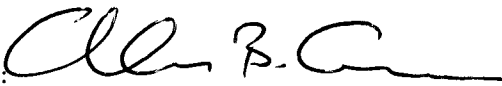


SOFTWARE RELEASE NOTICE

1. SRN Number: GLGP-SRN-215		
2. Project Title: PVHA-YM		Project No. 20.01402.462
3. SRN Title: PVHA-YM Version 1.0		
4. Originator/Requestor: Chuck Connor		Date: 2/15/00
5. Summary of Actions <ul style="list-style-type: none"> <input checked="" type="checkbox"/> Release of new software <input type="checkbox"/> Release of modified software: <ul style="list-style-type: none"> <input type="checkbox"/> Enhancements made <input checked="" type="checkbox"/> Corrections made <input type="checkbox"/> Change of access software <input checked="" type="checkbox"/> Software Retirement <i>ALX MCK 01/31/03</i> 		
6. Persons Authorized Access		
Name	Read Only/Read-Write	Addition/Change/Delete
David Ferrill	RO	Addition
John Stamatakos	RO	Addition
Larry MCKague	RO	Addition
John Trapp (NRC)	RO	Addition
Chuck Connor	RW	Addition
Britt Hill	RO	Addition
Ron Martin	RO	Addition
7. Element Manager Approval: Larry McKague <i>ALX MCKague</i>		Date: 2/17/00
8. Remarks:		

2/13/00

SOFTWARE SUMMARY FORM

01. Summary Date: 2/15/00		02. Summary prepared by (Name and phone) Chuck Connor 210.522.6649		03. Summary Action: NEW	
04. Software Date: 2/15/00		05. Short Title: PVHA-YM Version 1.0			
06. Software Title: PVHA-YM Version 1.0					07. Internal Software ID: NONE
08. Software Type: <input type="checkbox"/> Automated Data System <input checked="" type="checkbox"/> Computer Program <input type="checkbox"/> Subroutine/Module		09. Processing Mode: <input checked="" type="checkbox"/> Interactive <input type="checkbox"/> Batch <input type="checkbox"/> Combination		10. Application Area a. General: <input checked="" type="checkbox"/> Scientific/Engineering <input type="checkbox"/> Auxiliary Analyses <input type="checkbox"/> Total System PA <input type="checkbox"/> Subsystem PA <input type="checkbox"/> Other b. Specific:	
11. Submitting Organization and Address: CNWRA/SwRI 6220 Culebra Road San Antonio, TX 78228			12. Technical Contact(s) and Phone: Chuck Connor 210.522.6649		
13. Software Application: PVHA-YM is a probabilistic volcanic hazard assessment application used to calculate probability of volcanic eruptions through the proposed Yucca Mountain repository.					
14. Computer Platform Windows NT, SUN, SGI, Linux		15. Computer Operating System: N/A		16. Programming Language(s): JAVA	
17. Number of Source Program Statements: 6000		18. Computer Memory Requirements: 128 Mb		19. Tape Drives: N/A	
20. Disk Units: hard drive		21. Graphics: JVM 1.1			
22. Other Operational Requirements: NONE					
23. Software Availability: <input checked="" type="checkbox"/> Available <input type="checkbox"/> Limited <input type="checkbox"/> In-House ONLY			24. Documentation Availability: <input checked="" type="checkbox"/> Available <input type="checkbox"/> Preliminary <input type="checkbox"/> In-House ONLY		
25. Software Developer:  Date: 02/16/00					

DESIGN VERIFICATION REPORT FOR CNWRA SOFTWARE: PVHA_YM V. 1.0

Date 2/15/2000

Probabilistic Volcanic Hazard Assessment Methods for a Proposed High-Level Radioactive Waste Repository at Yucca Mountain (Scientific and Engineering Software) Version 1.0

NOTE: This version of the PVHA_YM Version 1.0 Software is the initial release.

1. This Design Verification Report is prepared by (names of CNWRA software custodian and software developer): Bruce Mabrito / Chuck Connor

Full Title of CNWRA scientific and engineering software: Probabilistic Volcanic Hazard Assessment Methods for a Proposed High-Level Radioactive Waste Repository at Yucca Mountain Version 1.0

Demonstration work station: On the B. Mabrito NT computer.

Operating System: NT. This software can operate, according to the developer, on UNIX, SUN, SGI, LINUX and any others available.

2. Software Requirements Description and any changes thereto follow QAP-002 requirements?
YES NO N/A

Notes: Dated February 20, 1997.

3. Software Development Plan (SDP) and any changes have been approved by the Element Manager?
YES NO N/A

Notes: The SRD for PVHA_YM predates the requirement for an SDP.

4. Design and Development
Module-level testing is documented in either scientific notebooks or in Software Change Reports?
YES NO N/A

Notes: Documented in Scientific Notebook 115E

5. Is the CNWRA scientific and engineering software developed in accordance with the conventions described in the SDP?
YES NO N/A

Notes:

6. Is the CNWRA software documented internally?
 YES NO N/A

Does the primary program header contain the following information?

A. Program title, Developed for (Customer), Office/Division/Date/Customer Contact/Telephone number, Software Developer, Telephone number, titles of Associated Documentation/Designator, and the Disclaimer Notice?

YES NO N/A

B. Source code module header information provides Program Name, Client Name, Contract Reference, Revision number?

YES NO N/A

Notes: No individual modules in JAVA, each "class" has a header.

7. Software designed so that individual runs are uniquely identified by Date, Time, Name of software and version?

YES NO N/A

8. The physical labeling on the software or the referenced list has Program Name/Title, Module/Name/Title, Module Revision, File Type (i.e. ASCII, OBJ, EXE), Recording Date and Operating System of the Supporting Hardware?

YES NO N/A

9. Users' Manual

Is there a Users' Manual for the software?

YES NO N/A

If no, explain: The Users' Manual is on the CD of PVHA_YM in the html section.

Are there basic instructions for the use of the software?

YES NO N/A

Notes: On the CD, accessible via the web browser.

10. Acceptance Testing

Does the acceptance testing demonstrate whether or not requirements in the SRD have been fulfilled?

YES NO N/A

Notes: See Scientific Notebook 115E

Has acceptance testing been conducted for each intended computer platform and operating system?

YES NO N/A

Notes: Yes, Windows NT, SUN, SILICON Graphics, LINUX

Have installation tests been performed on the target platform?

YES NO N/A

Notes: Yes, the tests were conducted in the respective web browsers.

11. Configuration Control

Is the Software Summary Form completed and signed?

YES NO N/A

If no, explain:

12. Is a software technical description prepared, documenting the essential mathematical and numerical basis?

YES NO N/A

If no, explain: In Scientific Notebook 115E.

13. Is the source code available (or, is the executable code available in the case of commercial codes)?

YES NO N/A

The source code is located in the CNWRA QA Records Room.

14. Have all the script/make files and executable files been submitted to the Software Custodian?

YES NO N/A

Notes:

Allen B. C... 2/15/00
Date
CNWRA Software Code Developer

Steve Mah... 2/15/00
Date
CNWRA Software Custodian

Attachments/

Original to: Software Folder
cc: CNWRA Software Developer
Cognizant EM

6/131

PVHA_YM version 1.0 - Probabilistic Volcanic Hazard Assessment Methods for a Proposed High-Level Radioactive Waste Repository at Yucca Mountain, Nevada

Prepared for

Nuclear Regulatory Commission

Contract NRC-02-97-009

Prepared by

Charles B. Connor

Center For Nuclear Waste Regulatory Analyses

San Antonio, Texas

February 2000

Abstract

The assessment of long-term performance of the proposed high-level radioactive waste repository at Yucca Mountain, Nevada, requires the use of mathematical models to consider the probability of disruptive scenarios. Volcanism is one important disruptive scenario to consider in site evaluation. The purpose of the PVHA_YM software is to provide and document mathematical models developed to assist staff in the probabilistic volcanic hazards assessment (PVHA) of the Yucca Mountain site.

PVHA_YM is intended to be launched from a standard web browser. PVHA_YM includes java applets that can be used to estimate the probability of a volcanic event occurring

7/131

within a effective area about the repository using kernel density estimators to smooth the point pattern map distribution of previous volcanic events in the region. Two types of kernel density estimators are included: Gaussian and Epanechnikov. These density estimators are used to calculate the probability of volcanic events at the site, and to plot conditional probability maps of the location of volcanic events, given the occurrence of a volcanic event in the magmatic system. Articles are included that provide an overview of basaltic volcanism, summarize the application of kernel density estimators to the Yucca Mountain PVHA, and illustrate their application in analogous volcanic fields.

Table of Contents

Acknowledgments

Quality of Data, Analyses, and Code Development

Overview of PVHA YM

Definition of Volcanic Events

Basis for the PVHA

Explanation of Parameters Used in PVHA YM

Applets

ProbGraph1

ProbGraph2

ProbMap1

ProbMap2

Articles

Geologic controls on Basaltic Volcanism: Application to a Volcanic Hazards Assessment at Yucca Mountain

Three Nonhomogeneous Poisson models for the probability of basaltic volcanism: Application to Yucca Mountain

Recurrence rates of basaltic volcanism: An example from the Springerville volcanic field, Arizona

Introduction to Basaltic Volcanic Fields

Code Limitations

Source Code

Acknowledgments

This report was prepared to document work performed by the Center for Nuclear Waste Regulatory Analyses (CNWRA) for the U.S. Nuclear Regulatory Commission (NRC) under Contract No. NRC-02-97-009. The activities reported were performed on behalf of the U.S. NRC Office of Nuclear Material Safety and Safeguards, Division of Waste Management. This report is an independent product of the CNWRA and does not necessarily reflect the views or regulatory position of the NRC.

