic and thermal property curves are given on p. 52–53. The block-property file allows individual cells to be matched with the hydrologic-property data on p. 36 (using the block numbers displayed on this page).

GWE 10/25/99

Block Property Assignments for Model m20.

m20Prop.inp

blkToProp (Version 2.0) input data for model m20

```
**********************************************************************************
blk:        Block Number
rkName:     Hydrostratigraphic model layer name
istM:       Associated Van-Gen curve number for matrix in METRA input file
istF:       Associated Van-Gen curve number for fracture in METRA input file
iTh:        Associated thermal-property curve number for matrix in METRA input file
**********************************************************************************
```

blk	rkName	istM	istF	iTh
1	tcw12	1	2	1
2	ptn21	3	4	2
3	tsw32	5	6	3
4	tsw33	7	8	4
5	tsw34	9	10	5
6	tsw35	11	12	6
7	tsw36	13	14	7
8	ch2vc	15	16	8
9	pp3vp	17	18	9
10	pp2zp	19	20	10
11	bf3vp	21	22	11
12	bf2zp	23	24	12
13	bf2zp	23	24	12
14	bf3vp	21	22	11
15	pp2zp	19	20	10
16	tcw12	1	2	1
17	ptn21	3	4	2
18	tsw32	5	6	3
19	tsw33	7	8	4
20	tsw34	9	10	5
21	tsw35	11	12	6
22	tsw36	13	14	7
23	ch2vc	15	16	8
24	pp3vp	17	18	9
25	pp3vp	17	18	9
26	ch2vc	15	16	8

GWD

GWD

GWD

Hydrologic and Thermal Property Curves for Model m20.

m20ThCurves.dat

```
:
:  METRA-DCM, Thermal and Hydrologic Properties for model m20:
:  From Near-Field/Altered-Zone Models Report UCRL-ID-129179
:
: ****************************************************************
: Hydrologic properties from Tables 3-10 and 3-11 of UCRL-ID-129179
: ****************************************************************
: i = sequential number of material types
: type = the characteristic curves (Van-Gen, Linear, tabular, and corey)
: swirm = irreducible liquid saturation for the matrix
: rpmm = Van Genuchten parameter for matrix
: alpham = Van Genuchten parameter for matrix
: swext = liquid saturation below which the capillary pressure
: is calculated based on the slope dPcw/dSw evaluated at SWEXT.
: sgc = residual (immobile) gas saturation, fraction
: iecm = Equivalent Continuum Model (ECM) formulation
:    (0 do not invoke, 1 invoke, 2 ECM with tables)
: swirf = residual liquid saturation for fracture, fraction
: alphaf = parameter in Van-Genuchten equation for fracture (1/Pa)
: phim = matrix porosity (fraction)
: phif = fracture porosity (fraction)
: permm = intinsic matrix permeability (m^2)
: permf = intrinsic fracture permeability
:
Pckr                              :relative perm and pc keyword
:   i  type-curv   swirm   rpmm(lamda) alpham     swext    sgc     iecm
    1  Van-Gen     0.13    0.2447     2.01e-6   -0.0e+8   0.0      0  : (TCw12 - matrix)
    2  Van-Gen     0.01     0.669     2.37e-3   -0.0e+8   0.0      0  : (TCw12 - fracture)
    3  Van-Gen     0.10    0.3859     3.43e-5   -0.0e+8   0.0      0  : (PTn24 - matrix)
    4  Van-Gen     0.01     0.667     9.13e-4   -0.0e+8   0.0      0  : (PTn24 - fracture)
    5  Van-Gen     0.04    0.2861     2.00e-5   -0.0e+8   0.0      0  : (TSw32 - matrix)
    6  Van-Gen     0.01     0.667     1.42e-3   -0.0e+8   0.0      0  : (TSw32 - fracture)
    7  Van-Gen     0.06    0.2479     6.21e-6   -0.0e+8   0.0      0  : (TSw33 - matrix)
    8  Van-Gen     0.01     0.667     1.73e-3   -0.0e+8   0.0      0  : (TSw33 - fracture)
    9  Van-Gen     0.18    0.3212     1.19e-6   -0.0e+8   0.0      0  : (TSw34 - matrix)
   10  Van-Gen     0.01     0.643     9.34e-4   -0.0e+8   0.0      0  : (TSw34 - fracture)
   11  Van-Gen     0.08    0.1983     4.01e-6   -0.0e+8   0.0      0  : (TSw35 - matrix)
   12  Van-Gen     0.01     0.667     1.26e-3   -0.0e+8   0.0      0  : (TSw35 - fracture)
   13  Van-Gen     0.18    0.5138     8.08e-7   -0.0e+8   0.0      0  : (TSw36 - matrix)
   14  Van-Gen     0.01     0.667     1.32e-3   -0.0e+8   0.0      0  : (TSw36 - fracture)
   15  Van-Gen     0.06    0.2291     4.12e-5   -0.0e+8   0.0      0  : (CH2/3vc - matrix)
   16  Van-Gen     0.01     0.667     1.18e-3   -0.0e+8   0.0      0  : (CH2/3vc - fracture)
   17  Van-Gen     0.07    0.3142     1.66e-5   -0.0e+8   0.0      0  : (PP3vp - matrix)
   18  Van-Gen     0.01     0.667     1.42e-3   -0.0e+8   0.0      0  : (PP3vp - fracture)
   19  Van-Gen     0.18    0.3568     8.39e-6   -0.0e+8   0.0      0  : (PP2zp - matrix)
   20  Van-Gen     0.01     0.667     1.14e-3   -0.0e+8   0.0      0  : (PP2zp - fracture)
   21  Van-Gen     0.07    0.3142     1.66e-5   -0.0e+8   0.0      0  : (BF3vp - matrix)
   22  Van-Gen     0.01     0.667     1.42e-3   -0.0e+8   0.0      0  : (Bf3vp - fracture)
   23  Van-Gen     0.18    0.3568     8.39e-6   -0.0e+8   0.0      0  : (Bf2zp - matrix)
   24  Van-Gen     0.01     0.667     1.14e-3   -0.0e+8   0.0      0  : (Bf2zp - fracture)
0
:
: ******************************************************
:  Thermal properties from Table 3-5 of UCRL-ID-129179
: ******************************************************
: no = sequential number of data set
: rho = rock density (kg/m^3)
: cpr = rock specific heat (J/kg-K)
: ckdry = thermal conductivity of dry rock (J/s/m-K)
: cksat = thermal conductivity of liquid saturated rock (J/s/m-K)
: crp = pore compressibility with pressure at constant T (1/Pa)
: crt = absolute value of pore compressibility with pressure at constant T (1/Pa)
: tau = tortuosity for binary diffusion
: cdiff = vapor-dir diffusion coefficient, (m^2/s)
: cexp = exponent for binary diffusion
: enbd = enhanced binary diffusion coefficient
:
Thermal-prop
: no rho       cpr     ckdry  cksat    crp  crt   tau  cdiff     cexp  enbd
   1  2510.0   837.    1.28   1.88      0    0    .5  2.13e-5   1.8   0.   : (TCw12 - matrix)
   2  2340.0  1080.    0.35   0.50      0    0    .5  2.13e-5   1.8   0.   : (PTn21 - matrix)
   3  2550.0   866.    1.06   1.62      0    0    .5  2.13e-5   1.8   0.   : (TSw32 - matrix)
   4  2510.0   883.    0.71   1.80      0    0    .5  2.13e-5   1.8   0.   : (TSw33 - matrix)
   5  2530.0   948.    1.56   2.33      0    0    .5  2.13e-5   1.8   0.   : (TSw34 - matrix)
   6  2540.0   900.    1.20   2.02      0    0    .5  2.13e-5   1.8   0.   : (TSw35 - matrix)
   7  2560.0   865.    1.42   1.84      0    0    .5  2.13e-5   1.8   0.   : (TSw36 - matrix)
   8  2240.0  1200.    0.58   1.17      0    0    .5  2.13e-5   1.8   0.   : (CH2/3vc - matrix)
   9  2580.0   841.    0.66   1.26      0    0    .5  2.13e-5   1.8   0.   : (PP3vp - matrix)
  10  2510.0   644.    0.74   1.35      0    0    .5  2.13e-5   1.8   0.   : (PP2zp - matrix)
  11  2580.0   841.    0.66   1.26      0    0    .5  2.13e-5   1.8   0.   : (BF3vp - matrix)
  12  2510.0   644.    0.74   1.35      0    0    .5  2.13e-5   1.8   0.   : (BF2zp - matrix)
0
```

GWD

GWD

Pages 48 - 53 GWD 10/25/99
sentries by

Pages 54-64 entries by GW 12/28/99 GW

Dec 28 1999

It took several trials to get the m20 model to run to steady state. The set of input files that were used for the successful run are:

m22.dat
h22.dcm
m21a.int

in addition to m20.con, m20.phk, and m20.bc, which are also listed on p.50.

Thermal Loading

The repository-axis elements were selected as based on nodal points (element centroids) that lie within a 10-m zone centered on the repository axis, i.e., centroid lies at Z coordinate of -434 ± 5 m. The elements that satisfy this criterion are listed on p. 55 in order of X-coordinates. Thermal load was applied to elements in this list that lies within $X = 74.3$ and $X = 1296.4$ (p. 29) with a center-to-center spacing of about 81 m between them. Such elements are circled in the list on p. 55.

repoLine.out

Node	xCoord	zCoord	ΔX (m)	A (m²)
1655	14.58	-431.89		
1656	43.73	-435.31		
1657	72.89	-438.74		
1610	102.19	-431.43	87.9	314
1611	131.39	-434.83		
1612	160.59	-438.23		
1565	190.05	-430.81	88.1	317
1566	219.29	-434.18		
1567	248.53	-437.56		
1520	278.16	-430.04	88.3	318
1521	307.43	-433.38		
1522	336.71	-436.73		
1475	366.46	-429.34	88.5	313
1476	395.77	-432.69		
1477	425.09	-436.04		
1430	454.92	-428.97	88.7	308
1431	484.27	-432.35		
1432	513.62	-435.73		
1386	572.96	-432.17	88.7	306
1387	602.35	-435.58		
1341	661.86	-432.16	88.9	304
1342	691.27	-435.59		
1296	750.95	-432.31	89.1	303
1297	780.40	-435.77		
1250	810.76	-429.15		286
1251	840.24	-432.63	89.3	
1252	869.72	-436.12		289
1205	900.21	-429.60		
1206	929.73	-433.12	89.5	
1207	959.24	-436.63		292
1160	989.87	-430.22		
1161	1019.42	-433.77	89.7	
1162	1048.96	-437.31		295
1115	1079.72	-431.01		
1116	1109.30	-434.58	89.9	
1117	1138.88	-438.15		283
1070	1169.69	-432.41		
1071	1199.31	-436.02	91.2	
1024	1230.13	-430.88		257
1025	1259.77	-434.55		
1026	1289.41	-438.21		
978	1290.68	-429.57		
979	1320.35	-433.29		
980	1350.02	-437.01		
933	1381.05	-432.24		
934	1410.75	-436.01		
3115	1445.18	-429.76		
3116	1480.96	-433.73		
3117	1516.74	-437.70		
3093	1517.98	-429.45		
3094	1553.70	-433.42		
3095	1589.42	-437.40		
3071	1590.54	-429.15		
3072	1626.21	-433.12		
3073	1661.87	-437.10		
3050	1698.48	-432.84		
3051	1734.09	-436.82		
3028	1770.53	-432.56		
3029	1806.08	-436.54		
3006	1842.54	-430.02		
3007	1878.01	-434.00		
3008	1913.49	-437.98		
2985	1949.80	-429.15		
2986	1985.19	-433.12		
2987	2020.58	-437.09		
2965	2091.84	-432.18		
2966	2127.14	-436.14		
2944	2197.96	-431.18		
2945	2233.18	-435.13		

Heat source applied to circled elements

The heat source magnitude is calculated as

$$q = \left(60 \frac{\text{MTU}}{\text{acre}}\right)\left(\frac{Q\ \text{W}}{70011\ \text{MTU}}\right)\left(\frac{1\ \text{acre}}{4047\ \text{m}^2}\right)(81\ \text{m})$$

where Q is the number of Watts for all 70,011 MTU (total repository waste load), which is a function of time as shown on p.56. The quantity q calculated above, unit of W/m, is the heat output per unit drift length of one drift. The proposed drift design is a circular section of 5.5-m diameter, that is a cross-sectional area of 23.8 m². On the other hand, there are no drifts in the model.

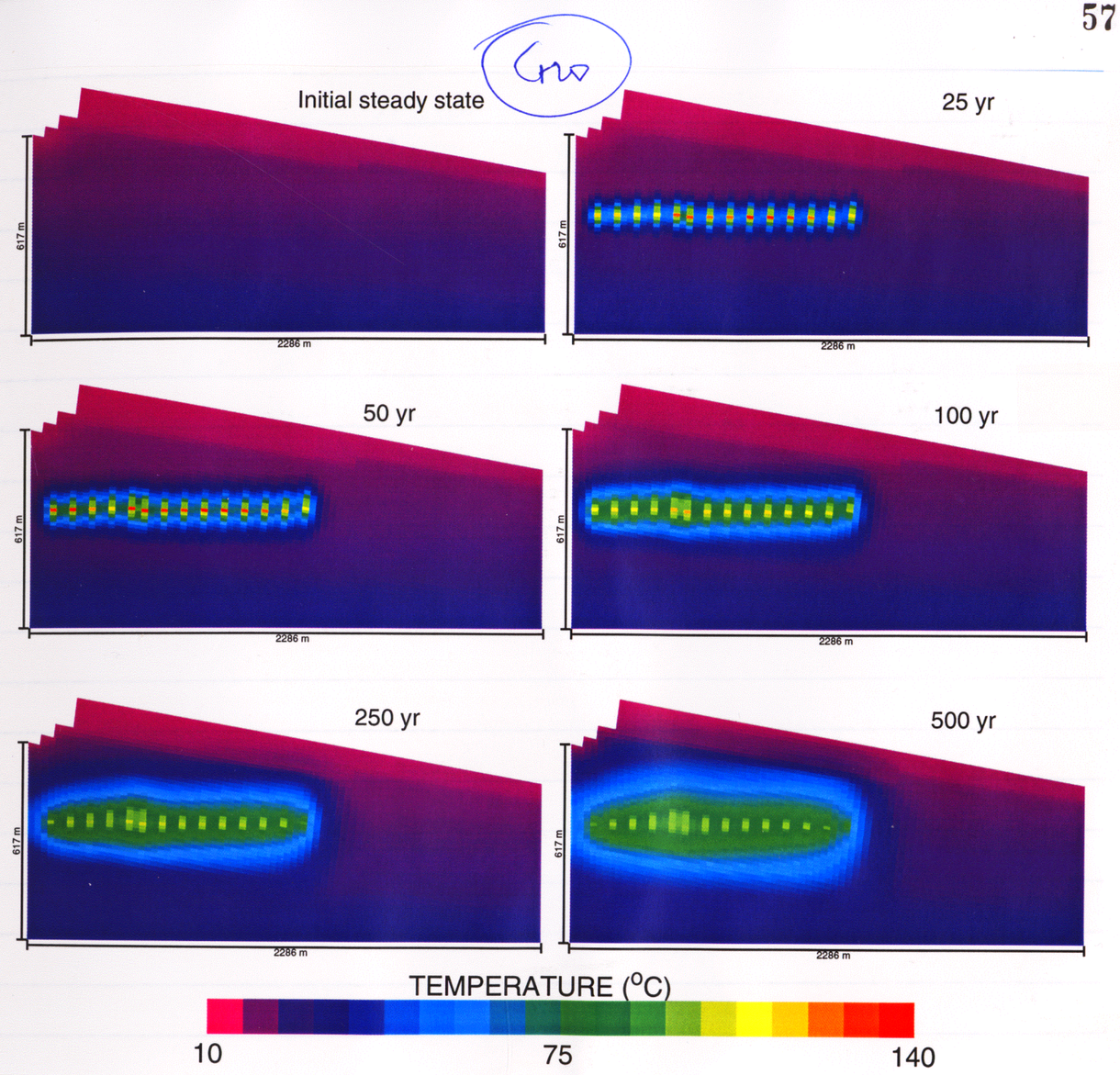
10^6 Q

Table V-1. Average Thermal Decay for All WPs

(A)	(B) (Sec. 4.3.23)	(C) (Sec. 4.3.23)	(D) (Sec. 4.3.23)	(E) (Sec. 4.3.23)	(F) (B)+(C)+(D)+(E)	(G) (F)/ 71.499486 x100%
Year	Total Heat from 2,859 44-BWR Pkgs (MW)	Total Heat from 4,137 21-PWR Pkgs (MW)	Total Heat from 3,259 DHLW Pkgs (MW)	Total Heat from 683 12-PWR Pkgs (MW)	Total Heat from All (10,938) Waste Pkgs (MW) (Total Initial Heat)	Percent of Total Initial Heat
0	21.423086	37.436260	4.875464	7.764676	71.499486	100.00%
5	19.017122	33.491405	4.399650	6.610624	63.518801	88.84%
10	17.301296	30.536741	3.923836	5.890347	57.652220	80.63%
20	14.643423	25.862371	3.252482	4.839801	48.598077	67.97%
30	12.580315	22.215593	2.581128	4.051793	41.428829	57.94%
40	10.962790	19.329458	2.260442	3.448308	36.000998	50.35%
50	9.651849	16.991320	1.939757	2.961375	31.544301	44.12%
60	8.602695	15.112725	1.619071	2.577586	27.912077	39.04%
70	7.755189	13.588046	1.298386	2.268359	24.909980	34.84%
80	7.055883	12.335578	0.977700	2.017372	22.386533	31.31%
90	6.494939	11.315100	0.902091	1.818288	20.530418	28.71%
100	6.012681	10.456422	0.826482	1.646766	18.942351	26.49%
150	4.567073	7.878865	0.448438	1.145978	14.040354	19.64%
200	3.824607	6.569736	0.196061	0.909198	11.499602	16.08%
300	3.038994	5.220269	0.083430	0.682267	9.024960	12.62%
400	2.560390	4.412083	0.052579	0.560346	7.585398	10.61%
500	2.219442	3.838502	0.034363	0.482585	6.574892	9.20%
600	1.943922	3.378038	0.030191	0.422659	5.774810	8.08%
700	1.728933	3.016950	0.026020	0.377363	5.149266	7.20%
800	1.542874	2.706392	0.021848	0.339467	4.610581	6.45%
900	1.394818	2.458149	0.017677	0.309170	4.179814	5.85%
1,000	1.270218	2.248570	0.014340	0.284115	3.817243	5.34%
2,000	0.659264	1.219725	0.009370	0.160481	2.048840	2.87%
3,000	0.505804	0.954557	0.008588	0.127461	1.596410	2.23%
4,000	0.449766	0.853526	0.007806	0.114076	1.425174	1.99%
5,000	0.418991	0.795874	0.007034	0.106206	1.328105	1.86%
6,000	0.387847	0.736906	0.006747	0.097886	1.229386	1.72%
7,000	0.363303	0.690293	0.006460	0.091349	1.151405	1.61%
8,000	0.340257	0.646400	0.006173	0.085211	1.078041	1.51%
9,000	0.321149	0.609974	0.005887	0.080141	1.017151	1.42%
10,000	0.304807	0.578761	0.005602	0.075639	0.965009	1.35%

File:\123r4wt1997-tyAttach-5.wk4 hy1/

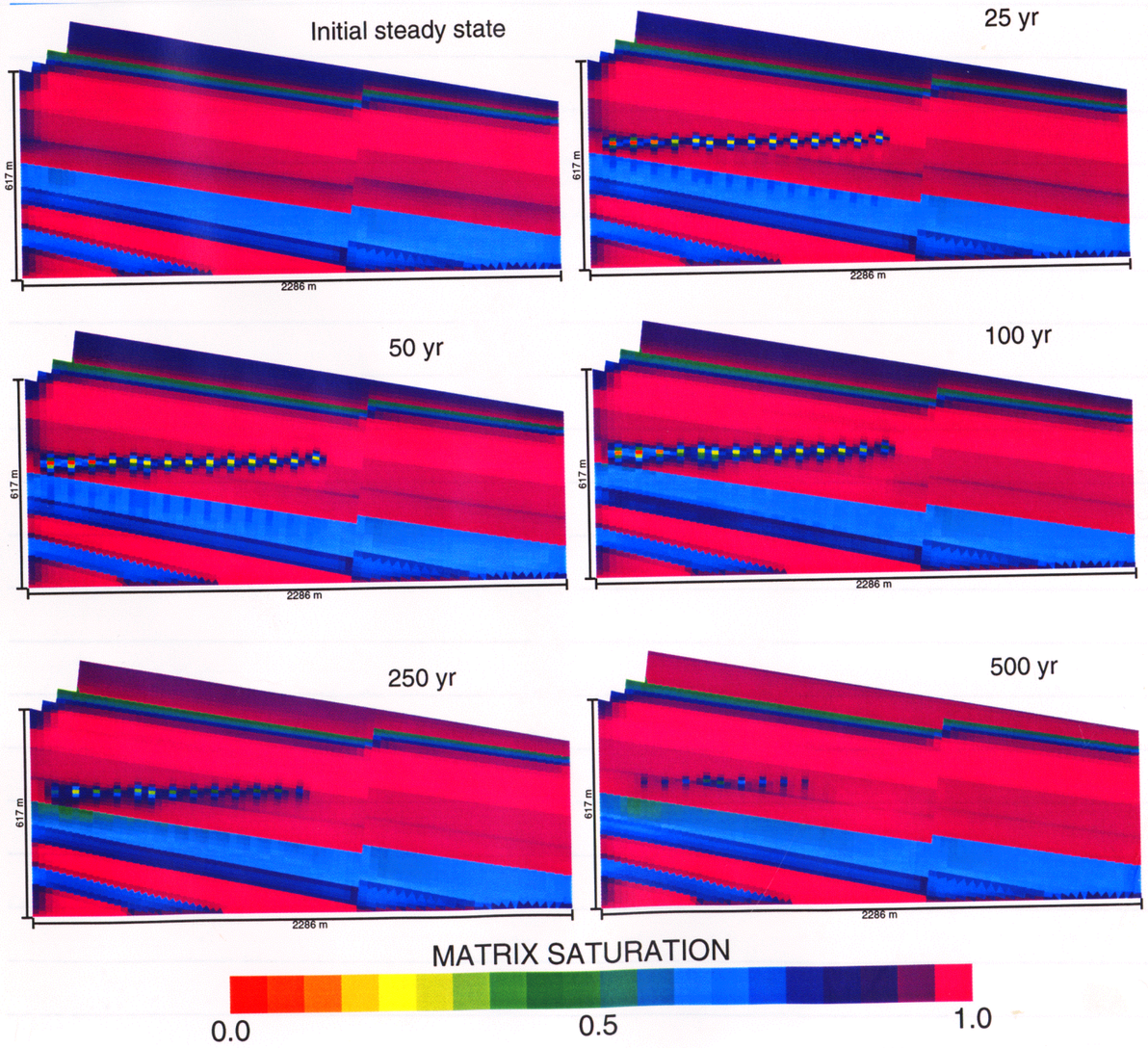
The area A of each of the (G10 12/26/99) elements over which the drift heat load q was applied is shown in the table on p. 55. As the table shows, each of the areas exceeds the drift cross-sectional area by a factor of about ~~12 300/25~~ (G10) 300 : 25, i.e., 12.

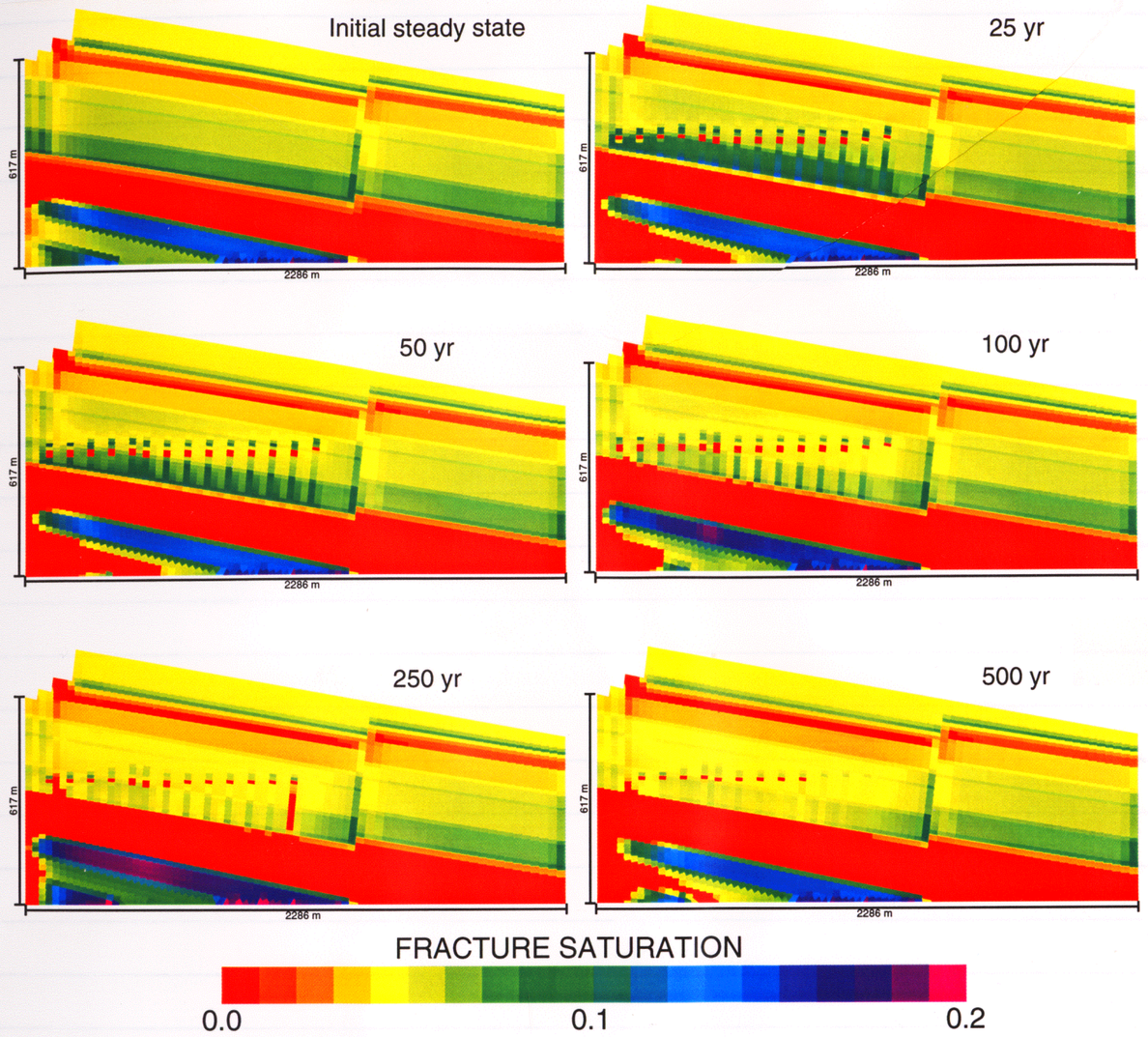


The results from model m22 (no heat) and h22 (m22 with heat) are presented on p. 57-59 in terms of contours of temperature and matrix and fracture saturations at the initial steady state (from m22) and at 25, 50, 100, 250, and 500 yr following waste emplacement (from h22). These

Contour plots were produced using Mathematica 4. The Mathematica macro (collection of function calls) is documented on p. 60-61.

To examine the hydrologic response in terms of the percolation-flux distributions, further processing of the METRA output is undertaken as follows. These steps are necessary





because METRA's flux output for unstructured grids is given in terms of node-to-node fluxes identified with reference to nodal connection numbers. The orientation and spatial location of the fluxes are not given. The post-processing steps for fluxes are (contd on p. 61, top-right corner):

m22SteadyState.nb Page 2 of 3

```
mntempf = 10.

General::spell1 : Possible spelling error: new symbol name "mntempf" is similar to existing symbol "mxtempf".

10.

mxfsat = 0.2

0.2

mnfsat = 0.

General::spell1 : Possible spelling error: new symbol name "mnfsat" is similar to existing symbol "mxfsat".

0.

tmp =
  OpenRead["\\\\Vulcan\\home\\ofoegbu\\HydrTherm\\RepoScale\\E200\\m22fldm4.xyp"];
Read[tmp, {String, String}]
data = ReadList[tmp, Number];
data = Partition[data, 13];
{!TITLE= 1.3305E+05 y,
 ! x      press     temp      sl      sg     xairl
   xairg   pcap    rel. h    psat    rhol    rhog     por}
data = Transpose[data];
msat = data[[4]];
tempm = data[[3]];
pressm = data[[2]];

General::spell1 : Possible spelling error: new symbol name "msat" is similar to existing symbol "fsat".

General::spell1 : Possible spelling error: new symbol name "tempm" is similar to existing symbol "tempf".

General::spell1 : Possible spelling error: new symbol name "pressm" is similar to existing symbol "pressf".

Min[pressm]

85385.

Max[pressm]

92540.

mxmsat = 1.0

General::spell1 : Possible spelling error: new symbol name "mxmsat" is similar to existing symbol "mxfsat".

1.
```

Result file. ← name changes

m22SteadyState.nb Page 1 of 3

■ Read and Display Results: m22SteadyState

```
<< Graphics`Legend`

tmp = OpenRead["\\\\Vulcan\\home\\ofoegbu\\aMesh\\segmt.m200"];
data = ReadList[tmp, {Number, Number, Number, Number, Number, Number, String}];
data = Transpose[ Drop[Transpose[data], -1]];
nodes = Flatten[Make[Transpose[data], -1]];
segmt = Partition[Flatten[Transpose[Drop[Transpose[data], -2]]], 2];
n = 1;
j = 1;
poly = Table[0, {Max[nodes]}, {100}];
Do[
   If[nodes[[i]] == nodes[[i + 1]],
      poly[[n, ++j]] = segmt[[2*i - 1]];
      poly[[n, ++j]] = segmt[[2*i]],
   n++;   j = 1],
   {i, 1, Length[nodes] - 1}]

poly = DeleteCases[poly, 0, 2];

tmp =
  OpenRead["\\\\Vulcan\\home\\ofoegbu\\HydrTherm\\RepoScale\\E200\\m22fldf4.xyp"];
Read[tmp, {String, String}]
data = ReadList[tmp, Number];
data = Partition[data, 13];
{!TITLE= 1.3305E+05 y,
 ! x      press     temp      sl      sg     xairl
   xairg   pcap    rel. h    psat    rhol    rhog     por}
data = Transpose[data];
fsat = data[[4]];
tempf = data[[3]];
pressf = data[[2]];

Min[pressf]

85385.

Max[pressf]

92540.

mxtempf = 140.

140.
```

File name changes depending on the result to be plotted.