The author thanks the following individuals for their contributions to CNWRA research on PVHA at Yucca Mountain: Britt Hill, John Stamatakos, David Ferrill, Peter LaFemina,

Michael Conway, Goodluck Ofoegbu, Ron Martin, Budhi Sagar, John Trapp, and Chris Condit. Britt Hill compiled the data presented in the Data Table. The java class "graph" used in PVHA_YM is modified from code developed and freely distributed by Leigh Brookshaw, under the GNU licensing agreement. Technical reviews were performed by Peter La Femina, Randy Folck, and H. Lawrence McKague. Programmatic review was performed by Wesley Patrick. The efforts of all of these individuals are gratefully acknowledged.

Quality of Data, Analyses and Code Development

Data: No CNWRA-generated original data are contained in this report. Sources for other data should be consulted for determining the level of quality for those data.

Codes: The PVHA code Version 1.0, including the java applets ProbGraph1, ProbGraph2, ProbMap1, and ProbMap2, has been developed following the procedures described in the CNWRA Technical Operating Procedure (TOP) - 018, Development and Control of Scientific and Engineering Software, which implements the quality assurance (QA) guidance contained in CNWRA QA manual.

[return to top](#)

[return to Table of Contents](#)

Overview of PVHA_YM

The purpose of this document is to present codes and procedures used by the U.S. Nuclear Regulatory Commission (NRC) and the Center for Nuclear Waste Regulatory Analyses (CNWRA) staff to perform an independent probabilistic volcanic hazard assessment (PVHA) for the proposed high-level radioactive waste repository at Yucca Mountain, Nevada. NRC and CNWRA staff will use the PVHA_YM codes to review the PVHA performed by the U.S. Department of Energy (DOE) as part of their license application to NRC for construction of the proposed repository. Specifically, PVHA_YM codes have been developed as part of the Igneous Activity Key Technical Issue (IA KTI) with the goal of streamlining license application review. During the license application review process, PVHA_YM will be used to assist NRC review of expected post-closure performance and to evaluate DOE compliance with CFR63.113.

Volcanic hazards at the proposed site stem from the proximity of Yucca Mountain to small-volume basaltic volcanoes [[Figure 1, Map showing the distribution of volcanoes in the area](#)]. Because of the potential of adversely affecting the long-term performance of the repository, basaltic volcanism is considered a disruptive scenario in total-system performance assessments (TPAs) of the site. Results of the PVHA are incorporated in TPA estimates of radiological dose to a critical group, and ultimately into a risk assessment based on the expected radiological dose to this group. In this context, PVHA_YM has been developed as a tool to assist the NRC in their mission to assure public health and safety.

This document includes several articles that describe the geologic and statistical basis for the PVHA. The PVHA relies on the distribution and ages of these basaltic volcanoes as indicators of the expected rate of volcanic activity and the expected distribution of future volcanic events in the Yucca Mountain region.

PVHA_YM is intended to be viewed and used within a web browser. Printed documentation is only provided to create a document that is easily referenced. Four computer programs (java applets) are used to estimate the probability of volcanic eruptions at the site using Gaussian and Epanechnikov kernels, and to create maps of the probable distribution of future volcanic events using the same two kernels. Instructions in the use of these codes are provided within each applet.

PVHA_YM requires Java Virtual Machine language version 1.1 or higher enabled on the web browser. This software has been tested using Netscape versions 4.5, 4.6 and 4.7 on Windows NT, Linux, SGI, and Sun operating systems.

[return to top](#)

[return to Table of Contents](#)

Definition of Volcanic Events

It is crucial to define volcanic events explicitly in PVHA. In this PVHA for the Yucca Mountain site, the definition of a volcanic event is limited to extrusive eruptions that result in the formation of basaltic vents as indicated by cinder cones, spatter mounds, and/or lava flows. A complete compilation of basaltic vents in the Yucca Mountain region is provided in the Data Table. Estimates of the probability of dike injection, without volcanic activity, are not considered explicitly in this PVHA.

Even using this restricted definition, it is possible to define volcanic events in various ways. Individual vents can be considered to be volcanic events, or closely spaced and similarly aged vents can be grouped together. Furthermore, only cones younger than a specific age might be included in the analysis as volcanic events. Because vents may be grouped together into volcanic events in varying ways depending their timing and distribution, several data sets were defined. Formally, each volcanic event in the following data sets is considered to be independent of the other volcanic events in the data set.

The regional recurrence rate used to estimate probability also depends on the definition of volcanic events. For example, grouping volcanic vents by alignments in the Yucca Mountain region results in fewer Quaternary volcanic events, and consequently a lower regional recurrence rate, than results from treating individual vents as independent volcanic events. In practice, a range of recurrence rates must be considered in the PVHA because of uncertainties about the appropriate values to use. Volcanism in the Yucca Mountain region may be episodic or respond to external factors, such as geologic strain rate, in ways not well understood at the present (Connor and Conway, 2000) (see Articles). Furthermore, the area likely affected by volcanic events varies with the definition of volcanic events. Therefore, a range of

10/
131

parameters is used in the PVHA (Connor and Hill, 1995; Connor et al., 2000) (see Articles).

Data set 1 includes known Miocene through Quaternary vents mapped in the Yucca Mountain region, including several magnetic anomalies, assumed to be Miocene or younger in age, that have not been drilled. See the data table for details. This data set does not include some petrologically distinct basalts of Miocene age, such as the 11.2 Ma Solitario Canyon dike. The number of volcanic events is 47 in data set 1. An average regional recurrence rate for this data set is approximately 5 events per million years. This data set corresponds to data set 1 used in Connor and Hill (1995) (see Articles) and treats individual volcanic vents as independent volcanic events. See the Data Table for details.

Data set 2 includes all known Pliocene - Quaternary vents mapped in the Yucca Mountain region, including several magnetic anomalies, assumed to be Miocene or younger in age, that have not been drilled. Several closely spaced vents have been grouped, and are treated as single volcanic events. For example, the two Little Cones are treated as a single volcanic event in this data set. The number of volcanic events is 20 in data set 2. This data set corresponds to data set 2 used in Connor and Hill (1995) (see Articles). Use of this data set implies that the distribution of Miocene vents has little influence on potential patterns of basaltic volcanism during the performance period of the repository. An average regional recurrence rate for this data set is approximately 6 volcanic events per million years. See the Data Table for details.

Data set 3 includes all known Quaternary vents mapped in the Yucca Mountain region. Magnetic anomalies are not included in this data set. Use of this data set implies that the expected distribution of future volcanic activity is best defined by the distribution of volcanism in the recent geologic past (last one million years), and that older volcanic vents are not relevant to the analysis. The number of volcanic events is 8 in data set 3. An average regional recurrence rate for data set 3 is 8 volcanic events per million years. See the Data Table for details.

Data set 4 includes three events. These are the Quaternary Crater Flat volcano alignment, taken as centered on Red Cone, the Sleeping Butte alignment, taken as centered on Hidden Cone, and Lathrop Wells. Use of this definition implies that vents that form alignments are not independent volcanic events, and that Quaternary volcanism is the best guide to future volcanic activity. The distribution of older volcanoes or volcanic alignments is not considered as part of calculations using this data set. An average recurrence rate for this data set is 3 volcanic events per million years. See the Data Table for details.

Data set 5 includes all of the vent locations reported in the Data Table. The number of volcanic events is 64 in data set 5. An average recurrence rate for this data set is 6 volcanic events per million years.

[return to top](#)

[return to Table of Contents](#)

11/131

Basis for the PVHA

The geologic and statistical basis for the PVHA for the Yucca Mountain site is provided in Connor and Hill [1995] and Connor et al. [2000] ([see Articles](#)). Briefly, kernel density estimators are used to calculate a probability surface directly from the location and timing of past, discrete volcanic events. As a result, kernel estimators are sensitive to patterns, such as vent clustering, commonly observed in basaltic volcano distributions. Furthermore, the resulting probability surfaces do not have the abrupt changes in probability that must be introduced in spatially homogeneous Poisson models. Thus, the kernel methods eliminate the need to define zones of volcanic activity a priori.

Two types of kernel density functions are provided: Gaussian and Epanechnikov kernels.

The Gaussian kernel is defined as:

$$k_i = 2 \pi \exp[-1/2 (d_i/h)^2]$$

where d_i is the distance from the point s to the i th volcano and h is the smoothing parameter.

The Epanechnikov kernel is defined as:

$$k_i = (2/\pi)[1 - (d_i/h)^2], \text{ if } (d_i/h)^2 < 1 \text{ and}$$

$$k_i = 0, \text{ otherwise}$$

In each case, the spatial recurrence rate of volcanic events in the 1 x 1 km area about the point s , given the occurrence of a volcanic event in the system is given by:

$$\lambda(s) = 1/(nh^2) \text{ Sum } \{ \text{from } i=1 \text{ to } n \} k_i.$$

The probability of volcanic events within the area of the repository site is then:

$$P[N \geq 1] = 1 - \exp[-t \lambda(s) \text{ sum } \{ \text{over area } a \} \lambda(s)]$$

where t is the time interval of interest, $\lambda(t)$ is the temporal recurrence rate of volcanic events in the magmatic system, and a is the effective area, an area within which a volcanic event might occur and disrupt the repository.

Assuming that $\lambda(s)$ does not vary on the scale of the repository:

$$P[N \geq 1] = 1 - \exp[-t \lambda(s) a \lambda(s)]$$

These techniques have been tested using the recurrence rates of volcanism and patterns of volcanic activity in other volcanic fields. In particular, these models have been tested in the Springerville volcanic field, Arizona ([see Articles](#)) An introduction to basaltic volcanic fields is given in Connor and Conway [2000] ([see Articles](#)).

12/131

[return to top](#)

[return to Table of Contents](#)

Explanation of Parameters Used in PVHA_YM

Several parameters may be varied in the analysis. These parameters include the smoothing factor used in the kernel functions, the regional recurrence rate of volcanic activity, the effective area of the repository, and the time interval for which probability calculations are made.