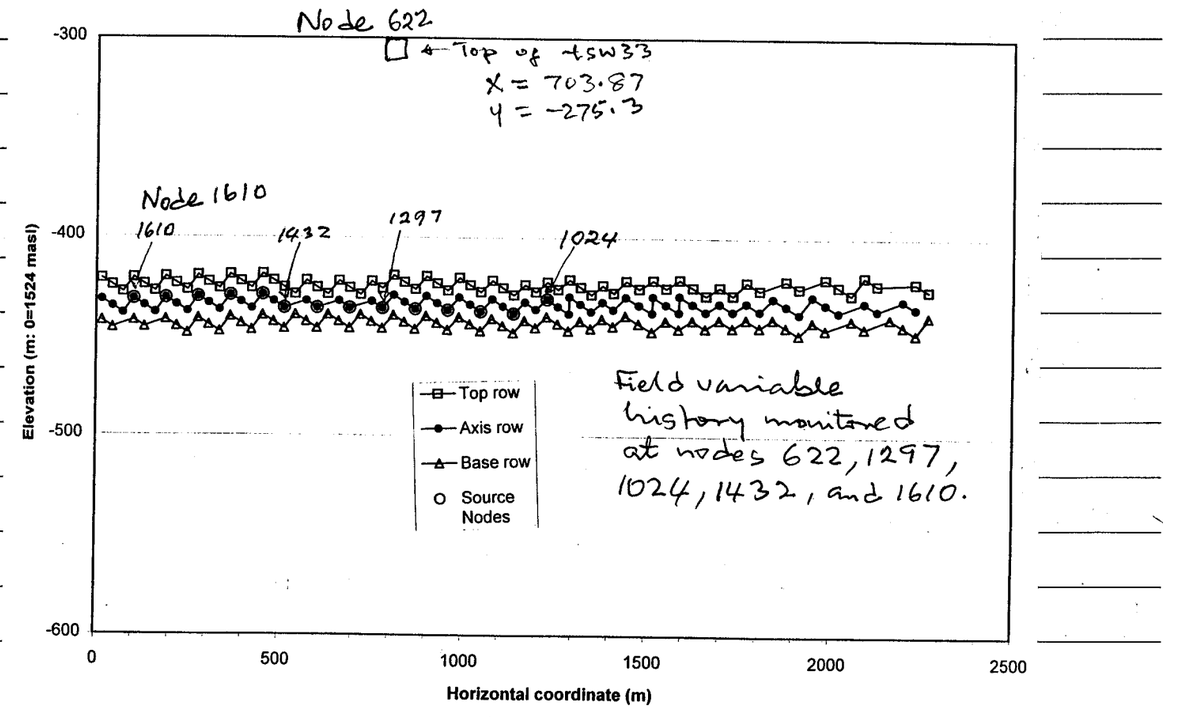
m22SteadyState.nb Page 3 of 3

```
mnmsat = 0.

General::spell :
 Possible spelling error: new symbol name "mnmsat" is similar to existing symbols {mnfsat, mxmsat}.

0.

mxtempm = 140.

General::spell1 : Possible spelling error: new symbol name "mxtempm" is similar to existing symbol "mxtempf".

140.

mntempm = 10.

General::spell :
 Possible spelling error: new symbol name "mntempm" is similar to existing symbols {mntempf, mxtempm}.

10.

pressm = pressm / Max[pressm];
pressf = pressf / Max[pressf];
tempm = (tempm - mntempm) / (mxtempm - mntempm);
tempf = (tempf - mntempf) / (mxtempf - mntempf);
fsat = (fsat - mnfsat) / (mxfsat - mnfsat);
msat = (msat - mnmsat) / (mxmsat - mnmsat);

obj1 = Graphics[Table[{Hue[(1 - tempm[[i]]) * 0.8], Polygon[poly[[i]]]},
    {i, 1, Length[fsat]}], AspectRatio → 0.5];
obj2 = Graphics[Table[{Hue[(fsat[[i]]) * 0.8], Polygon[poly[[i]]]},
    {i, 1, Length[fsat]}], AspectRatio → 0.5];
obj3 = Graphics[Table[{Hue[(msat[[i]]) * 0.8], Polygon[poly[[i]]]},
    {i, 1, Length[fsat]}], AspectRatio → 0.5];

ShowLegend[Show[obj3, DisplayFunction → Identity],
 {Hue[ 0.8 (1 - #)] &, 20, "1.0", "0", LegendShadow → None}]

ShowLegend[Show[obj2, DisplayFunction → Identity],
 {Hue[ 0.8 (1 - #)] &, 20, "0.2", "0", LegendShadow → None}]

ShowLegend[Show[obj1, DisplayFunction → Identity],
 {Hue[ 0.8 (#)] &, 20, "140", "10", LegendShadow → None}]
```

(1) Select a set of nodes to define the line for which flux profile is needed. For example, the figure below shows the node sets that define (approximately) the repository axis and two horizontal lines above and below the repository axis.

(2) Obtain the connectivity information for these nodes from

Repository-axis nodes and neighbors



the node-connectivity data file, e.g., m20.con (p.50).

(3) Extract matrix and fracture fluxes from METRA job_out file (job is user-specified job name). The extraction was performed using the code documented on the right side of this page. The section of the job_out file that contains the required velocities is first extracted using a text editor, thereafter the output from the text editor is passed as input the the code on the right side of this page.

D:\HydrTherm\RepoScale\E200\vFromOutFile.c

```
#include <stdio.h>
#include <stdlib.h>

main()
{
  char buf[100];
  int n,i,numCon,numNodes;
  float junk,vL,vG;
  float* mLv;
  float* mGv;
  float* fLv;
  float* fGv;
  FILE* Fout;
  char* outFile = "m32Vel.out";

  numNodes = 3708;
  numCon = 9626;
  mLv = (float*) malloc(numCon*sizeof(float));
  mGv = (float*) malloc(numCon*sizeof(float));
  fLv = (float*) malloc(numCon*sizeof(float));
  fGv = (float*) malloc(numCon*sizeof(float));
  if (!mLv || !mGv || !fLv || !fGv){
    printf("Memory allocation error\n");
    exit(1);
  }
  Fout = fopen(outFile,"w");
  if (!Fout){
    printf("Unable to open file %s\n",outFile);
    exit(1);
  }

/* Read and discard numNodes lines of matrix-to-fracture velocity */

  while(gets(buf))
    if (sscanf(buf,"%d %f %f %f",&n,&junk,&vL,&vG) == 4)
      if (n == numNodes){
        printf("All matrix-to-fracture velocity read and discarded\n");
        break;
      }

/* Read numCon lines node-to-node fracture velocity */

  for (i=0; i<numCon; i++){
    if (
      (!gets(buf)) ||
      (sscanf(buf,"%d %f %f %f",&n,&junk,&vL,&vG) != 4)
      ){
      printf("Invalid input for fracVel data: Connection number %d\n",i+1);
      exit(1);
    }
    fLv[i] = vL;
    fGv[i] = vG;
  }

/* Read numCon lines of node-to-node matrix velocity */

  for (i=0; i<numCon; i++){
    if (
      (!gets(buf)) ||
      (sscanf(buf,"%d %f %f %f",&n,&junk,&vL,&vG) != 4)
      ){
      printf("Invalid input for matVel data: Connection number %d\n",i+1);
      exit(1);
    }
    mLv[i] = vL;
    mGv[i] = vG;
  }

/* Print the node-to-node velocities */

  fprintf(Fout,"%10s%12s%12s%12s%12s\n","conNum","mLiqVel","mGasVel",
    "fLiqVel","fGasVel");
  for (i=0; i<numCon; i++)
    fprintf(Fout,"%10d%12.3E%12.3E%12.3E%12.3E\n",i+1,mLv[i],mGv[i],
      fLv[i],fGv[i]);
}
```

# of connections in model.

# of nodes in model.

(4) The node-connectivity list obtained in Step (2) and the output from Step (3) are passed to the code LigFlux that is documented on p. 63-64. The output from LigFlux is a profile of water flux in mm/yr. The direction of a flux is the same as the direction of the node connection for which the flux is given.

12/28/99 END

LigFlux.cpp Page 1 of 4

```
//.....
// LigFlux
//
// This code calculates total liquid flux in mm/yr from METRA output
// of fracture and matrix fluxes in m/s. The flux direction is from
// node to node. Nodal connectivity and fracture porosity, which are
// required for the calculations, are read from files.
//
// Modify the values of the following variables as necessary:
//
//   dcmFile      DCMPAR input block for metra
//   fluxFile     Given matrix and fracture fluxes from METRA
//   conFile      Defines node connectivity for a set of nodes
//   outFile      Output file
//   minConNum    Minimum connection index in conFile
//   maxConNum    Maximum connection index in conFile
//
// Author:        G. I. Ofoegbu
// Date:          December 1999
// System:        C++ compiler
//                (Code developed using Borland C++ Builder 4)
//.....

#include <stdio
#include <iomanip>
#include <iostream>
#include <fstream>
#include <string>
#include <stdlib>
#include <math>
#include <strings>

#pragma hdrstop
#include <condefs>

using namespace std;

char* messageBuffer;
void DumpAndQuit();

int ERRORBEND = 1;
int NORMALEND = 0;

// char* dcmFile = "M32Cases\\m32.dcm";
char* dcmFile = "d:\\HydrTherm\\RepoScale\\E200\\M32Cases\\M22.dcm";

struct ConData{
  int index;
  int node1;
  int node2;
  float x;
  float mLiqVel;
  float mGasVel;
  float fLiqVel;
  float fGasVel;
  ConData* next;
};
int GetConData(char* fileName, ConData** first, ConData** last);
void StoreConData(ConData* current, ConData** first, ConData** last);
void SaveFluxData(ConData* current, int conNum, float mLiqVel,
  float mGasVel, float fLiqVel, float fGasVel);
int GetFracPor(int node, float& fPor);
int PrintConData(ConData* first, char* outFile);

//.....
#pragma argsused
int main(int argc, char* argv[])
{
  char* fluxFile = "d:\\HydrTherm\\RepoScale\\E200\\M22Cases\\m22Vel.out";

  char* conFile = "d:\\HydrTherm\\RepoScale\\E200\\PtnVFlx.con";
  char* outFile = "d:\\HydrTherm\\RepoScale\\E200\\M22Cases\\M22PtnVFlx.fix";
  int minConNum = 954;
  int maxConNum = 1068;

  // Allocate memory for possible error messages

  messageBuffer = new char [151];
  if (!messageBuffer){
    cerr << "Memory allocation error for messageBuffer\n";
    cerr << "Press any key to end: ";
    getch();
    return (1);
  }
```

LigFlux.cpp Page 2 of 4

```
  // Setup the connections list defining the top and bottom rows of nodes
  // for which vertical flux data is required
  ConData* firstCon = 0;
  ConData* lastCon = 0;
  cout << "Setting up connections data ...\n";
  if (GetConData(conFile,&firstCon,&lastCon)){
    DeleteConData(firstCon);
    DumpAndQuit();
  }

  // Read flux data from file and extract the flux data for the
  // connections within the connections list that was setup previously
  ifstream Fn(fluxFile);
  if (!Fn){
    sprintf(messageBuffer,"Unable to open file %s",fluxFile);
    DeleteConData(firstCon);
    DumpAndQuit();
  }

  float mLiqVel,mGasVel,fLiqVel,fGasVel;
  char buf[1000];
  size_t nBuf = sizeof(buf);
  int conNum = 0;

  cout << "Adding flux to database ...\n";
  while (!Fn.eof()){
    Fn.getline(buf,nBuf);
    istrstream inpLine(buf,nBuf);
    inpLine >> conNum >> mLiqVel >> mGasVel >> fLiqVel >> fGasVel;
    if (conNum >= minConNum && conNum <= maxConNum)
      SaveFluxData(firstCon,conNum,mLiqVel,mGasVel,fLiqVel,fGasVel);
  }

  cout << "\nAdding total liquid velocities to database ...\n";
  cout << "Fracture porosities from file " << fluxFile << " ...\n";
  if (!PrintConData(firstCon,outFile)){
    DeleteConData(firstCon);
    DumpAndQuit();
  }

  cout << "\nPrinting results to file " << outFile << " ...\n";
  if (PrintConData(firstCon,outFile) == ERRORBEND){
    DeleteConData(firstCon);
    DumpAndQuit();
  }

  // Done

  DeleteConData(firstCon);
  delete[] messageBuffer;
  cout << "\nDone ... press any key to end:";
  getch();
  return 0;
}

int GetConData(char* fileName, CondData** first, CondData** last)
{
  int conIndex,node1,node2;
  float x;
  char buf[1000];
  size_t nBuf = sizeof(buf);

  ifstream Fn(fileName);
  if (!Fn){
    sprintf(messageBuffer,"Unable to open file %s",fileName);
    return (ERRORBEND);
  }

  while (!Fn.eof()){
    Fn.getline(buf,nBuf);
    istrstream inpLine(buf,nBuf);
    inpLine >> conIndex >> node1 >> node2 >> x;
    if (!conIndex) continue;
    CondData* current = new CondData(conIndex,node1,node2,x);
    if (!current){
      sprintf(messageBuffer,"Memory allocation failure");
      return (ERRORBEND);
    }
    StoreConData(current,first,last);
  }
  return (NORMALEND);
}

void StoreConData(CondData *current, CondData **first, CondData **last)
{
  if (!*last){
    *last = current;
    *first = current;
    return;
  }
  CondData *p = *first;

  // Open output file
```