The smoothing factor controls how probability is distributed about existing volcanic events, which are treated as a point process. Using a small smoothing factor tends to concentrate probability near existing volcanoes; a large smoothing factor results in a more even distribution of probability across the map area. The two kernel estimators, Gaussian and Epanechnikov, rely on different functions and, as a result, the smoothing factor has a different effect in each case. In the Gaussian kernel, the smoothing factor is equivalent to the standard deviation of a symmetric, bivariate Gaussian distribution. For an Epanechnikov kernel, the smoothing factor determines the radius of the Epanechnikov distribution function about each point. The probability is zero at distances greater than the smoothing factor from the point, and nonzero at distances less than the smoothing factor from the point (i.e., volcanic event).

Methods of estimating the smoothing factor are described in Connor et al. (2000) ([see Articles](#)). The kernel function (Gaussian or Epanechnikov) may be recast in polar coordinates as a cumulative distribution function. In this form, the cumulative kernel function can then be compared to observed nearest-neighbor distances between volcanic events. Using this approach, Connor et al (2000) ([see Articles](#)) found that values of the smoothing factor greater than 7 km for the Gaussian kernel yield conservative estimates of probability. For Epanechnikov kernels, a smoothing factor greater than 18 km yields conservative results. The larger value for the Epanechnikov kernel compared to the Gaussian kernel results from the definition of the smoothing factor as the limit of the probability distribution in the former case, and the standard deviation of the probability distribution in the latter case.

Probability models rely on estimates of the expected regional recurrence rate of volcanism in order to estimate the probability of future volcanic activity. In the Yucca Mountain region, estimates of the regional recurrence rate vary between 2 and 12 volcanic events per million years, with variations in the definitions of volcanic events accounting for at least part of this range. Furthermore, it is uncertain whether the rate of volcanic activity in the Yucca Mountain region is stationary, nonstationary, or episodic. As a result, uncertainty in the regional recurrence rate of volcanism is significant.

The effective area of the repository includes the actual footprint of the repository, plus the area within which a volcanic event may occur and cause volcanic disruption of the repository. This additional area is included because volcanic events are treated as

points in the analysis, but in reality affect an area about that point, due to dike injection and the formation of multiple vents. For data sets 1 and 2, Connor and Hill (1995) (see Articles) used an effective area of 8 square kilometers. If alignments are treated as single events, such as in data set 4, significantly larger areas should be used (e.g., 20-30 square kilometers). Alternatively, vent alignment length and orientation may be considered explicitly (Connor et al., 2000) (see Articles), but this feature has not been implemented in version 1.0 of PVHA_YM.

13/
131

The time interval for which probability is estimated may be varied to calculate annual probabilities, or to calculate probabilities of volcanic disruption of the proposed site for different performance periods. The probability calculation assumes that the time interval considered and the regional recurrence rate are independent.

[return to top](#)

[return to Table of Contents](#)

Applets

PVHA_YM calculations are performed using the following java applets.

ProbGraph1 calculates the probability of a volcanic event in an effective area about the repository, given a temporal recurrence rate, time interval of interest, and selected data set. This estimate uses a Gaussian kernel and plots probability as a function of the smoothing parameter.

ProbGraph2 calculates the probability of a volcanic event in an effective area about the repository, given a temporal recurrence rate, time interval of interest, and selected data set. This estimate uses an Epanechnikov kernel and plots probability as a function of the smoothing parameter.

ProbMap1 calculates the probability surface for the Yucca Mountain region using a selected data set and smoothing parameter. This estimate uses a Gaussian kernel and plots the conditional probability of a volcanic event within a 1 x 1 km area, given an event in the magmatic system.

ProbMap2 calculates the probability surface for the Yucca Mountain region using a selected data set and smoothing parameter. This estimate uses an Epanechnikov kernel and plots the conditional probability of a volcanic event within a 1 x 1 km area, given an event in the magmatic system.

[return to top](#)

[return to Table of Contents](#)

Articles

Two articles are included in this document that describe the development of kernel density estimators and their application to the CNWRA PVHA for Yucca Mountain. These articles are:

Connor et al., 2000, Geologic Factors controlling patterns of small volume basaltic volcanism: Application to a volcanic hazards assessment at Yucca Mountain, Nevada

Connor and Hill, 1995, Three nonhomogeneous Poisson models for the probability of basaltic volcanism: Application to Yucca Mountain, Nevada

Application of PVHA techniques to analogous basaltic volcanic fields, for the purposes of validating the use of density kernels in PVHA, is described in:

Condit and Connor, 1996, Recurrence rates of basaltic volcanism: An example from the Springerville volcanic field, Arizona.

An introduction to basaltic volcanic fields is provided in:

Connor and Conway, 2000, Basaltic volcanic fields.

The intent of this article is to provide background on the nature of basaltic volcanic fields, their physical characteristics, evolution, and origins.

[return to top](#)

[return to Table of Contents](#)

Code Limitations

PVHA_YM contains codes used to estimate probabilities of volcanic events within the area of the proposed Yucca Mountain repository to assist in reviewing the anticipated DOE license application. Nevertheless, a complete PVHA analysis for the proposed repository cannot be accomplished using PVHA_YM alone. For example, Connor et al. (2000) ([See Articles](#)) showed that geologic structure plays an important role in controlling the distribution of basaltic volcanism in the Yucca Mountain region. Their analysis showed the affect of including geologic structure in the PVHA was to increase the probability of volcanic disruption of the site (Connor et al. , 2000) ([See Articles](#)). The influence of geologic structure is not incorporated in version 1.0 of PVHA_YM. Furthermore, volcanic events may effect a larger area than represented by a single vent. In basaltic volcanic fields, volcanic events may be defined as volcanic alignments, with length and orientation. For example, in Data set 4, the Quaternary

15/1/31

Crater Flat alignment is treated as a single volcanic event. The areal dimension of volcanic events is not incorporated explicitly in version 1.0 of PVHA_YM, although the effective area parameter may be changed to help illustrate the influence of event area on probability estimates.

[return to top](#)

[return to Table of Contents](#)

16/131

/**

Program Name: PVHA_YM

Class Name: ProblGraph

Date: 02/11/00

Release Version: 1.0

Client Name: USNRC

U. S. Nuclear Regulatory Commission
NRC Office of Nuclear Material Safety and Safeguard
Division of Waste Management

NRC Contract: NRC 02-97-009

NRC Contact: Dr. John Trapp (301) 415-8063

CNWRA Contact: Dr. Charles Connor (210) 522-6649

Center For Nuclear Waste Regulatory Analyses
Southwest Research Institute
6220 Culebra Rd.
San Antonio, TX, 78238-5166, USA
cconnor@swri.edu

Documentation:
PVHA_YM version 1.0 - Probabilistic Volcanic
Hazard Assessment Methods for a Proposed
High-Level Radioactive Waste Repository
at Yucca Mountain, Nevada

CNWRA 2000-02

NUREG-Series Designator: N/A

DISCLAIMER

"This computer code / material was prepared as an account of work performed by the Center for Nuclear Waste Regulatory Analyses (CNWRA) for the Division of Waste Management of the Nuclear Regulatory Commission (NRC), an independent agency of the United States Government. Neither the developer(s) of the code nor any of their sponsors make any warranty, expressed or implied, or assume any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represent that its use would not infringe on privately-owned rights."

"In no event unless required by applicable law will the sponsors or those who have written or modified this code, be liable for damages, including any loss of profits, lost monies, or other special, incidental or consequential damages arising out of the use or inability to use the program (including but not limited to loss of data or data rendered inaccurate or losses sustained by third parties for failure of the program to operate with other programs), even if you have been advised of the

17/131

possibility of such damages or for any claim by any other party."

```
*****
*****
```

by C. Connor

```
*****
*****/
```

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

```
import graph.*;
import graph.ContourProb;
import graph.Volcano;
```

```
/*
```

```
*****
```

```
**
```

```
**           Applet ProbMap2
```

```
**
```

```
*****
```

Based on code originally written in TrueBasic by C Connor, 1992-1994
Graphics portion of this code modified from code written by Leigh Brookshaw

Purpose:

applet ProbMap2 calculates the conditional probability of a new volcano forming within a 1 km x 1 km region, given that a new volcano forms in the region, using an epanechnikov kernel

This applet uses the Restimate class to estimate probability and contours the results