LigFlux.cpp Page 3 of 4

```
  while (p){
    if ((current->x) > (p->x)){
      old = p;
      p = p->next;
    }
    else{
      if (old) {  // insert current at this point in the list */
        old->next = current;
        current->next = p;
        return;
      }
      current->next = p;  /* current becomes new first element */
      *first = current;
      return;
    }
  }
  first = first->next;
  while (first){
    if (!first->next){  /* current becomes new last element */
      *last = current;
      first = 0;
    }
    else first = first->next;
  }
}

int TotalLiqVelocity(ConData* first)
{
  float meterPr = 365.25*24.*60.*60.;
  float fPor = 1000.;
  float fPor1;
  float fPor2;

  float count = 0.0;
  int numDone = count;
  while (first){
    if (
      (GetFracPor(first->node1,fPor1) == ERRORBEND) ||
      (GetFracPor(first->node2,fPor2) == ERRORBEND)
    ) return (ERRORBEND);
    float fPor = 0.5*(fPor1 + fPor2);
    float mVel = first->mLiqVel;
    float fVel = first->fLiqVel;
    first->liqVel = (mVel + fVel)*secPerYr*mmPerM;
    count += 1.0;
    numDone = count;
    if (fmod(count,10.) < 0.09)
      cout << "Working ... processed " << numDone << " connections\n";
    first = first->next;
  }
  cout << "Done ... processed " << numDone << " connections\n";
  return (NORMALEND);
}
int GetFracPor(int node, float& fPor)
{
  ifstream Fin(dcmFile);
  if (!Fin){
    sprintf(messageBuffer,"Unable to open file %s",dcmFile);
    return (ERRORBEND);
  }

  int n1,n2;
  float x;
  char buf[150];
  size_t nBuf = sizeof(buf);

  ifstream Fn(fileName);
  if (!Fn){
    sprintf(messageBuffer,"Unable to open file %s",fileName);
    return (ERRORBEND);
  }

  while (!Fn.eof()){
    Fn.getline(buf,nBuf);
    istrstream inpLine(buf,nBuf);
    inpLine >> n1 >> n2 >> x >> x >> x >> fPor;
    if ((n1==node) && (n2==node)) return (NORMALEND);
  }
  return (ERRORBEND);
}

int PrintConData(CondData *first, char* outFile)
{
  ofstream Fout(outFile);
  if (!Fout){
    sprintf(messageBuffer,"Unable to open file %s",outFile);
    return (ERRORBEND);
  }

  CondData *current = first;
  while (current){
    Fout << current->conIndex << " " << current->node1 << " "
         << current->node2 << " " << current->x << " "
         << current->mLiqVel << " " << current->mGasVel << " "
         << current->fLiqVel << " " << current->fGasVel << "\n";
    current = current->next;
  }
  return (NORMALEND);
}

int PrintConData(CondData *first, char* outFile)
{
  if (!first){
    messageBuffer = "Empty list passed to PrintConData";
    return (ERRORBEND);
  }
```

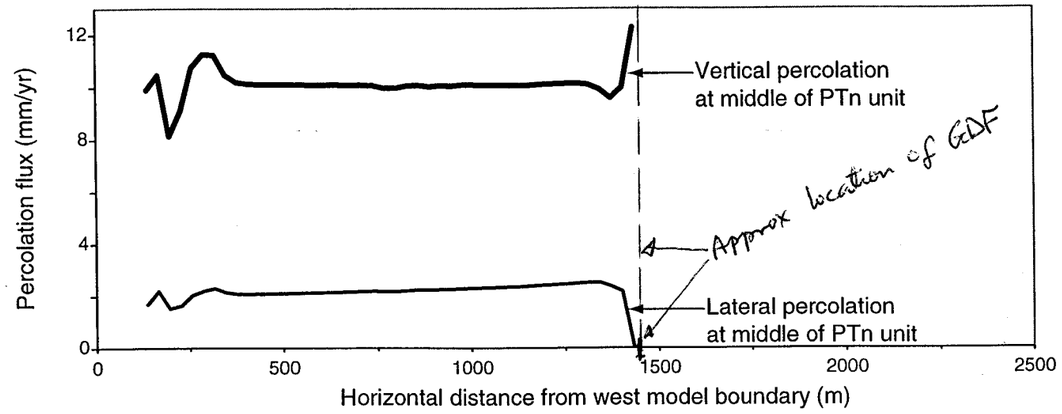
LiqFlux.cpp Page 4 of 4

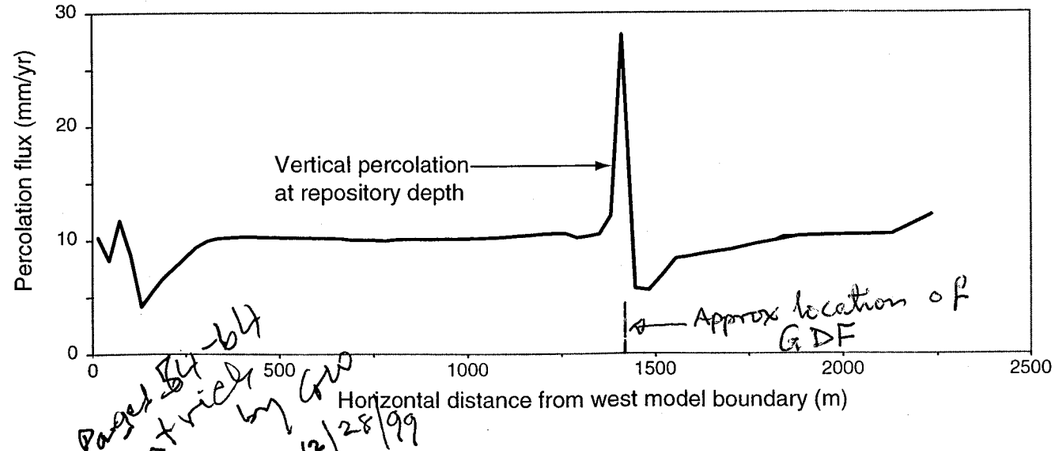
```
ofstream Fout(outFile);
if (!Fout){
  sprintf(messageBuffer,"Unable to open file %s",outFile);
  return (ERROREND);
}
Fout << setw(8) << "ConNum"
     << setw(8) << "Node1"
     << setw(8) << "Node2"
     << setw(10) << "xCoord"
     << setw(15) << "LiqVel mm/yr"
     << setw(14) << "mLiqVel m/s"
     << setw(14) << "mGasVel m/s"
     << setw(14) << "fliqVel m/s"
     << setw(14) << "fGasVel m/s"
     << endl;

while (first){
  Fout << setw(8) << first->index
       << setw(8) << first->node1
       << setw(8) << first->node2
       << setiosflags(ios::fixed) << setprecision(2)
       << setw(10) << first->x
       << resetiosflags(ios::fixed)
       << setiosflags(ios::scientific) << setprecision(4)
       << setw(15) << first->liqVel
       << setw(14) << first->mLiqVel
       << setw(14) << first->mGasVel
       << setw(14) << first->fLiqVel
       << setw(14) << first->fGasVel
       << resetiosflags(ios::scientific)
       << endl;
  first = first->next;
}
return (NORMALEND);
}

void DeleteConData(ConData** first)
{
 ConData *p = *first;
 ConData *current = p;

 while (p){
   current = p;
   p = p->next;
   delete current;
 }
}

void DumpAndQuit()
{
 cerr << "\n" << messageBuffer << "\n";
 delete[] messageBuffer;
 cerr << "Press any key to end: ";
 getch();
 exit(1);
}
```

GW 12/28/99

Three profiles of percolation flux (one at repository depth and two along the middepth of PTn) are shown in the figure below. The profiles are from model m22. They illustrate lateral flow in the PTn, interruption of lateral flow at the GDF, and the effects on vertical percolation flux at the repository depth.

GW 12/28/99





Page 64 entries by GW 12/28/99

January 5 2000

History of Field Variables

Pages 65-66 for entries by GW 1/5/2000

The histories of temperature, fracture saturation, and matrix saturation are monitored at the nodes shown in the figure on p. 61, i.e., Nodes 622, 1024, 1297, 1432, and 1610. The history results are stored in METRA output file job_prn (where "job" is the user-assigned job name). They are extracted from this file (job_prn) and re-written in a form suitable for X-Y plotting using the C-code fvHist.c documented below.

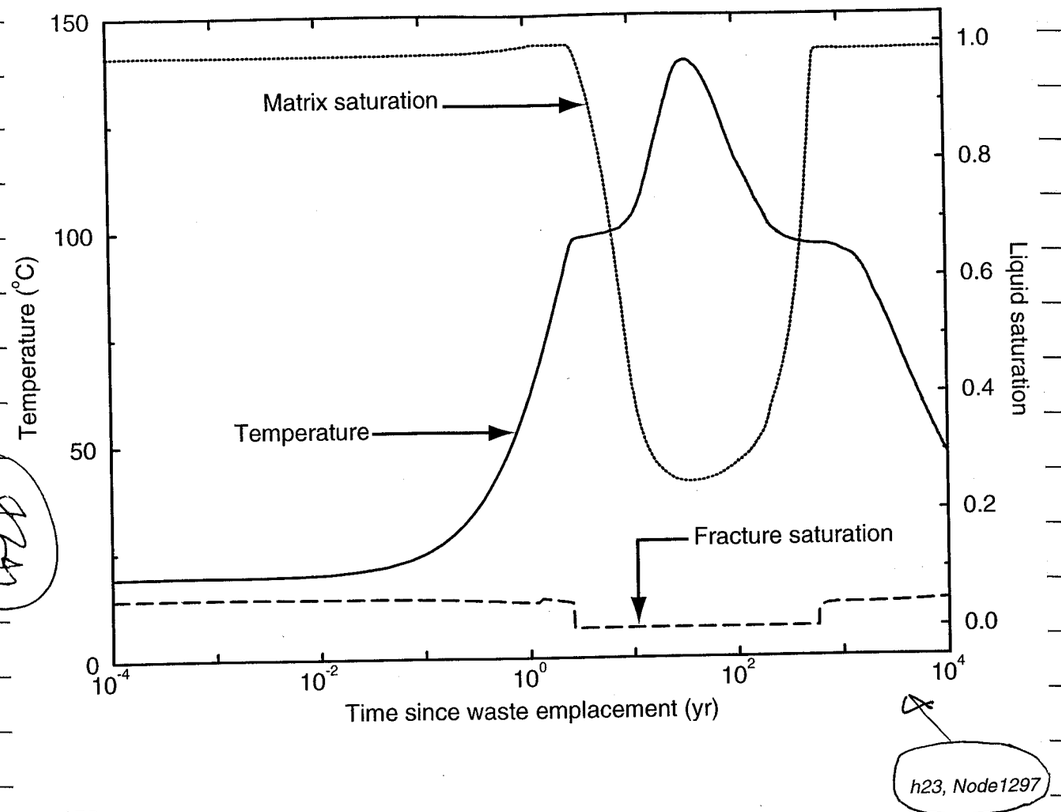
D:\HydrTherm\RepoScale\E200\fvHist.c

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

main()
{
  int nodes[] = {622,1024,1297,1432,1610};
  int i;
  char* inFileName = "h32_prn";

  for (i=0; i<5; i++)
    GetHistory(nodes[i],inFileName);
}

GetHistory(int node, char* inFileName)
{
  char buf[121];
  char* outFile = "h32N1234.his";
  int elem, step;
  float time, dt, p, temp, sg, pa;
  float msat,fsat,curTime,mTemp;
  FILE* Fin;
  FILE* Fout;

  int fNode = 2*node;
  int mNode = fNode - 1;

  Fin = fopen(inFileName,"r");
  if (!Fin)
    DumpAndQuit(strcat("Unable to open file ",inFileName));

  sprintf(outFile,"h32N%04d.his",node);
  Fout = fopen(outFile,"w");
  if (!Fout)
    DumpAndQuit(strcat("Unable to open file ",outFile));

  fprintf(Fout,"%%lls%12s%12s%12s\n","Time-yr","Temp-C","matSat","fracSat");

  while (fgets(buf,120,Fin))
    if (sscanf(buf,"%f %f %d %f %f %f %f %*s %d",&time,&dt,&elem,&p,&temp,&sg,&pa,&step)
        == 8)
      if (elem == mNode){
        msat = 1.0 - sg;
        curTime = time;
        mTemp = temp;
        fgets(buf,120,Fin);
        if(
         (sscanf(buf,"%f %f %d %f %f %f %f %*s %d",&time,&dt,&elem,&p,&temp,&sg,&pa,&step)
          != 8) ||
         (elem != fNode)
        ){
          sprintf(buf,"Invalid input found for node %d at time %.3E",node,curTime);
          DumpAndQuit(buf);
        }
        fsat = 1.0 - sg;
        fprintf(Fout,"%12.3E%12.2f%12.4f%12.4f\n",curTime,mTemp,msat,fsat);
      }