This applet uses the Volcano class to select data sets

This applet uses the ContourProb class to plot the map

```
*****/
```

```
// classes to handle defined exceptions
```

```
class OddSF extends Exception {}
```

```
class OddData extends Exception {}
```

```
public class ProbMap2 extends Applet {
```

```
    ContourProb graph          = new ContourProb();    // Graph class to do the plottin
    Axis xaxis;
    Axis yaxis;
    DataSet data;
    Volcano volcano = new Volcano();
```

```
    TextField cinput          = new TextField(4);      // Number of contours
    TextField xinput          = new TextField(4);      // X grid dimension
    TextField yinput          = new TextField(4);      // Y grid dimension
    TextField xmininput       = new TextField(10);     // Minimum x value input
    TextField xmaxinput       = new TextField(10);     // Maximum x value input
    TextField ymininput       = new TextField(10);     // Minimum y value input
    TextField ymaxinput       = new TextField(10);     // Maximum y value input
    TextField vdatainput      = new TextField(2);      // data set to plot
    TextField sfinput         = new TextField(4);      // smoothing factor input
```

```

Button plot          = new Button("Calculate"); // Button to plot it.
public void init() {

    // define the title with a time date stamp and version ###
    Date d = new Date();
    int realmonth = d.getMonth() + 1;
    int realyear = d.getYear() + 1900;
    int realmin = d.getMinutes();
    String titlestring;

    if (realmin < 10) {
        titlestring = "Epanechnikov Probability Map          "
            + realmonth + "/" + d.getDate() + "/" + realyear
            + ", " + d.getHours() + ":" + "0" + realmin
            + "          PVHA_YM version 1.0";
    }
    else {titlestring = "Epanechnikov Probability Map          "
        + realmonth + "/" + d.getDate() + "/" + realyear
        + ", " + d.getHours() + ":" + realmin
        + "          PVHA_YM version 1.0";
    }

    Label title = new Label(titlestring, Label.CENTER);

    Panel panel          = new Panel();
    GridBagLayout gridbag = new GridBagLayout();
    GridBagConstraints c = new GridBagConstraints();
    Font font           = new Font("Helvetica",Font.PLAIN,14);

    title.setFont(new Font("Helvetica",Font.PLAIN,14));

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

    // Data from .html
    vdatainput.setText(getParameter("DATASET"));
    sfinput.setText(getParameter("SMOOTH"));
    cinput.setText(getParameter("CONTOURS"));
    xinput.setText(getParameter("XGRID"));
    yinput.setText(getParameter("YGRID"));
    xmininput.setText(getParameter("XMIN"));
    xmaxinput.setText(getParameter("XMAX"));
    ymininput.setText(getParameter("YMIN"));
    ymaxinput.setText(getParameter("YMAX"));

    panel.setLayout(gridbag);
    /*
    ** create labels for all the text input fields
    */
    Label clabel      = new Label("Number of Contours");
    Label xlabel      = new Label("X grid dimension");
    Label ylabel      = new Label("Y grid dimension");
    Label xminlabel   = new Label("Minimum Easting");
    Label xmaxlabel   = new Label("Maximum Easting");
    Label yminlabel   = new Label("Minimum Northing");
    Label ymaxlabel   = new Label("Maximum Northing");
    Label vdatalabel  = new Label("Data Set (1-5)");
    Label sflabel     = new Label("Smoothing Factor (km)");
    /*
    ** Set the fonts and colors to use
    */
    clabel.setFont(font);
    xlabel.setFont(font);

```

18/131

19/
/131

```

ylabel.setFont (font);
vdatalabel.setFont (font);
sflabel.setFont (font);
xminlabel.setFont (font);
xmaxlabel.setFont (font);
yminlabel.setFont (font);
ymaxlabel.setFont (font);

cinput.setFont (font);
cinput.setBackground (Color.lightGray);
xinput.setFont (font);
xinput.setBackground (Color.lightGray);
yinput.setFont (font);
yinput.setBackground (Color.lightGray);
vdatainput.setFont (font);
sfinput.setBackground (Color.lightGray);
sfinput.setFont (font);
vdatainput.setBackground (Color.lightGray);
xmininput.setFont (font);
xmininput.setBackground (Color.lightGray);
xmaxinput.setFont (font);
xmaxinput.setBackground (Color.lightGray);
ymininput.setFont (font);
ymininput.setBackground (Color.lightGray);
ymaxinput.setFont (font);
ymaxinput.setBackground (Color.lightGray);
plot.setFont (font);
plot.setBackground (Color.cyan);

/*
** First row of the GridBag contains the plot
*/
c.weightx = 1.0;
c.weighty = 1.0;
c.gridwidth = 4;
c.gridwidth=GridBagConstraints.REMAINDER;
c.fill = GridBagConstraints.BOTH;

gridbag.setConstraints (graph, c);

/*
** Second row of the gridBag contains the function input and the
** plotit button
*/
c.fill = GridBagConstraints.NONE;
c.weightx=0.0;
c.weighty=0.0;
c.gridheight=1;

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

c.anchor = GridBagConstraints.WEST;
gridbag.setConstraints (vdatainput, c);

c.fill = GridBagConstraints.WEST;
c.gridwidth=GridBagConstraints.REMAINDER;

gridbag.setConstraints (plot, c);

/*
** Third row contains the Number of contours and smoothinf factor

```

20/1/31

```
*/
c.gridwidth=1;
c.anchor = GridBagConstraints.EAST;
gridbag.setConstraints(clabel,c);

c.anchor = GridBagConstraints.WEST;
gridbag.setConstraints(cinput,c);

c.anchor = GridBagConstraints.EAST;
gridbag.setConstraints(sflabel,c);

c.anchor = GridBagConstraints.WEST;
c.gridwidth=GridBagConstraints.REMAINDER;
gridbag.setConstraints(sfinput,c);

/*
** Fourth row contains the grid dimensions
*/
c.gridwidth=1;
c.anchor = GridBagConstraints.EAST;
gridbag.setConstraints(xlabel,c);

c.anchor = GridBagConstraints.WEST;
gridbag.setConstraints(xinput,c);

c.anchor = GridBagConstraints.EAST;
gridbag.setConstraints(ylabel,c);

c.anchor = GridBagConstraints.WEST;
c.gridwidth=GridBagConstraints.REMAINDER;
gridbag.setConstraints(yinput,c);

/*
** Fifth row contains the x range of the grid
*/

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

c.anchor = GridBagConstraints.WEST;
gridbag.setConstraints(xmininput,c);

c.anchor = GridBagConstraints.EAST;
gridbag.setConstraints(xmaxlabel,c);

c.anchor = GridBagConstraints.WEST;
c.gridwidth=GridBagConstraints.REMAINDER;
gridbag.setConstraints(xmaxinput,c);

/*
** Sixth row contains the y range of the grid
*/

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

c.anchor = GridBagConstraints.WEST;
gridbag.setConstraints(ymininput,c);

c.anchor = GridBagConstraints.EAST;
gridbag.setConstraints(ymaxlabel,c);
```

21/131

```

c.anchor = GridBagConstraints.WEST;
c.gridwidth=GridBagConstraints.REMAINDER;
gridbag.setConstraints(ymaxinput,c);

// display the fields
panel.add(graph);
panel.add(vdatalabel);
panel.add(vdatainput);

panel.add(plot);
panel.add(clabel);
panel.add(cinput);
panel.add(sflabel);
panel.add(sfinput);
panel.add(xlabel);
panel.add(xinput);
panel.add(ylabel);
panel.add(yinput);
panel.add(xminlabel);
panel.add(xmininput);
panel.add(xmaxlabel);
panel.add(xmaxinput);
panel.add(yminlabel);
panel.add(ymininput);
panel.add(ymaxlabel);
panel.add(ymaxinput);

//label the graph axes
xaxis = graph.createXAxis();
xaxis.setTitleText("Easting");

yaxis = graph.createYAxis();
yaxis.setTitleText("Northing");

// set the graph colors
graph.setDataBackground(new Color(230,240,240));
graph.setBackground(new Color(240,250,250));
graph.setContourColor(Color.black);
graph.setLabelledContourColor(Color.red);

// set the label interval
// default condition -> contours not labelled
graph.setLabelLevels(5);
graph.setDrawLabels(false);

// since its a map, make it square
graph.square = true;

plot();
}

// the plot class does the work of creating the array to contour
// and interpeting the parameters from input
void plot() {
    int levels;
    int nx;
    int ny;
    int i, j;
    int setnum;
    double xmax;
    double xmin;
    double ymax;

```

22 / 131

```

double ymin;
double x, y;
int count = 0;
boolean error = false;
double smoothing_factor;

try {
    graph.setNLevels( Integer.parseInt(cinput.getText()) );
} catch(Exception e) {
    this.showStatus("Error with number of contour levels!");
    System.out.println(
        "Number of contour levels error "+e.getMessage());
    return;
}

try {
    nx    = Integer.parseInt(xinput.getText());
    ny    = Integer.parseInt(yinput.getText());

} catch(Exception e) {
    this.showStatus("Error parsing grid dimensions!");
    System.out.println(
        "Error parsing grid dimensions! "+e.getMessage());
    return;
}

try {
    xmax = Double.valueOf(xmaxinput.getText()).doubleValue();
    xmin = Double.valueOf(xmininput.getText()).doubleValue();
} catch(Exception e) {
    this.showStatus("Error with X range!");
    System.out.println("X range error "+e.getMessage());
    return;
}

try {
    ymax = Double.valueOf(ymaxinput.getText()).doubleValue();
    ymin = Double.valueOf(ymininput.getText()).doubleValue();
} catch(Exception e) {
    this.showStatus("Error with Y range!");
    System.out.println("Y range error "+e.getMessage());
    return;
}

try {
    setnum = Double.valueOf(vdatainput.getText()).intValue();
} catch(Exception e) {
    this.showStatus("Illegal data set value: enter an integer!");
    System.out.println("Dataset error "+e.getMessage());
    return;
}

// check for illegal data sets
try {
    if (setnum < 1 || setnum > 5) throw (new OddData() );
} catch(Exception e4) {
    this.showStatus("Error: Data Set between 1 and 5");
    System.out.println("Data Set error ");
    return;
}

try {
    smoothing_factor = Double.valueOf(sfinput.getText()).doubleValue();
} catch(Exception e) {
    this.showStatus("Illegal smoothing factor");
}

```

23
/31

```

        System.out.println("smoothing factor error "+e.getMessage());
        return;
    }

    // Smoothing factor must be positive real number
    try {
        if (smoothing_factor <= 0.0) throw (new OddSF() );
    } catch(Exception e3) {
        this.showStatus("Error: Make Smoothing Factor > 0");
        System.out.println("Smoothing Factor error ");
        return;
    }

    if(xmin>=xmax || ymin>= ymax ) {
        this.showStatus("Error with Grid range!");
        System.out.println("grid range error! ");
        return;
    }
    if( nx < 2 || ny < 2 ) {
        this.showStatus("Error with Grid dimensions!");
        System.out.println("grid dimension error!");
        return;
    }

    graph.setRange(xmin,xmax,ymin,ymax);

    //the volcano data set is determined from setnum
    volcano.setVolcanoset(setnum);

    //array is the grid to plot
    double array[] = new double[nx*ny];

    this.showStatus("Calculating Grid Values!");
    count = 0;

    // instantiate the volcanodata set
    volcano.getVolcanoes();

    // instaniate lambdal - a new instance of Restimate
    Restimate lambdal = new Restimate(smoothing_factor, Volcano.nvolcanoes);

    // fill the array
    for(j=0; j<ny; j++) {
        y = ymin + j*(ymax-ymin)/(ny-1);
        for(i=0; i<nx; i++) {
            x = xmin + i*(xmax-xmin)/(nx-1);
            try {

                // use the epanechnikov method in Restimate
                array[count++] = lambdal.epanh(Volcano.vs, x,y);
            } catch(Exception e) {
                array[count++] = 0.0;
                error = true; }
        }
    }

    //instantiate graph with array and array dimensions
    graph.setGrid(array,nx,ny);

    this.showStatus("Calculating Contours!");

    graph.repaint();
}

```

23A / 131

```
//keep track of Calculate events
public boolean action(Event e, Object a) {
    if(e.target instanceof Button) {
        if( plot.equals(e.target) ) {
            plot();
            return true;
        }
    }
    return false;
}
}
```

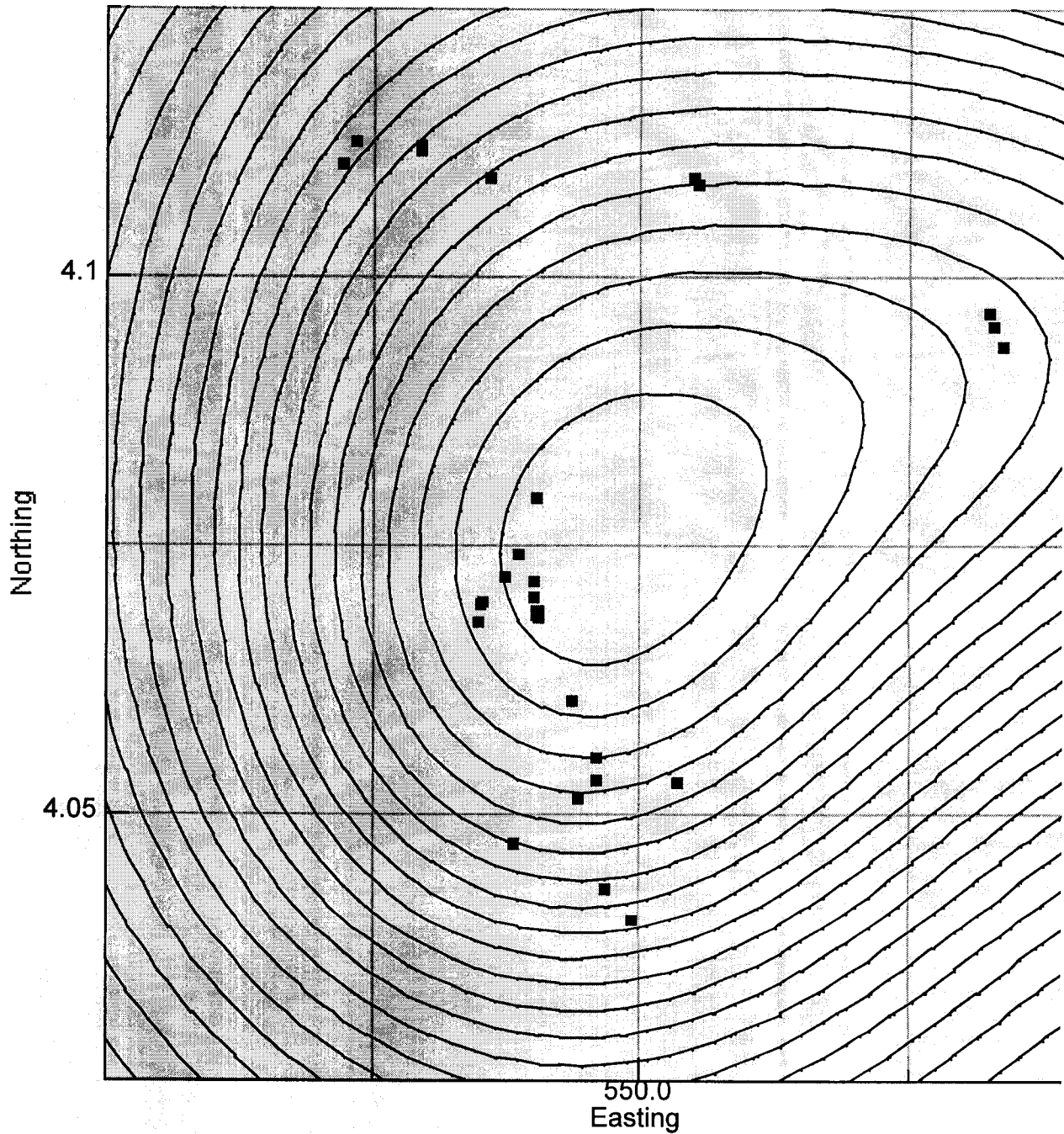
24/131

Gaussian Probability Map

2/15/2000, 16:01

PVHA_YM version

$\times 10^6$



Data Set (1-5)	1	<input type="button" value="Calculate"/>	
Number of Contours	20	Smoothing Factor (km)	28.0
X grid dimension	50	Y grid dimension	50
Minimum Easting	500000.0	Maximum Easting	600000.0
Minimum Northing	4025000.0	Maximum Northing	4125000.0

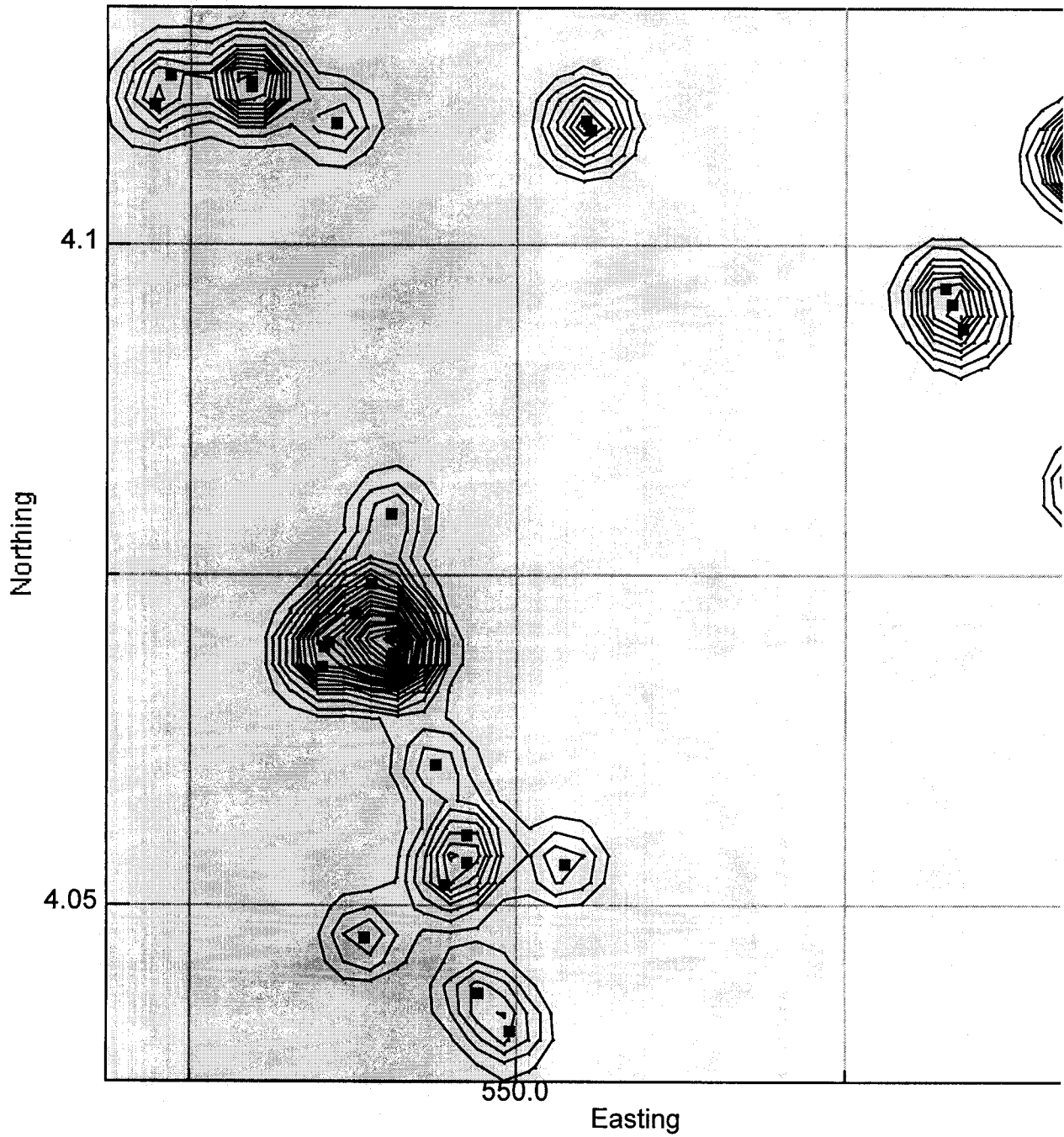
25/131

Gaussian Probability Map

2/15/2000, 16:01

PVHA_YM version

$\times 10^6$



Data Set (1-5)	1	<input type="button" value="Calculate"/>	
Number of Contours	20	Smoothing Factor (km)	2.0
X grid dimension	50	Y grid dimension	50
Minimum Easting	500000.0	Maximum Easting	600000.0
Minimum Northing	4025000.0	Maximum Northing	4125000.0