  fclose(Fin);
  fclose(Fout);
}
DumpAndQuit(char* s)
{
  printf("\n");
  puts(s);
  printf("\n");
  exit(1);
}
```

Modify node list as necessary

Change METRA output filename as necessary.

The results for nodes 1297 and 622 are shown on p. 66 (top figure: node 1297; bottom figure: node 622). The results are from model h23 (same as h22, except that h23 includes history monitoring whereas h22 does not).



h23, Node1297

h23, Node622

Pages 67-68 entered by GW 2/14/2000

February 14 2000

Investigate the effects of a minor fault that intersects the PTn without necessarily producing measurable offset (e.g. normal fault to the west of the GSDF on p. 29). The fault intersects the PTn at a horizontal coordinate of about ~~875-900~~ 840 – 880 m. GW 2/14/00 The following PTn nodes define the fault zone

353	← Centroid at X = 860.33 m
397	← ✓ ✓ ✓ 859.36 m
441	← ✓ ✓ ✓ 858.39 m

Three cases were analyzed (Cases m34, m35, and m36), which differ with respect to the applied modification of porosity, permeability and van Genuchten parameters relative to the base case m22, as shown below:

Parameter	m22 (base model) Value	m34 Value (ϕ/10 case)	m35 Value [(ϕ/2 + α/10) case]	m36 Value (ϕ/2 case)
Matrix porosity	0.369	0.0369	0.1845	0.1845
Matrix permeability (m^2)	1.61E-14	1.61E-16	4.025E-15	4.025E-15
Fracture porosity	4.84E-5	4.84E-6	2.42E-5	2.42E-5
Horizontal-fracture permeability (m^2)	5.25E-13	5.25E-16	6.5625E-14	6.5625E-14
Vertical-fracture permeability (m^2)	5.25E-13	5.25E-13	5.25E-13	5.25E-13
Matrix van Genuchten α (1/Pa)	3.43E-5	3.43E-5	3.43E-6	3.43E-5
Matrix van Genuchten λ	0.3859	0.3859	0.5	0.3859

Pages 65-66 entered by GW 1/5/2000

The input files for these cases are.

m34.dat
m34.phk
m34.dcm } for case m34

m35.dat
m35.phk
m35.dcm } for case m35

m36.dat
m36.phk
m36.dcm } for case m36

In addition, each of the three cases use the following files

m20.con	Connectivity data
m20.int	Initial conditions
m22.bc	Boundary conditions

Pages 67-68 by GW 2/24/2000 entries

February 24 2000 Pages 69-79 by GW 2/24/2000 entries

Investigate the effects of geomechanical altered zones at and close to the repository axis. Altered zone is laterally discontinuous, may extend up to 25 m above and below repository axis, and may be characterized by

(1) Increase in fracture porosity (aperture) from inelastic rock-mass response. Both horizontal and vertical fractures may be affected.

(2) Decrease in vertical-fracture aperture from elevated horizontal compressive stress.

Change in Fracture Porosity and Permeability

$$b_h = R_{bh}({}^0b_h)$$

$$b_v = R_{bv}({}^0b_v)$$

${}^0b_h$ and ${}^0b_v$ = Base-case horizontal and vertical fracture aperture.

b_h, b_v = applied horizontal and vertical fracture aperture

R_{bh}, R_{bv} = aperture-change ratio (model value ÷ base-model value) for horizontal and vertical fractures.

Using the following properties of the base model:

$^0b_v = {}^0b_h = {}^0b$ = base-model fracture aperture

$S_{fv} = S_{fh} = S_f$ = fracture spacing

it can be shown that the average change in fracture porosity is represented by the ratio $R_{\phi f}$ where

$$R_{\phi f} = \frac{1}{3}(2R_{bv} + R_{bh})$$

$R_{\phi f}$ = Model fracture porosity / Base-model fracture porosity

Also, using the cubic-law relationship between fracture aperture and permeability, it can be shown that the fracture-permeability change ratio is

$$R_{kh} = (R_{bh})^3$$

$$R_{kv} = (R_{bv})^3$$

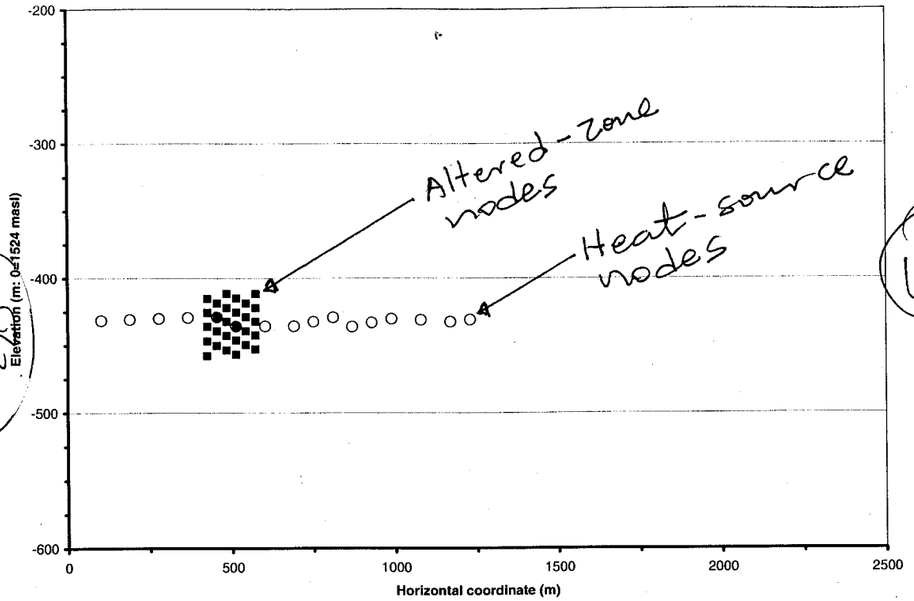
where

R_{kh} = Model horizontal-fracture permeability / Base-model horizontal-fracture permeability

and R_{kv} is defined similarly for vertical fractures.

Altered Zone Geometry and Properties

Nodes that lie within a rectangle defined by $x = 419 - 581$ m and $z = -459 - -409$ m were selected to define an altered zone that extends about 160 m horizontally and 25 m above and below the repository axis. The altered zone is shown in figure on p. 72.



Analysis Cases

6 cases were analyzed that represent different combinations of R_{bh} and R_{bv} as follows

Case	R_{bh}	R_{bv}
m41	10.0	1.0
m42	1.0	0.1
m43	$10^{0.5}$ (=3.1623)	1.0
m44	1.0	$10^{-0.5}$ (= 0.3162)
m45	10.0	10.0
m46	10.0	0.1

The input files for the various cases are

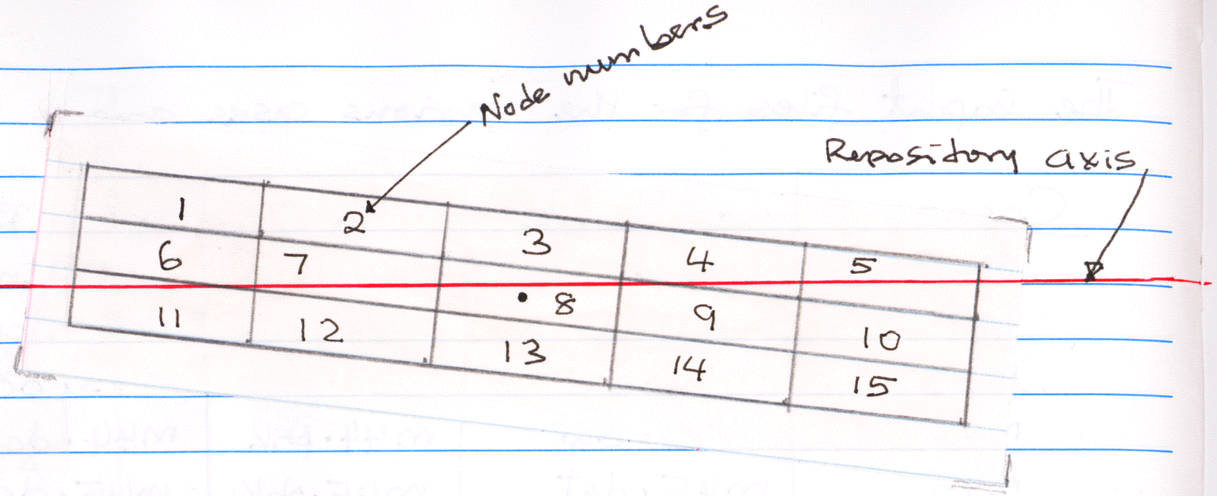
Case	.dat file	.phk file	.dcm file
m41	m41.dat	m41.phk	m41.dcm
m42	m42.dat	m42.phk	m42.dcm
m43	m43.dat	m43.phk	m43.dcm
m44	m44.dat	m44.phk	m44.dcm
m45	m45.dat	m45.phk	m45.dcm
m46	m46.dat	m46.phk	m46.dcm

In addition, each of the cases uses the following input files:

m20.con	Connectivity & data
m22.ini	Initial conditions
m22.bc	Boundary conditions.

Post-Processing for Percolation Flux Through Repository Axis

The procedure described on p. 58-64 for obtaining percolation fluxes relied only on vertical or (near-vertical) connections to or from the repository axis. Repository-level percolation flux was determined using vertical connections from the Top-row node set to the Axis-row node set (see figure on p. 61).



This procedure was modified to account for ~~all~~ fluxes along all connections that cross the repository axis, whether the connection is down-dip, dip-normal, or diagonal. For example, in the mesh shown above, node 8 would receive cross-repository fluxes along ~~three~~ a dip-normal connection (q_{38}), a down-dip connection (q_{78}) and a diagonal connection (q_{28}), where each q_{ij} represents the flux per unit area from node i to node j. The previous procedure used q_{38} and ignored q_{78} and q_{28} for such a node. The previous approach was satisfactory for models such as m22, in which flow was mostly vertical at the repository ~~axis~~ level. Significant lateral flow may cross the repository axis in models such as m41, . . . , m46. For such models, a significant error may result

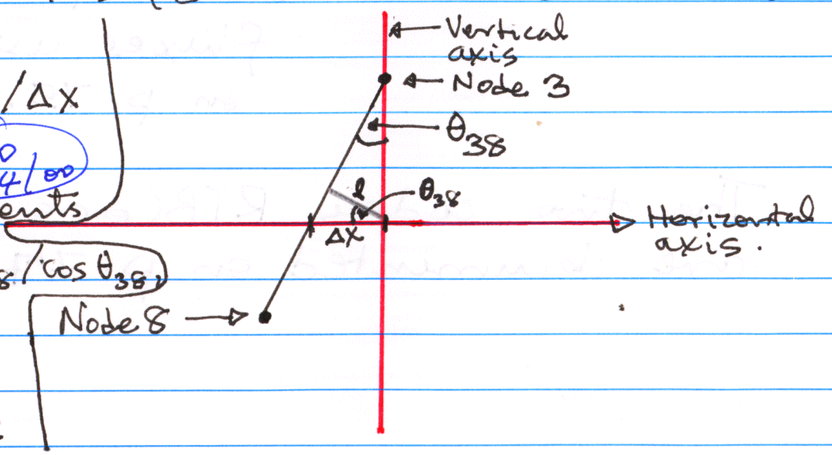
from neglecting non-normal ~~non-vertical~~ cross-repository fluxes such q_{78} and q_{28}. To avoid such errors, the vertical flux component crossing the repository axis into a node (such as node 8) was calculated as follows

$$q_{v8} = \frac{(A_{78} q_{78} + A_{28} q_{28} + A_{38} q_{38})}{A_{38}/(\cos \theta_{38})}$$

where: (1) node 8 is a generic node below the repository axis that receives flux directly from one or more nodes above the repository axis; (2) A_{38} is the area normal to the dip-normal (near-vertical) connection ~~towards~~ to the generic node; and (3) θ_{38} is the angle of the dip-normal connection, measured clock-wise from the vertical axis (shown in the sketch below)

$\cos \theta_{38} = \ell / \Delta x$

$\therefore \Delta x$ represents the area $A_{38}/\cos \theta_{38}$, which is the horizontal projection of A_{38}.



The area $A_{38}/(\cos\theta_{38})$ is the horizontal projection of A_{38}, i.e., the area of node 8 normal to cross-repository vertical flux arriving at node 8.

This calculation procedure (described from p. 75) was implemented through two codes: (GW 2/24/02)

RTBCons sorts out all direct connections from a node above the repository axis to a node below the axis, obtains areas Aij and connection cosines $\cos\theta_{ij}$

RTBFlux Uses the outputs from RTBCons and from code on p. 62 to calculate cross-repository vertical fluxes using the equation on p. 75.

The two codes RTBCons and RTBFlux are documented on p. 77 – 79.

RTBCons p1 of 3

```
/*****************************************************************************
This code extracts the connections data for nodal connections that cross
the repository axis (i.e., connections from a node above the repository
axis to a node below the repository axis). Connections data are read from
METRA's CONNE input block (given in a named file). List of nodes above
and below the repository axis are read from files prepared by code
REPONODES. Modify the values of the following variables as necessary:

conneFile       file containing METRA CONNE input block for model
topFile         list of nodes above repository axis
baseFile        list of nodes below repository axis
outFile         output file for across-axis connections data
nMin1           nMin1, nMax1, nMin2, and nMax2 define the range
nMax1           of node numbers in the file topFile
nMin2
nMax2
zAtRepoLine     z-Coordinate at the repository axis

Author:         G. I. Ofoegbu
Date:           February 2000
System:         C++ compiler
                (Code developed using Borland C++ Builder 4)
*****************************************************************************/

#include <stdio>
#include <iostream>
#include <iomanip>
#include <fstream>
#include <strstrea>
#include <conio>
#include <stdlib>
#include <math>
#include <string>

#pragma hdrstop
#include <condefs>

using namespace std;

char* messageBuffer;
void DumpAndQuit();

int ERROREND = 1;
int NORMALEND = 0;

struct NdData{
  int node;
  float x;
  float z;
  NdData* next;
  NdData(int n, float xc, float zc){
    node = n;
    x = xc;
    z = zc;
    next = 0;
  }
};

struct ConData{
  int index;
  int node1;
  int node2;
  float x;
  float area;
  double cTheta;
  ConData* next;
  ConData(int nc, int n1, int n2){
    index = nc;
    node1 = n1;
    node2 = n2;
    next = 0;
  }
};

int GetNodes(char* fileName, NdData** firstNode);
NdData* FindNode(NdData* firstNode, int nodeNum);
float RepoLine(float xTop,float xBase,float zTop,float zBase,float z0);
double ConneCosine(float xTop,float xBase,float zTop,float zBase);
void StoreCon(ConData* current, ConData** first, ConData** lastNode);
int PrintConList(ConData* first, char* outFile);
void DeleteNodes(NdData** firstNode);
void DeleteConList(ConData** firstCon);

//---------------------------------------------------------------------------
#pragma argsused
int main(int argc, char* argv[])
{
  char* conneFile = "d:\\HydrTherm\\RepoScale\\EastWest01\\m20.con";
  char* topFile = "d:\\HydrTherm\\RepoScale\\E200\\rTopZone.out";
  char* baseFile = "d:\\HydrTherm\\RepoScale\\E200\\rBaseZone.out";
  char* outFile = "d:\\HydrTherm\\RepoScale\\E200\\rTBCons.out";
  int nMin1 = 882;
  int nMax1 = 1655;
  int nMin2 = 2897;
  int nMax2 = 3116;
  float zAtRepoLine = -433.86;

  // Allocate memory for possible error messages

  messageBuffer = new char [151];
  if (!messageBuffer){
    cerr << "Memory allocation error for messageBuffer\n";
    cerr << "Press any key to end: ";
    getch();
    return (1);
  }

  //
  // Setup two node lists defining the top and bottom rows of nodes
  // to be connected ~~north-south~~
  //
  NdData* firstTopNode = 0;
  NdData* firstBaseNode = 0;
  if (
     (GetNodes(topFile,&firstTopNode) == ERROREND) ||
     (GetNodes(baseFile,&firstBaseNode) == ERROREND)
  ){
    DeleteNodes(&firstTopNode);
    DeleteNodes(&firstBaseNode);
    DumpAndQuit();
  }

  //
  // Read node connectivity data from file and determine the connectivity
  // and connection numbers between top-zone and base-zone nodes

  ifstream Fn(conneFile);
  if (!Fn){
    sprintf(messageBuffer,"Unable to open file %s",conneFile);
    DeleteNodes(&firstTopNode);
    DeleteNodes(&firstBaseNode);
    DumpAndQuit();
  }
  int conNum = 0;
  NdData* pTopNode = 0;
  NdData* pBaseNode = 0;
  ConData* firstCon = 0;
  ConData* lastCon = 0;
  int topNode,baseNode;
  int node1,node2,inc,conDir,dir2;
```

RTBCons p2 of 3