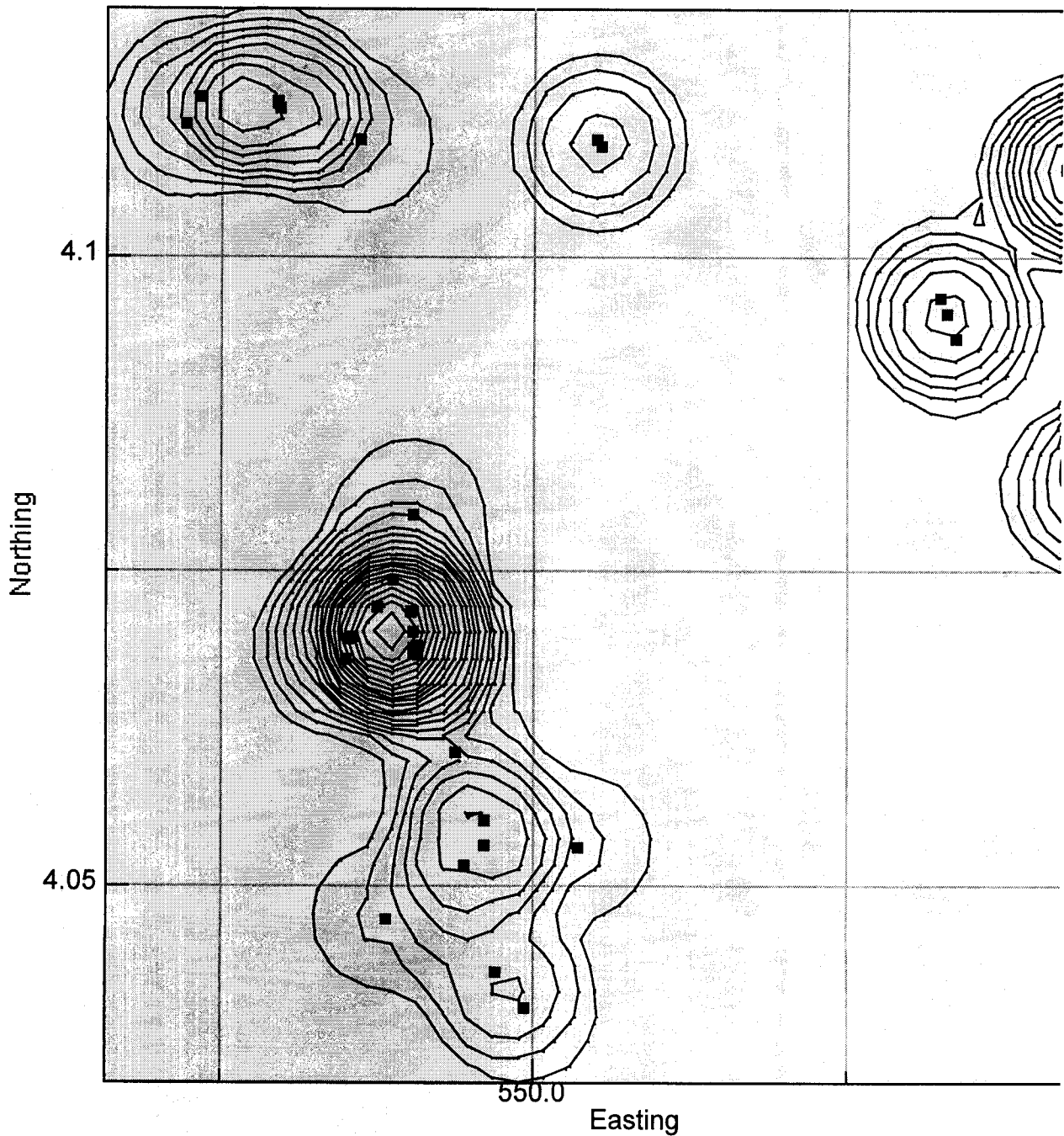
26/131

Epanechnikov Probability Map

2/15/2000, 16:10

PVHA_YM versi

$\times 10^6$



Data Set (1-5)	1	Calculate	
Number of Contours	20	Smoothing Factor (km)	8.0
X grid dimension	50	Y grid dimension	50
Minimum Easting	500000.0	Maximum Easting	600000.0
Minimum Northing	4025000.0	Maximum Northing	4125000.0

27/131



UNITED STATES
NUCLEAR REGULATORY COMMISSION
WASHINGTON, D.C. 20555-0001

January 30, 2001

"PVHA-YM"
→ Software Folder

Southwest Research Institute
Attn: R. B. Kalmbach, Director, Contracts
6220 Culebra Road
P.O. Drawer 28510
San Antonio, TX 78228-0510

Subject: Copyright PVHA-YM Software Code Agreement under Contract NRC-02-97-009

Reference: Letter R. B. Kalmbach, SwRI, to B. Maehan, NRC Dtd. 12/5/00

Dear Mr. Kalmbach:

NRC hereby grants SwRI approval to copyright the subject software subject to the following condition:

The contractor recognizes that the Government reserves the right to distribute copies of the subject computer software, and any modifications to the software funded by the NRC, to the parties and participants in the Nuclear Waste Policy Act, as Amended (NWPAA) proceedings or any other organization that is directly involved in evaluating actions related to the NWPAA, and to the parties and participants in any other NRC regulatory activities involving probabilistic volcanic hazard assessment. These government rights to distribution of the subject software may include but are not limited to applicants, licensees, States, affected Indian tribes, affected units of local government, nuclear industry and trade groups such as the Electric Power Research Institute, Edison Electric Institute and public interest groups. Further, consistent with Section 1.6 of the contract: NRC-02-97-009-73 CONTRACTOR ORGANIZATIONAL CONFLICTS OF INTEREST (JAN 1993), the contractor agrees that it will not sell or distribute the software to, or for the use of such parties or participants, and that it will not provide technical services relating to the software to such parties or participants, including parties or participants in the NWPAA licensing process. In addition, the contractor agrees to include in any licensing agreement that it may enter into with a third party such limitations as are necessary to preserve the rights of the Government, and limit the sale and distribution of the software as described above. These reserved rights are in addition to the license granted the Government under paragraph "c", of Section 1.6 of Contract NRC-02-97-009, 52.227-14 RIGHTS IN DATA-GENERAL (JUN 1987) ALTERNATE 1 (JUN 1987) ALTERNATE III (JUN 1987) ALTERNATE V (JUN 1987).

cc: Ems BH
Dir CC
Kalmbach
McLeod
Maldonado

28/131

Should you have any questions regarding this matter, please call me on 301-415-6730.

Sincerely,


Barbara D. Meehan, Contracting Officer
Contract Management Branch No. 2
Division of Contracts and
Property Management

cc: Wesley C. Patrick, SwRI/CNWRA
Buhdi Sagar, SwRI/CNWRA

29 / 131

CNWRA A center of excellence in earth sciences and engineering

A Division of Southwest Research Institute
6220 Culebra Road • San Antonio, Texas, U.S.A. 78228-5166
(210) 522-5160 • Fax (210) 522-5155

*File with/in
PVHA_YM
SOFTWARE
Folder*

February 18, 2000
Contract No. NRC-02-97-009
Account No. 20.01402.462

U.S. Nuclear Regulatory Commission
ATTN: Dr. John S. Trapp, NMSS
Mail Stop 7 C6
Two White Flint North
11545 Rockville Pike
Rockville, MD 20852-2738

Subject: Transmittal of CNWRA Intermediate Milestone 01402.462.050: PVHA_YM Version 1.0—Probabilistic Volcanic Hazard Assessment Methods for a Proposed High-Level Radioactive Waste Repository at Yucca Mountain, Nevada, CNWRA 2000-02

Dear Dr. Trapp:

The enclosed Intermediate Milestone (01402.462.050), entitled "PVHA_YM Version 1.0—Probabilistic Volcanic Hazard Assessment Methods for a Proposed High-Level Radioactive Waste Repository at Yucca Mountain, Nevada," documents methods and computer codes developed by NRC and CNWRA staff to perform an independent probabilistic volcanic hazard assessment (PVHA) for the proposed high-level radioactive waste repository at Yucca Mountain, Nevada. NRC and CNWRA staff will use these methods and computer codes to review the PVHA performed by the DOE as part of their license application to NRC. PVHA_YM was developed as part of the Igneous Activity Key Technical Issue (IA KTI) with the goal of streamlining license application review with regard to this sometimes contentious issue. During license application review, PVHA_YM will be used by NRC and CNWRA staff to both review expected post-closure performance and to evaluate DOE compliance with 10 CFR 63.113. In this context, one important goal of development of PVHA_YM is to provide DOE with a clear and practical understanding of the approach NRC and CNWRA staff will use in their implementation of the review methods outlined in section 4.2.1.3.10 of the Yucca Mountain Review Plan.

The CD-ROM contains documentation, computer codes, and previously published articles that summarize CNWRA and NRC efforts to understand and clarify scientific issues related to the probability of volcanic disruption of the proposed repository during the performance period. Files contained on the CD-ROM are prepared in Hypertext Markup Language (HTML) format and can be viewed on any computer platform using standard web browser software (e.g., Netscape Version 4.5). Execution of the java applications contained on the CD-ROM requires that Java Virtual Machine (JVM) language, version 1.1 or higher, be enabled on the web browser.



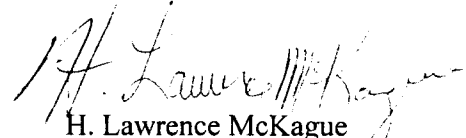
Washington Office • Twinbrook Metro Plaza #210
12300 Twinbrook Parkway • Rockville, Maryland 20852-1606

30/131

John S. Trapp
February 18, 2000
RE: IM 20.01402.462.050
Page 2

This intermediate milestone is listed incorrectly as IM 01402.461.050 in CNWRA Operations Plans for the Repository Program, Revision 14, Change 1. The operation plan will be corrected by noting this change in table 2 in the PMPR for period 5. If you have further questions about this IM, please contact Dr. Chuck Connor (210-522-6649), or me at (210-522-5183).