```
  float d1,d2,area,angle;
  char buf[100];
  size_t nBuf = sizeof(buf);

  while (!Fn.eof()){                                    // Begin #1 Loop
    Fn.getline(buf,nBuf);
    istrstream inpLine(buf,nBuf);
    inpLine >> node1 >> node2 >> inc >> conDir >> dir2
            >> d1 >> d2 >> area >> angle;
    if (inpLine.good()){                                // Begin #2 Loop
      conNum++;
      if (
        ((node1 >= nMin1) && (node1 <= nMax1)) ||
        ((node1 >= nMin2) && (node1 <= nMax2))
      ){                                                // Begin #3 Loop
        pTopNode = FindNode(firstTopNode,node1);
        if (pTopNode) pBaseNode = FindNode(firstBaseNode,node2);
        if ((pTopNode) && (pBaseNode)){                 // Begin #4 Loop
          topNode = pTopNode->node;
          baseNode = pBaseNode->node;
          ConData* current = new ConData(conNum,topNode,baseNode);
          if (!current){
            sprintf(messageBuffer,"Memory allocation error");
            DeleteNodes(&firstTopNode);
            DeleteNodes(&firstBaseNode);
            DeleteConList(&firatCon);
            DumpAndQuit();
          }
          current->x = RepoLine(pTopNode->x,pBaseNode->x,pTopNode->z,
            pBaseNode->z,zAtRepoLine);
          current->cTheta = ConneCosine(pTopNode->x,pBaseNode->x,pTopNode->z,
            pBaseNode->z);
          current->area =  area;
          StoreCon(current,&firstCon,&lastCon);
        }                                               // End #4 Loop
        pTopNode = 0;
        pBaseNode = 0;
      }                                                 // End #3 Loop
    }                                                   // End #2 Loop
  }                                                     // End #1 Loop

  if (PrintConList(firstCon,outFile) == ERROREND){
    DeleteNodes(&firstTopNode);
    DeleteNodes(&firstBaseNode);
    DeleteConList(&firstCon);
    DumpAndQuit();
  }

// Done

  DeleteNodes(&firstTopNode);
  DeleteNodes(&firstBaseNode);
  DeleteConList(&firstCon);
  delete[] messageBuffer;
  cout << "Done .. press any key to end:";
  getch();
  return 0;
}
int GetNodes(char* fileName, NdData** firstNode)
{
  int nodeNum;
  float x,z;
  char buf[100];
  size_t nBuf = sizeof(buf);

  ifstream Fn(fileName);
  if (!Fn){
    sprintf(messageBuffer,"Unable to open file %s",fileName);
    return (ERROREND);
  }

  NdData* preNode = *firstNode;
  while (!Fn.eof()){
    Fn.getline(buf,nBuf);
    istrstream inpLine(buf,nBuf);
    inpLine >> nodeNum >> x >> z;
    if (inpLine.good()){
      NdData* current = new NdData(nodeNum,x,z);
      if (!current){
        sprintf(messageBuffer,"Memory allocation error");
        return (ERROREND);
      }
      if (!(*firstNode)){
        *firstNode = current;
        preNode = current;
      }
      else{
        preNode->next = current;
        preNode = current;
      }
    }
  }
  if (!(*firstNode)){
    sprintf(messageBuffer,"No valid data found in file %s",fileName);
    return (ERROREND);
  }
  return (NORMALEND);
}

NdData* FindNode(NdData* firstNode, int nodeNum)
{
  while (firstNode){
    if ((firstNode->node) == nodeNum)
      return (firstNode);
    firstNode = firstNode->next;
  }
  return (0);
}

float RepoLine(float xTop,float xBase,float zTop,float zBase,float z0)
{
  return (
    xTop + (z0 - zTop)/(zBase - zTop)*(xBase - xTop)
  );
}

double ConneCosine(float xTop,float xBase,float zTop,float zBase)
{
  double theta = atan2(xBase-xTop, zBase-zTop);
  double ct = cos(theta);
  return (sqrt(ct*ct));
}

void StoreCon(ConData *current, ConData **first, ConData **last)
{
  if (!*last){          /* current is the very first item in the list */
    *last = current;
    *first = current;
    return;
  }

  ConData *old = 0;
  ConData *p = *first;
  while (p){
    if ((current->x) > (p->x)){
      old = p;
      p = p->next;
    }
    else{
      if (old) {      /* insert current at this point in the list */
        old->next = current;
        current->next = p;
```

p.3 of 3

```
      return;
    }
    current->next = p;    /* current becomes new first element */
    *first = current;
    return;
  }
 }

 (*last)->next = current;  /* current becomes new last element */
 *last = current;
}

int PrintConList(ConData* first, char* outFile)
{

 if (!first){
   sprintf(messageBuffer,"Empty list passed to PrintConList");
   return (ERRORREND);
 }

// Open output file
 ofstream Fout(outFile);
 if (!Fout){
   sprintf(messageBuffer,"Unable to open file %s",outFile);
   return (ERRORREND);
 }
 Fout << setw(10) << "ConNum"
      << setw(10) << "Node1"
      << setw(10) << "Node2"
      << setw(12) << "xCoord"
      << setw(22) << "cosTheta"
      << setw(12) << "area"
      << endl;

 while (first){
   Fout << setw(10) << (first->index)
        << setw(10) << (first->node1)
        << setw(10) << (first->node2)
        << setioflags(ios::fixed) << setprecision(2)
        << setw(12) << (first->x)
        << setprecision(16)
        << setw(22) << (first->cTheta)
        << setprecision(3)
        << setw(12) << (first->area)
        << endl;
   first = first->next;
 }
 return (NORMALEND);
}

void DeleteNodes(NdData** firstNode)
{
 NdData *p = *firstNode;
 NdData *current = p;

 while (p){
   current = p;
   p = p->next;
   delete current;
 }
}

void DeleteConList(ConData** first)
{
 ConData *p = *first;
 ConData *current = p;

 while (p){
   current = p;
   p = p->next;
   delete current;
 }
}

void DumpAndQuit()
{
 cerr << "\n" << messageBuffer << "\n";
 delete[] messageBuffer;
 cerr << "Press any key to end: ";
 getch();
 exit(1);
}
```

p.1 of 4

```
/*****************************************************************
RTBFlux

This code calculates the vertical component of liquid flux across
the repository axis (in mm/yr) from METRA output of fracture and matrix
fluxes (given in m/s). Nodal connectivity data for connections that cross
the repository axis are read from the output of code RTBCONS.
Fracture porosity data, which are required to combine fracture and matrix
fluxes, are read from the METRA DCM input block for the appropriate model.

Some of the nodes below the repository axis that are used in the calculation
receive fluxes through both vertical, diagonal, and lateral connections. The
fluxes are combined to obtain the total flux into such nodes.

Modify the values of the following variables as necessary:

domFile                 DCMPAR input block for metra
fluxFile                Gives matrix and fracture fluxes from METRA
conFile                 Defines node connectivity for a set of nodes
outFile                 Output file
minConNum               Minimum connection index in conFile
maxConNum               Maximum connection index in conFile

Author:                         G. I. Ofoegbu
Date:                           February 2000
System:                         C++ compiler
                                (Code developed using Borland C++ Builder 4)
*****************************************************************/
{
#include <stdio>
#include <iostream>
#include <iomanip>
#include <fstream>
#include <strstrea>
#include <conio>
#include <stdlib>
#include <math>
#include <string>

#pragma hdrstop
#include <condefs>

using namespace std;

char* messageBuffer;
void DumpAndQuit();

int ERROREND = 1;
int NORMALEND = 0;

//char* dcmFile = "d:\\HydrTherm\\RepoScale\\E200\\M22Cases\\m22.dcm";
//char* dcmFile = "d:\\HydrTherm\\RepoScale\\E200\\M33Cases\\m34.dcm";
char* dcmFile = "d:\\HydrTherm\\RepoScale\\E200\\M40Cases\\m46.dcm";

struct ConData{
  int index;
  int node1;
  int node2;
  float x;
  float cTheta;
  float area;
  float mLiqVel;
  float mGasVel;
  float fLiqVel;
  float fGasVel;
  float liqVel;
  ConData* next;
  ConData(int conNum, int n1, int n2, float xc, float c2t, float a){
    index = conNum;
    node1 = n1;
    node2 = n2;
    x = xc;
    cTheta = c2t;
    area = a;
    next = 0;
  }
};
int GetConData(char* fileName, ConData** first, ConData** last);
void StoreConData(ConData* current, ConData** first, ConData** last);
void SaveFluxData(ConData* first, int conNum, float mLiqVel, float mGasVel,
  float fLiqVel, float fGasVel);
int TotalLiqVelocity(ConData* first);
int GetFracPor(int node, float& fPor);
int ExtractDiagCons(ConData** firstCon, ConData** dcFirst, ConData** dcLast);
void CombineFluxes(ConData* firstCon, ConData* dcFirst);
void AddDiagFlux(int theNode, ConData* dcFirst, float& fluxSum);
int PrintConData(ConData* first, char* outFile);
void DeleteConData(ConData** first);

//---------------------------------------------------------------------------
#pragma argsused
int main(int argc, char* argv[])
{
//  char* fluxFile = "d:\\HydrTherm\\RepoScale\\E200\\M22Cases\\m22Vel.out";
//  char* fluxFile = "d:\\HydrTherm\\RepoScale\\E200\\M33Cases\\m34Vel.out";
  char* fluxFile = "d:\\HydrTherm\\RepoScale\\E200\\M40Cases\\m46Vel.out';

//  char* outFile = "d:\\HydrTherm\\RepoScale\\E200\\M22Cases\\m22rTBFlx.out";
//  char* outFile = "d:\\HydrTherm\\RepoScale\\E200\\M33Cases\\m34rTBFlx.out";
  char* outFile = "d:\\HydrTherm\\RepoScale\\E200\\M40Cases\\m46rTBFlx.out';

  char* conFile = "d:\\HydrTherm\\RepoScale\\E200\\rTBCons.out';
  int minConNum = 2200;
  int maxConNum = 8200;

// Allocate memory for possible error messages

  messageBuffer = new char [151];
  if (!messageBuffer){
    cerr << "Memory allocation error for messageBuffer\n';
    cerr << "Press any key to end: ";
    getch();
    return (1);
  }

//
// Setup the connections list defining the top and bottom rows of nodes
// for which vertical flux data is required
//
  ConData* firstCon = 0;
  ConData* lastCon = 0;
  cout << "Setting up connections database ... \n"
       << "from file "
       << conFile << " ... \n";
  if (GetConData(conFile,&firstCon,&lastCon) == ERROREND){
    DeleteConData(&firstCon);
    DumpAndQuit();
  }
//
// Read flux data from file and extract the flux data for the
// connections within the connections list that was setup previously
//

  ifstream Fn(fluxFile);
  if (!Fn){
    sprintf(messageBuffer,"Unable to open file %s",fluxFile);
    DeleteConData(&firstCon);
    DumpAndQuit();
  }

  float mLiqVel,mGasVel,fLiqVel,fGasVel;
  char buf[100];
```

p.2 of 4

```
  size_t nBuf = sizeof(buf);
  int conNum = 0;

  cout << "\nAdding flux to database ...\n"
       << "Flux results from file "
       << fluxFile << " ...\n";
  while (!Fn.eof()){
    Fn.getline(buf,nBuf);
    istrstream inpLine(buf,nBuf);
    inpLine >> conNum >> mLiqVel >> mGasVel >> fLiqVel >> fGasVel;
    if (inpLine.good())
      if ((conNum >= minConNum) && (conNum <= maxConNum))
        SaveFluxData(firstCon,conNum,mLiqVel,mGasVel,fLiqVel,fGasVel);
  }

  cout << "\nAdding total liquid velocities to database ...\n';
  cout << "Fracture porosity file: " << dcmFile << endl;
  if (TotalLiqVelocity(firstCon) == ERROREND){
      DeleteConData(&firstCon);
      DumpAndQuit();
  }
//
// Split the ConData list into two lists: a verticalConnections list and
// a diagonalConnections list
//
  ConData* dcFirst = 0;
  ConData* dcLast = 0;
  int nDc = ExtractDiagCons(&firstCon,&dcFirst,&dcLast);
  cout << nDc << " connections transferred to diagonal list\n';

  cout << "\nAdding diagonal fluxes to vertical fluxes ... "
       << "for connections to same node2\n";
  CombineFluxes(firstCon,dcFirst);

  cout << "\nPrinting results to file " << outFile << " ...\n";
  if (PrintConData(firstCon,outFile) == ERROREND){
      DeleteConData(&firstCon);
      DeleteConData(&dcFirst);
      DumpAndQuit();
  }

// Done

  DeleteConData(&firstCon);
  DeleteConData(&dcFirst);
  delete[] messageBuffer;
  cout << "\nDone .. press any key to end:";
  getch();
  return 0;
}

int GetConData(char* fileName, ConData** first, ConData** last)
{
  int conIndex,node1,node2;
  float x;
  float cTheta;
  float area;
  char buf[100];
  size_t nBuf = sizeof(buf);

  ifstream Fn(fileName);
  if (!Fn){
    sprintf(messageBuffer,"Unable to open file %s",fileName);
    return (ERROREND);
  }

  while (!Fn.eof()){
    Fn.getline(buf,nBuf);
    istrstream inpLine(buf,nBuf);
    inpLine >> conIndex >> node1 >> node2 >> x >> cTheta >> area;
    if (inpLine.good()){
      ConData* current = new ConData(conIndex,node1,node2,x,cTheta,area);
      if (!current){
        sprintf(messageBuffer,"Memory allocation failure");
        return (ERROREND);
      }
      StoreConData(current,first,last);
    }
  }
  return (NORMALEND);
}

void StoreConData(ConData *current, ConData **first, ConData **last)
{
  if (!*last){          /* current is the very first item in the list */
    *last = current;
    *first = current;
    return;
  }

  ConData *old = 0;
  ConData *p = *first;
  while (p){
    if ((current->x) > (p->x)){
      old = p;
      p = p->next;
    }
    else{
      if (old) {      /* insert current at this point in the list */
        old->next = current;
        current->next = p;
        return;
      }
      current->next = p;    /* current becomes new first element */
      *first = current;
      return;
    }
  }

  (*last)->next = current;  /* current becomes new last element */
  *last = current;
}

void SaveFluxData(ConData* first, int conNum, float mLiqVel, float mGasVel,
  float fLiqVel, float fGasVel)
{
  while (first){
    if ((first->index) == conNum){
      first->mLiqVel = mLiqVel;
      first->mGasVel = mGasVel;
      first->fLiqVel = fLiqVel;
      first->fGasVel = fGasVel;
      return;
    }
    first = first->next;
  }
}

int TotalLiqVelocity(ConData* first)
{
  float secPerYr = 365.25*24.*60.*60.;
  float mmPerM = 1000.;
  float fPor1;
  float fPor2;

  float count = 0.0;
  int numDone = count;
  while (first){
    if (
      (GetFracPor(first->node1,fPor1) == ERROREND) ||
      (GetFracPor(first->node2,fPor2) == ERROREND)
    ) return (ERROREND);
```

p.3 of 4

79

```
    float fracPorAvg = 0.5*(fPor1 + fPor2);
    float mVel = first->mLiqVel;
    float fVel = first->fLiqVel;
    fVel *= fracPorAvg;
    first->liqVel = (mVel + fVel)*secPerYr*mmPerM;
    count += 1.0;
    numDone = count;
    if (fmod(count,10.) < 0.09)
      cout << "Working ... processed " << numDone << " connections\n";
    first = first->next;
  }
  cout << "Done ... processed " << numDone << " connections\n";
  return (NORMALEND);
}
int GetFracPor(int node, float& fPor)
{
  ifstream Fin(dcmFile);
  if (!Fin){
    sprintf(messageBuffer,"Unable to open file %s",dcmFile);
    return (ERROREND);
  }
  int m1,m2,inc;
  float sigmaf,gasAmod,xlm,ylm,zlm,liqAmod;
  char buf[151];
  size_t nBuf = sizeof(buf);

  while (!Fin.eof()){
    Fin.getline(buf,nBuf);
    istrstream inpLine(buf,nBuf);
    inpLine >> m1 >> m2 >> inc >>
      sigmaf >> gasAmod >> xlm >> ylm >> zlm >> liqAmod;
    if ((inpLine.good()) && (m1 == node)){
      fPor = sigmaf;
      Fin.close();
      return (NORMALEND);
    }
  }
  Fin.close();
  sprintf(messageBuffer,"Fracture porosity data not found for node %d",node);
  return (ERROREND);
}

int ExtractDiagCons(ConData** firstCon, ConData** dcFirst, ConData** dcLast)
{
//
// Go through the connections list until the first vertical connection
// is encountered. All connections ahead of the first vertical
// connection are re-assigned to the diagonalConnections list. Thereafter
// re-assign firstCon to the address of the first vertical connection
//
  int nDc = 0;
  ConData* theCon = *firstCon;
  ConData* curDc = 0;
  float minVCos = 0.5;
  while (theCon){
    if ((theCon->cTheta) > minVCos)
      break;
    curDc = theCon;
    theCon = theCon->next;
    curDc->next = 0;
    nDc++;
    StoreConData(curDc,dcFirst,dcLast);
  }
  *firstCon = theCon;
//
// Transfer all diagonal connections to the diagonalConnections list
//
  ConData* preVc = theCon;
  while (preVc->next){
    theCon = preVc->next;
    if ((theCon->cTheta) > minVCos)
      preVc = theCon;
    else{
      curDc = theCon;
      theCon = theCon->next;
      preVc->next = theCon;
      curDc->next = 0;
      nDc++;
      StoreConData(curDc,dcFirst,dcLast);
    }
  }
  return (nDc);
}

void CombineFluxes(ConData* vcFirst, ConData* dcFirst)
{
  ConData* theVc = vcFirst;
  while (theVc){
    float area = theVc->area;
    float cTheta = theVc->cTheta;
    float fluxSum = area*(theVc->liqVel);
    AddDiagFlux(theVc->node2,dcFirst,fluxSum);
    theVc->liqVel = fluxSum/(area/cTheta);
    theVc = theVc->next;
  }
}

void AddDiagFlux(int theNode, ConData* dcFirst, float& fluxSum)
{
  ConData* theDc = dcFirst;
  while (theDc){
    if ((theDc->node2) == theNode)
      fluxSum += (theDc->area)*(theDc->liqVel);
    theDc = theDc->next;
  }
}

int PrintConData(ConData* first, char* outFile)
{

  if (!first){
    sprintf(messageBuffer,"Empty list passed to PrintConData");
    return (ERROREND);
  }

// Open output file
  ofstream Fout(outFile);
  if (!Fout){
    sprintf(messageBuffer,"Unable to open file %s",outFile);
    return (ERROREND);
  }
  Fout << setw(8) << "ConNum"
       << setw(8) << "Node1"
       << setw(8) << "Node2"
       << setw(10) << "xCoord"
       << setw(15) << "LiqVel mm/yr"
       << setw(15) << "effectiveArea"
       << endl;

  while (first){
    Fout << setw(8) << (first->index)
         << setw(8) << (first->node1)
         << setw(8) << (first->node2)
         << setioflags(ios::fixed) << setprecision(2)
         << setw(10) << (first->x)
         << setprecision(3)
         << setw(15) << (first->liqVel)
         << setw(15) << ((first->area)/(first->cTheta))
         << endl;
    first = first->next;
  }
  return (NORMALEND);
```

p.4 of 4

```
}
void DeleteConData(ConData** first)
{
  ConData *p = *first;
  ConData *current = p;