Sincerely yours,



H. Lawrence McKague
Geology and Geophysics Manager

HLM/adm
Enclosures

cc:	J. Linehan	J. Holonich	W. Patrick	T. Nagy
	D. DeMarco	W. Reamer	CNWRA Directors	C. Connor
	B. Meehan	S. Wastler	CNWRA Element Managers	B. Hill
	J. Greeves	D. Brooks	P. Maldonado	J. Stamatakos



CENTER FOR NUCLEAR WASTE REGULATORY ANALYSES QUALITY ASSURANCE SURVEILLANCE REPORT

PROJECT NO.: 20.01402.159

REPORT NO.: 2000-13

PAGE 1 OF 2

SURVEILLANCE SCOPE: Review of CNWRA Developed Scientific and Engineering Software to determine whether the documentation present in the CNWRA Software Working Records Folders is adequate.

REFERENCE DOCUMENTS: Technical Operating Procedure-018, Development and Control of Scientific and Engineering (S&E) Software; QAP-004, Surveillance Control; Nonconformance Report 2000-03.

STARTING DATE: 3/7/2000

ENDING DATE: 6/9/2000

QA REPRESENTATIVE: B. Mabrito

PERSONS CONDUCTING TEST/EXAM/ACTIVITY: Various CNWRA staff working on Developed S&E software.

SATISFACTORY FINDINGS: During the course of this surveillance, CNWRA Developed S&E software and documentation was checked and contact made with CNWRA staff who worked with the software. In each case, the particular S&E software folder was reviewed for completeness and where no Design Verification Report (DVR) was located, the objective evidence in the folder was compared to the DVR form questions and discussions were held with cognizant CNWRA staff. The list of Developed S&E software reviewed is included in Attachment A.

In each case, key elements of the DVR were compared against that which was included in each software folder in the QA working records. Also, the previous version of the software code documentation was checked to ensure that the earlier DVR had been properly completed. The later version of the software documentation showed the specific changes made through the Software Change Reports. Based on this review, it is clear that although in a few cases no DVR was accomplished, product quality did not suffer. The minor enhancements and "bug" fixes made to TPA Version 3.2.3 and 3DStress Version 1.3.1 and 1.3.2 software were clearly identified and controlled so that the CNWRA product being delivered met the client's requirements.

UNSATISFACTORY FINDINGS: None.

NONCONFORMANCE REPORT NO.: None.

ATTACHMENTS: Attachment A.

RECOMMENDATIONS/ACTIONS: N/A.

APPROVED: 
CENTER DIRECTOR OF QUALITY ASSURANCE

DATE: 6/12/2000

DISTRIBUTION:

ORIGINAL - CENTER QA DIRECTOR QA Records
ORIGINATOR
PRINCIPAL INVESTIGATORS OF EACH CODE
ELEMENT MANAGERS
B. Sagar, H. Garcia

<u>NAME OF S&E SOFTWARE</u>	<u>DESIGN VERIFICATION REPORT</u>		<u>NOTES</u>
3DStress Version 1.2	Present	Dated 5/8/97	
3DStress Version 1.3	Present	Dated 8/7/98	
3DStress Version 1.3.1	Not Present		Software Release Notice Dated 7/15/99
3DStress Version 1.3.2	Not Present		Software Release Notice Dated 9/16/99
ASHPLUME Version 1.0	Present	Dated 6/23/97	
BREATH Version 1.1	Not Present		Software Release Notice Dated 9/21/95
BREATH Version 1.2	Present	Dated 9/17/97	
EBSPAC Version 1.0	Present	Dated 5/15/97	
EBSPAC Version 1.1	Present	Dated 6/17/97	
FAULTING Version 1.0	Not Present		Software Release Notice Dated 1/21/98 Module put under TPA Code and controlled in that manner.
GEOINVRT Version 1.0	Software Code Not Finished		Software Requirements Description only.
HAZINFO Version 1.0	Software Code Not Finished		Software Requirements Description only.
MULTIFLO Version 1.2	Present	Dated 3/2/2000	
MULTIFLO Version 2.0	Software Code Not Finished		Software Requirements Description only.
PVHA Version 1.0 <i>PUHVIEW 1.0 ASN 6/13/2000</i>	Present	Dated 2/15/2000	
SUFLAT Version 1.0	Not Present		Element Manager (EM) determined that this software has not been used for regulatory reviews and will not be used for such work. EM requested the folder be archived in QA Records to reflect previous efforts on code.
TECTRAN Version 1.0	Software Code Not Finished		Software Requirements Description only.
TPA Version 3.2	Present	Dated 7/17/98	
TPA Version 3.2 (PP) Beta	Present	Dated 11/25/98	
TPA Version 3.2.3	Not Present		Software Release Notice Dated 7/14/99
TPA Version 3.3	Present	Dated 11/24/99	
TPA Version 4.0	Present	Dated 3/31/2000	

32/131

Shirley

Attached are 2 hard copies and one electronic copy of a code called PVH_YM. I talked with Mark Muller and he said we could either copyright the individual modules or the combined package of all modules, but leaned towards the latter. That is what we have elected to do. I put a cover sheet on each of the hard copies, but not in the electronic file, indicating the code was a package of modules. If this is insufficient let me know.

Larry McKague
10/6/00

PVH_YM Copyright Package

PVH_YM is a software package used to estimate and graphically display the probability of volcanism in the Yucca Mountain Region.

Attached are descriptions of the eight modules that make up the PVH_YM software. These modules are:

- ContourProb.java
- Hestimate.java
- Prob1Graph.java
- Prob2Graph.java
- ProbMap1.java
- ProbMap2.java
- Restimate.java
- volcano.java.

(C) 2000 Southwest Research Institute. All rights reserved

34/131

package graph;

import java.awt.*;
import java.applet.*;
import java.net.URL;
import java.util.*;
import java.io.InputStream;

/*

**
** Class ContourProb
**

by Chuck Connor
Center for Nuclear Waste Regulatory Analyses
Southwest Research Institute
6220 Culebra Rd
San Antonio Texas. 78238-5166, USA
cconnor@swri.edu

Created: December, 1999
(C) 2000 Southwest Research Institute
All rights reserved

Graphics portion of this code modified from code written by Leigh Brookshaw

Purpose: this applet provides contouring capability to the ProbMap classes
that call it. This applet also draws the volcano locations and the
location of the proposed YM repository.

**
** This class extends the interactive graphics class to incorporate
** contours.
**
*****/

public class ContourProb extends G2Dint {

/*

**
** Constants
**
*****/

/*
** The minimum length of a curve before it gets a label
*/

static final int MINCELLS = 30;

/*
** Default number of contour levels
*/

static final int NLEVELS = 12;