  while (p){
    current = p;
    p = p->next;
    delete current;
  }
}

void DumpAndQuit()
{
  cerr << "\n" << messageBuffer << "\n";
  delete[] messageBuffer;
  cerr << "Press any key to end: ";
  getch();
  exit(1);
}
```

March 1 2000

Pages 80-81 GW entry by 3/1/2000

Thermal-Load Cases

The list of nodes used for thermal-load application was changed in order to locate some of the source nodes in areas of elevated flux. The new set of heat-source nodes are shown as circles in the figure on p.72. They are also shown below to clearly illustrate their relationship with the mesh and with the simulated PTn fault zone.

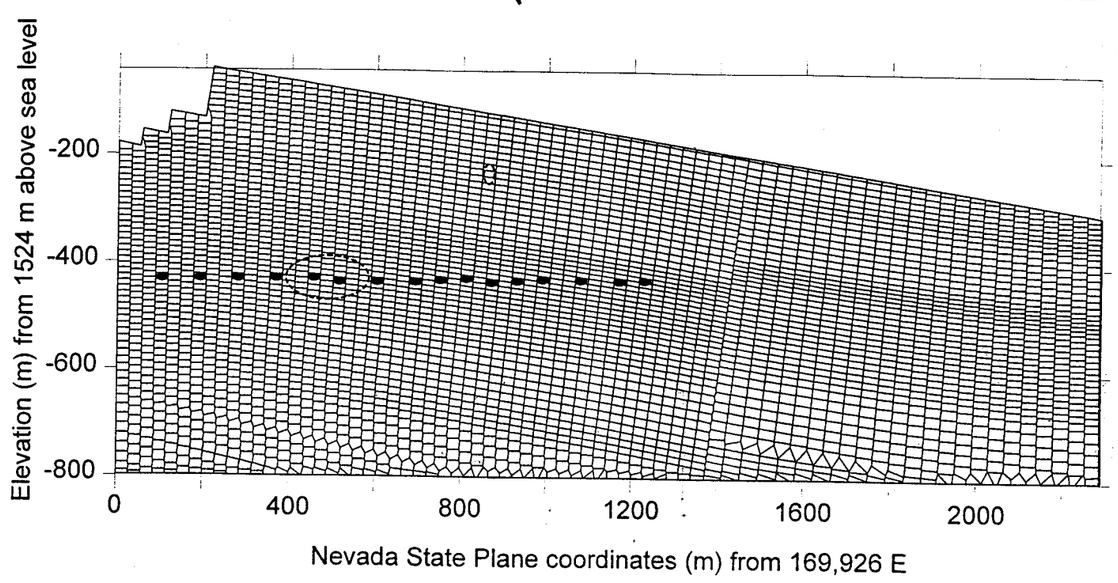


Figure * Finite difference discretization of the model domain. Cells used for heat-source application are marked with dark dots. Two ellipses with dashed boundaries represent generic geomechanically altered zones: a fault zone intersecting the PTn represented by the sub-vertical ellipse centered at about (860,-240), and a thermal-mechanical altered zone represented by the horizontal ellipse centered at about (500,-434).

The heat-source strength applied to each of the nodes was calculated using the procedure described on p.55. Two thermal-load cases were analyzed:

h24	corresponding with base case m22
h34	corresponds with case m34 described on p.68.

The input files are:

for case h24

h24.dat
m20.con
m22.phk
h23.int (from m22 steady state)
m22.dam
m20.bc
h34.src (heat source definition)

for case h34

m20.con, m20.bc, h34.src
h34.dat, h34.int (from m34 steady state)
m34.phk, m34.dam

GW 3/1/2000

Pages 80-81 GW entries by 3/1/2000

April 20 2000

Pages 82-88 GW 4/20/2000 entries

A few Results

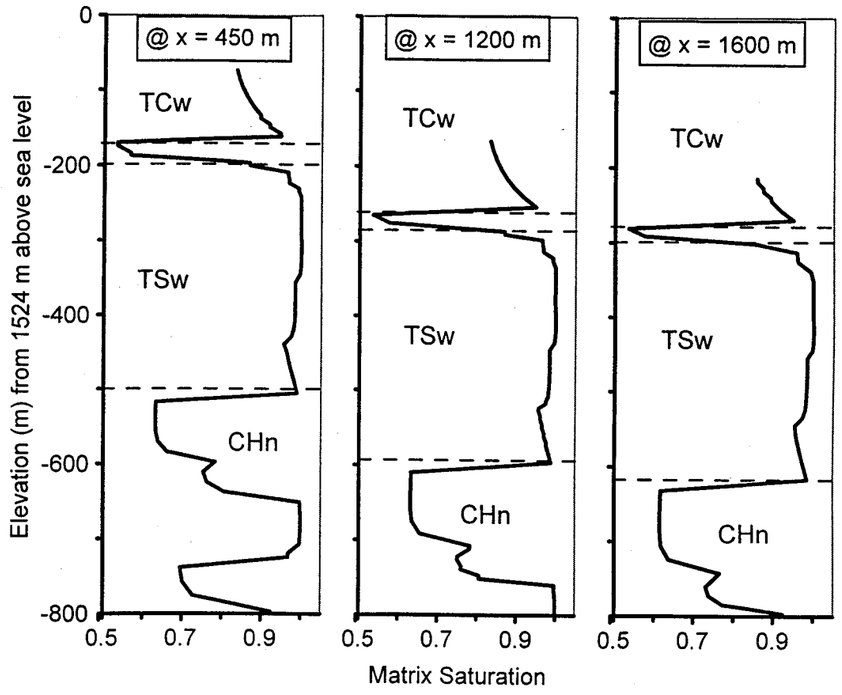
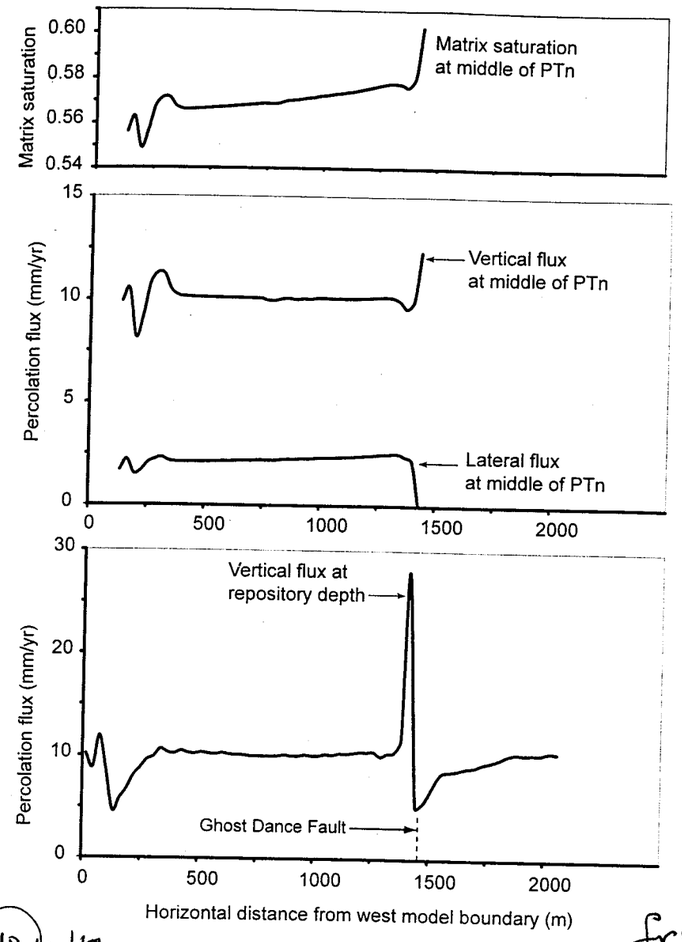


Figure ~~4~~ Vertical profiles of matrix saturation at horizontal distances of 450, 1200, and 1600 m from the west model boundary. Broken lines represent the interfaces between welded and non-welded tuff units. The profiles represent the initial steady-state moisture distribution (before waste emplacement) calculated using the base-case model. ← model m22

The saturation values in the TSw should be in the 0.9 – 0.97 range but are close to 1.0 here in tsw34 and tsw35. It has been suggested that parameter bklken on p.36 should be increased to 3.0 to improve the

saturation prediction. This suggestion will be investigated.



GW 4/20/00

from m22

Figure ~~5~~ Profiles of matrix saturation and vertical and lateral fluxes at mid-depth of the PTn unit, and vertical percolation flux at the repository depth, illustrating the effects of PTn-TSw contrast, stratigraphic dip, and major faults such as the GDF on deep percolation. The PTn profiles include only the part of ~~the PTn~~ on the west side of the GDF ~~(Fig 1)~~. GW 4/20/00

GW

From m34, m36, and m35 (p. 67)

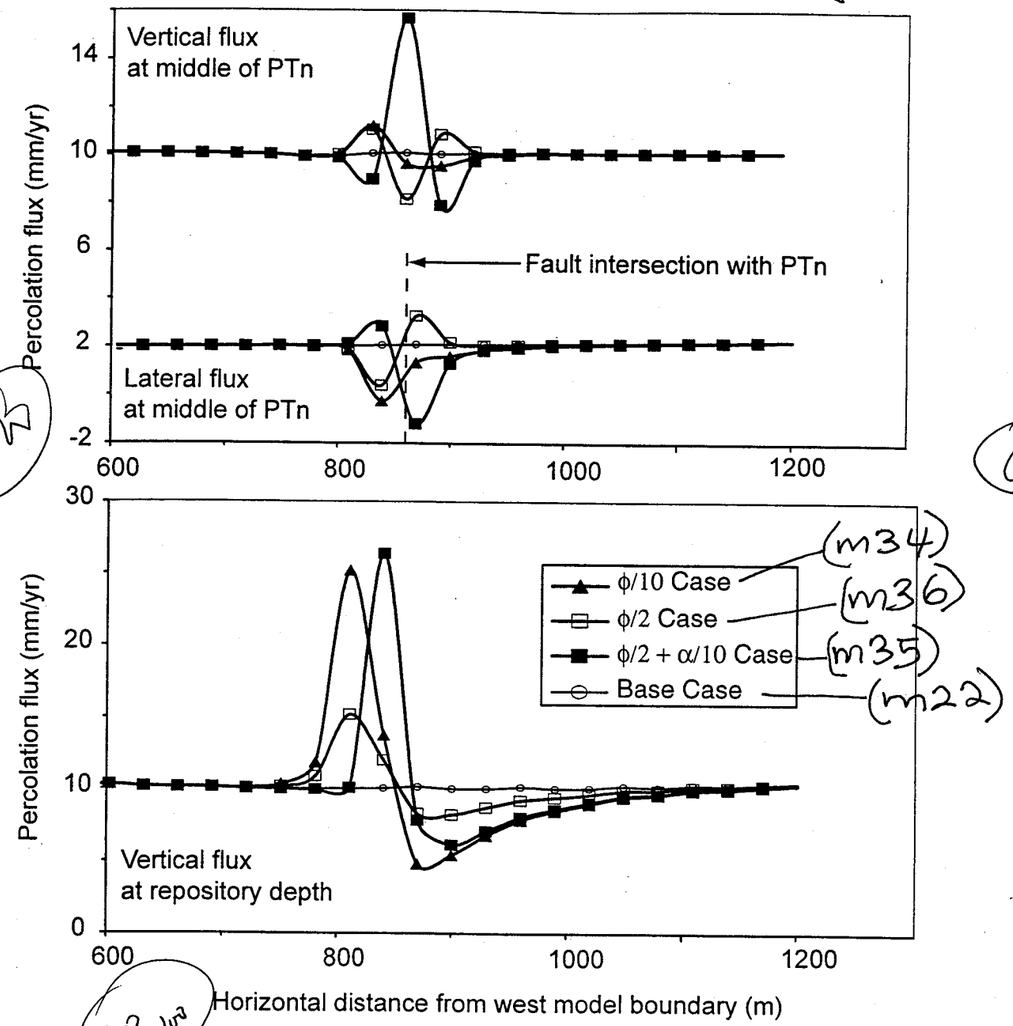


Figure X Profiles of percolation flux at the mid-depth of the PTn unit and at the repository depth, illustrating the effects of a generic fault zone that intersects, but does not necessarily offset, the PTn. Lateral flux is positive in the direction of increasing horizontal distance.

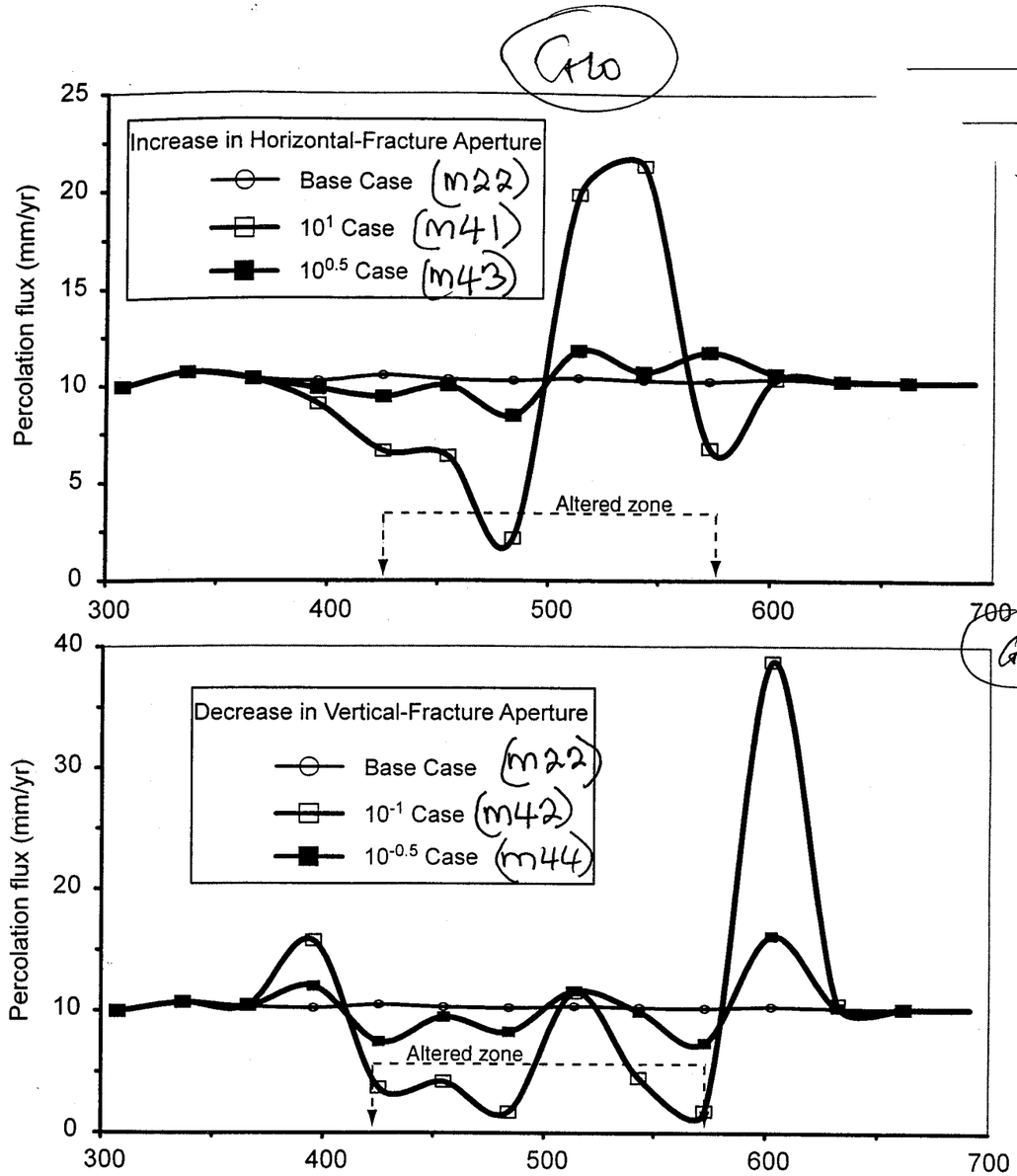


Figure X Profiles of percolation flux at the repository depth, illustrating the effects of a generic thermal-mechanical altered zone. The results represent the effects of horizontal-fracture dilation with no change in vertical fractures and vertical-fracture closure with no change in horizontal fractures. Both cases are compared with the base case for which there is no change in fracture aperture.

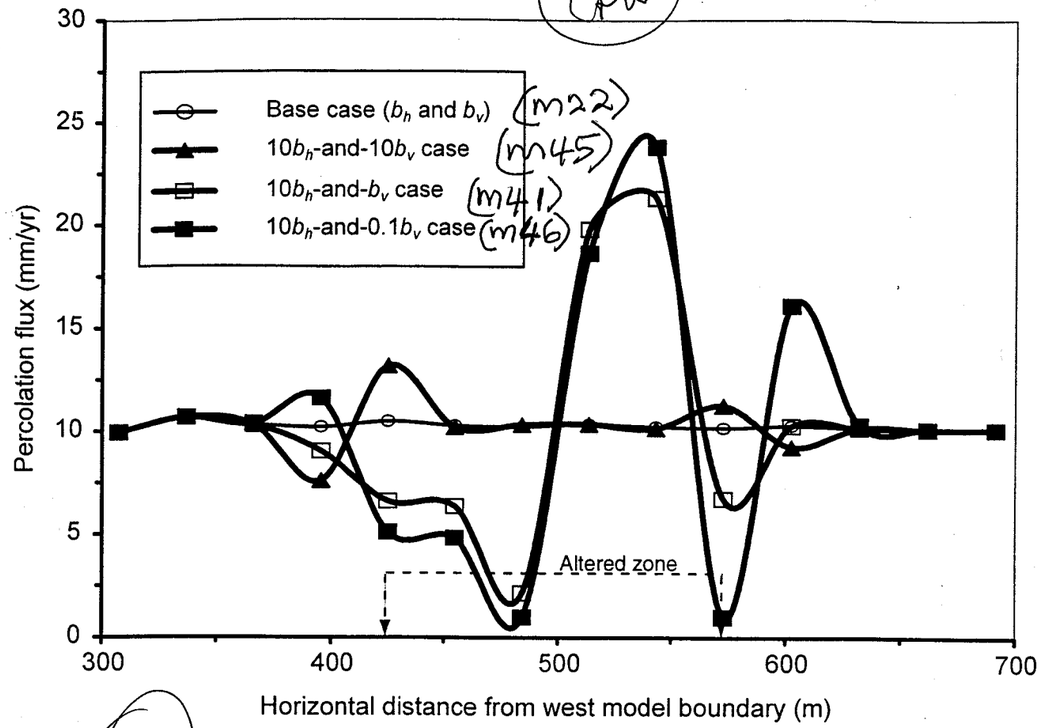


Figure 36: Profiles of percolation flux at the repository depth, illustrating the effects of a generic thermal-mechanical altered zone. The results represent the effects of combining horizontal-fracture dilation with different magnitudes of net vertical-fracture dilation and closure. Parameters b_h and b_v represent the base-case horizontal and vertical fracture apertures, respectively

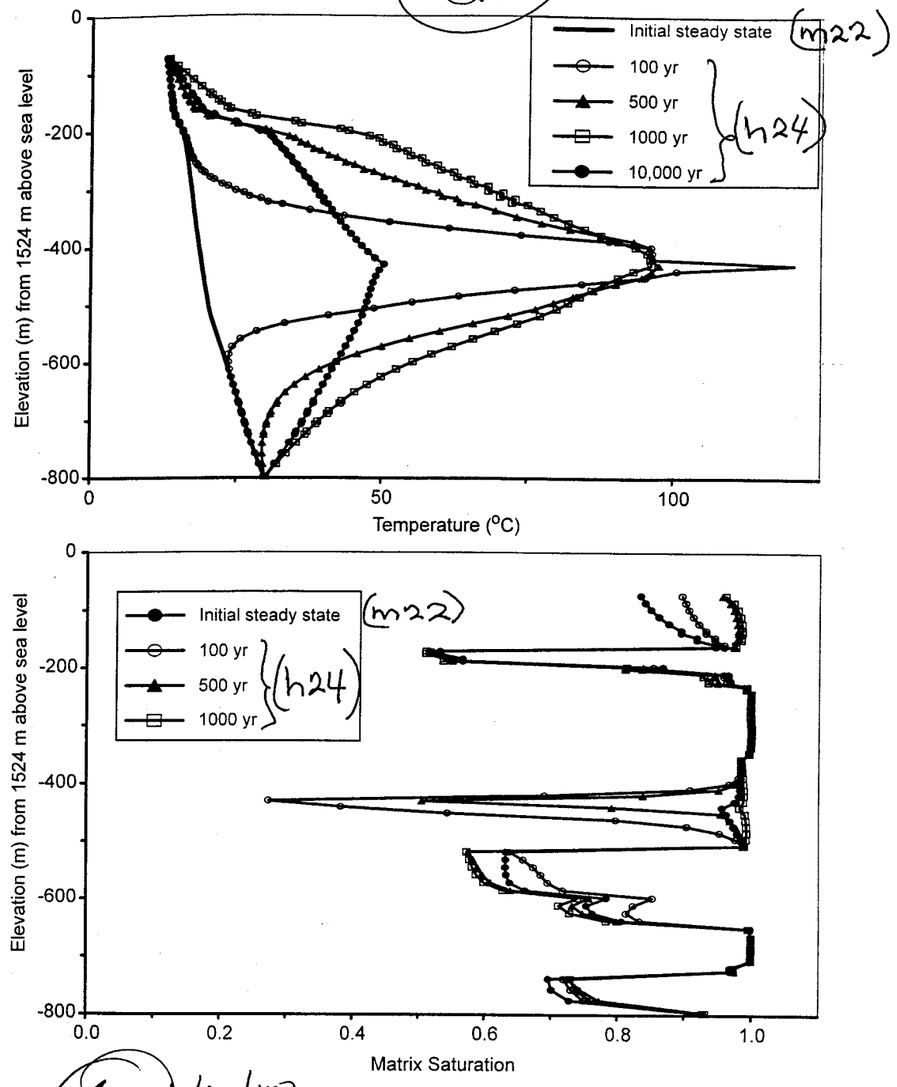


Figure 37: Vertical profiles of temperature and matrix saturation from the base-case model with thermal loading. Profiles represent the behavior at a horizontal distance of 450 m from the west model boundary.

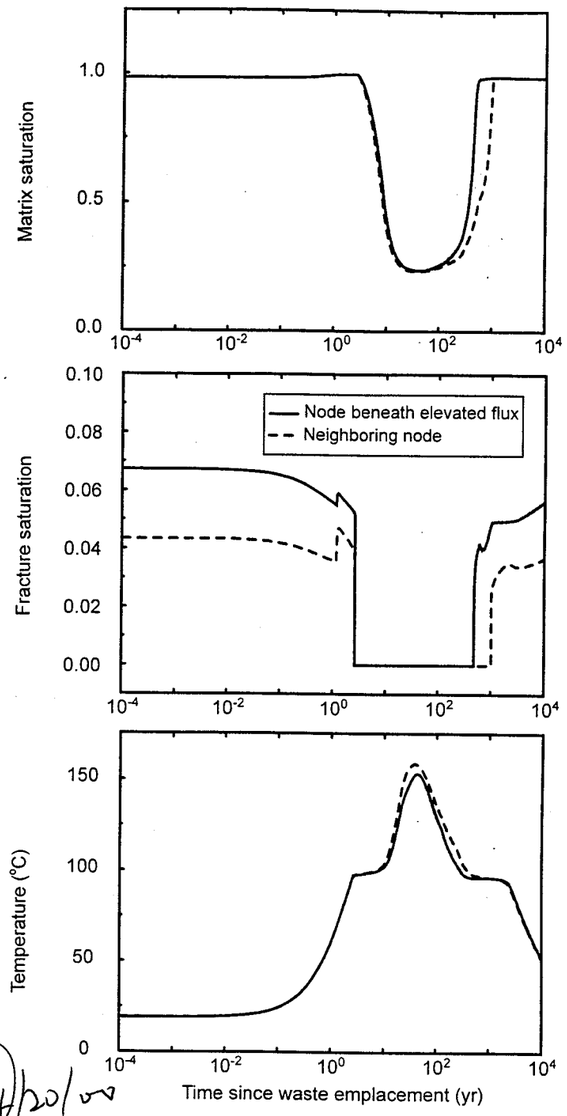


Figure 4.1: Histories of temperature and saturation illustrating the effects of focussed water flux from a generic minor fault in the PTn. Taken from a thermal-load case that includes the PTn fault zone as illustrated in Fig. A. Solid and broken-line curves are for the 7th and 6th heat-source nodes (counting from the east), respectively.

Both curves are from model h34

Pages 87-88 entries 4/20/2000

I have reviewed this scientific notebook and find it in compliance with SPP-501. There is sufficient information regarding procedures used for conducting tests, acquiring and analyzing data so that another qualified individual could repeat the activity.

9-30-04