/******
**
** Protected Variables
**
*****/

```
/**
 * Dimension of the contour grid in the X direction
 */
protected int nx;

/**
 * Dimension of the contour grid in the Y direction
 */
protected int ny;

/**
 * Vector array containing the Contour curves.
 * Each index in the array contains curves at a given
 * contour level
 */
protected Vector curves[];

/**
 * If set the class calculates the contour levels based on
 * the data minimum and maximum. Default value <i>true</i>.
 */
protected boolean autoLevels;

/*
 * If true the contour levels are calculated in
 * logarithmic intervals
 */
protected boolean logLevels;

/*
 * If true the limits of the plot are the limits of the
 * data grid not the limits of the contours!
 */
protected boolean gridLimits;

/*
 * The array of contour levels
 */
protected double levels[];

/**
 * The label for each contour level
 */
protected TextLine labels[];

/**
 * Font to use in drawing Labels
 */
protected Font labelfont;

/**
 * Color to use in drawing Labels
 */
protected Color labelcolor;

/**
 * Style to use in drawing Labels. TextLine.SCIENTIFIC or
 * TextLine.ALGEBRAIC.
 */
protected int labelStyle;

/**
 * Precision to use in drawing Labels.
 */
protected int labelPrecision;

/**
 * Number of Significant figures to use in drawing Labels.
 */
protected int labelSignificant;

/**
 * Which levels will get labels. If it is equal to 1 every level
 * gets a label, equal to 2 every second level etc. If it is equal to 0
```

```
* no labels are displayed.
*/
protected int labelLevels;
/**
 * If false labels are not drawn
 */
protected boolean drawLabels;
/**
 * If true the labels will be calculated for each
 * contour level. These might not look all that hot.
 */
protected boolean autoLabels;
/**
 * Color to draw non labelled contour line
 */
protected Color contourColor;
/**
 * Color to draw labelled contour line
 */
protected Color labelledColor;

/**
 * The data grid, a 2D array stored in linear form.
 * It is assumed that [0,0] is the bottom left corner
 * and the data is ordered by row.
 */
protected double grid[];

/**
 * The X minimum limit of the data grid
 */
protected double xmin;
/**
 * The X maximum limit of the data grid
 */
protected double xmax;
/**
 * The Y minimum limit of the data grid
 */
protected double ymin;
/**
 * The Y maximum limit of the data grid
 */
protected double ymax;
/**
 * The minimum value of the grid values
 */
protected double zmin;
/**
 * The maximum value of the grid values
 */
protected double zmax;

/**
 * Boolean value if true Contours will not be calculated
 */
public boolean noContours = false;

/*
*****
**
** Constructors
**
*****/
```

```
/**
 * Instantaite the class
 */
public ContourProb() {

    grid = null;
    xmin = 0.0;
    xmax = 0.0;
    ymin = 0.0;
    ymax = 0.0;
    zmin = 0.0;
    zmax = 0.0;

    nx = 0;
    ny = 0;

    levels = new double[NLEVELS];
    labels = new TextLine[NLEVELS];

    autoLevels = true;
    logLevels = false;
    gridLimits = false;
    autoLabels = true;
    labelfont = new Font("Helvetica",Font.PLAIN,12);
    labelcolor = Color.blue;
    labelLevels = 1;
    labelStyle = TextLine.ALGEBRAIC;
    labelPrecision = 2;
    labelSignificant = 3;
    drawlabels = true;

    contourColor = null;
    labelledColor = null;

    curves = null;

}

/*
*****
**
** Methods
**
*****/
/**
 * Load the grid to contour from a URL. There are 2 formats for the data
 * optionally the limits of the grid can be parsed.<BR>
 * <PRE>
 * The expected format of the data
 *      1st Number:  nx
 *      2nd Number:  ny
 *      nx*ny numbers following
 *
 * Optionally
 *      1st Number:  nx
 *      2nd Number:  ny
 *      3rd Number:  xmin
 *      4th Number:  xmax
 *      5th Number:  ymin
 *      6th Number:  ymax
 *      nx*ny numbers following
 * </PRE><BR>
 * If xmin, xmax, ymin, ymax are not specified they are assumed
```

```
* to be [1.0,nx,1.0,ny]
*
* @param file URL of the file to load
* @return <i>true</I> of the load was successful.
*
*/
public boolean loadGrid( URL file) {
    byte b[] = new byte[50];
    int nbytes = 0;
    int max = 100;
    int inc = 100;
    int n = 0;
    double data[] = new double[max];
    InputStream is = null;
    boolean comment = false;
    int c;

    try {
        is = file.openStream();

        while( (c=is.read()) > -1 ) {

            switch (c) {

                case '#':
                    comment = true;
                    break;
                case '\r': case '\n':
                    comment = false;
                case ' ': case '\t':
                    if( nbytes > 0 ) {
                        String s = new String(b,0,0,nbytes);
                        data[n] = Double.valueOf(s).doubleValue();
                        n++;
                        if( n >= max ) {
                            max += inc;
                            double d[] = new double[max];
                            System.arraycopy(data, 0, d, 0, n);
                            data = d;
                        }

                        nbytes = 0;
                    }
                    break;
                default:
                    if( !comment ) {
                        b[nbytes] = (byte)c;
                        nbytes++;
                    }
                    break;
            }

        }

        if (is != null) is.close();
    } catch(Exception e) {
        System.out.println("Failed to load Grid from file ");
        e.printStackTrace();
        if (is != null) try { is.close(); } catch (Exception ev) { }
        return false;
    }

    if(n < 1 ) {
        System.out.println("Failed to load Grid from file ");
    }
}
```

```
        return false;
    }

    nx = (int)(data[0] + 0.5);
    ny = (int)(data[1] + 0.5);

    if( n == nx*ny+6 ) {
        xmin = data[2];
        xmax = data[3];
        ymin = data[4];
        ymax = data[5];

        grid = new double[nx*ny];
        System.arraycopy(data, 6, grid, 0, nx*ny);
    } else
    if( n == nx*ny+2 ) {
        xmin = 1.0;
        xmax = (double)(nx);
        ymin = 1.0;
        ymax = (double)(ny);

        grid = new double[nx*ny];
        System.arraycopy(data, 2, grid, 0, nx*ny);
    } else {
        System.out.println("Error loading grid, Wrong number of points ");
        grid = null;
        return false;
    }

    zrange();
    calcLevels();

    detachCurves();
    curves = null;

    return true;
}

/**
 * Set the range of the grid
 * @param xmin Minimum X value
 * @param xmax Maximum X value
 * @param ymin Minimum Y value
 * @param ymax Maximum Y value
 */

public void setRange(double xmin,double xmax,double ymin,double ymax) {

    if( xmin >= xmax || ymin >= ymax ) return;

    this.xmin = xmin;
    this.xmax = xmax;
    this.ymin = ymin;
    this.ymax = ymax;
}

/**
 * Return the range of the grid
 * @return An array contining xmin,xmax,ymin,ymax.
 */
public double[] getRange() {
    double d[] = new double[4];
```

```
        d[0] = xmin;
        d[1] = xmax;
        d[2] = ymin;
        d[3] = ymax;

        return d;
    }
/**
 * return the dimensions of the grid
 * @return An array containing the number of columns, number of rows.
 */
    public int[] getDim() {
        int i[] = new int[2];

        i[0] = nx;
        i[1] = ny;

        return i;
    }
/**
 * Return the grid
 * @return An array of size nx by ny contining the data grid.
 */
    public double[] getGrid() { return grid; }
/**
 * Manually set the contour levels.
 * @param levels An array containing the contour levels
 * @param nl The number of contour levels in the array
 */
    public void setLevels(double levels[], int nl) {
        int i;
        if( levels == null || nl <= 0 ) return;

        detachCurves();
        curves = null;

        autoLevels = false;

        this.levels = new double[nl];

        System.arraycopy(levels, 0, this.levels, 0, nl);

        labels = new TextLine[nl];
        for(i=0; i<labels.length; i++) {
            labels[i] = new TextLine( String.valueOf( (float)levels[i] ) );
        }
    }
/**
 * Manually set the Contour labels.
 * @param labels An array containing the labels.
 * @param nl Number of labels in the Array.
 */
    public void setLabels(TextLine labels[], int nl) {
        if( labels == null || nl <= 0 ) return;

        autoLabels = false;
        this.labels = new TextLine[nl];

        System.arraycopy(labels, 0, this.labels, 0, nl);
    }
/**
```

```
* Set the font to be used with All the labels
* @param f Font
*/
public void setLabelFont(Font f) {
    labelfont = f;
}

/**
 * Set the Color to be used with all the labels.
 * @param c Color
 */
public void setLabelColor(Color c) {
    labelcolor = c;
}

/**
 * Set the grid to be contoured.
 * @param grid Array of values
 * @param nx Number of columns
 * @param ny Number of rows
 */
public void setGrid(double grid[],int nx,int ny) {
    this.grid = grid;
    this.nx = nx;
    this.ny = ny;

    zrange();
    calcLevels();
}

/**
 * Delete all the Contours
 */
public void deleteContours() {
    if(curves == null) return;
    detachCurves();
    curves = null;
}

/**
 * Detach contours so that they will not be plotted.
 */
public void detachContours() {
    if(curves == null) return;
    detachCurves();
}

/**
 * Attach contours so that they will be plotted.
 */
public void attachContours() {
    if(curves == null) return;
    attachCurves();
}

/**
 * Set the contour's color.
 * @param c Color
 */
public void setContourColor(Color c) { contourColor = c; }

/**
 * Set the labelled contour's color.
 * @param c Color
 */
public void setLabelledContourColor(Color c) { labelledColor = c; }
```

```
/**
 * Return the contour levels.
 * @return An array containing the contour levels
 */
public double[] getLevels() { return levels; }
/**
 * If true the limits of the plot will be the grid limits.
 * If false the limits of the plot will be the contours.
 * @param b boolean
 */
public void setLimitsToGrid(boolean b) { gridLimits = b; }
/**
 * Set the contour levels that are to have labels.
 * <pre>
 *   if 0 no labels are drawn
 *   if 1 every level gets a label
 *   if 2 every 2nd level gets a label
 *   etc.
 * </pre>
 */
public void setLabelLevels(int i) {
    if(i<=0) labelLevels = 0;
    else    labelLevels = i;
}

/**
 * If true contour levels are calculated on a log scale.
 * @param b boolean
 */
public void setLogLevels(boolean b) {
    logLevels = b;

    if( zmin <= 0.0 || zmax <= 0.0 ) logLevels = false;
}

/**
 * Set the number of contour levels.
 * @@param l Number of contour levels
 */
public void setNLevels(int l) {
    if(l <= 0) return;

    levels = new double[l];

    calcLevels();

    detachCurves();
    curves = null;
}

/**
 * If true contour levels are calculated automatically.
 * @param b boolean
 */
public void setAutoLevels(boolean b) {
    autoLevels = b;
}

/**
 * If true contour levels are not labeled.
 * @param b boolean
 */
public void setDrawLabels(boolean b) {
    drawLabels = b;
}
```

}

```
/**
 * Set the label style, either TextLine.SCIENTIFIC or
 * TextLine.ALGEBRAIC.
 * @param s Style
 */
public void setLabelStyle(int s) {
    labelStyle = s;
    calcLabels();
}

/**
 * Get the label style, either TextLine.SCIENTIFIC or
 * TextLine.ALGEBRAIC.
 * @return style
 */
public int getLabelStyle() { return labelStyle; }

/**
 * Set the label precision.
 * @param s Precision
 */
public void setLabelPrecision(int p) {
    labelPrecision = p;
    calcLabels();
}

/**
 * Get the label precision.
 * @return precision
 */
public int getLabelPrecision() { return labelPrecision; }

/**
 * Set the label significant figures.
 * @param s number of significant figures
 */
public void setLabelSignificance(int s) {
    labelSignificant = s;
    calcLabels();
}

/**
 * Get the number of significant figures for labels.
 * @return number of significant figures
 */
public int getLabelSignificance() { return labelSignificant; }

/**
 * Add extra events to the G2Dint event handler.
 *
 * If '1' is pressed repaint without the labels
 * If 'L' is pressed repaint with the labels.
 */
public boolean keyDown(Event e, int key) {
    if(xaxis==null || yaxis==null) return false;

    if( super.keyDown(e,key) ) return true;

    switch ( key ) {

        case '1':
```

```
        drawlabels = false;
        repaint();
        return true;
    case 'L':
        drawlabels = true;
        repaint();
        return true;
    }

    return false;
}

/*
*****
**
** Private Methods
**
*****/

/*
** calcLevels()
**     Calculate the contour levels
*/
private void calcLevels() {
    int i;
    int l;

    if(!autoLevels) return;

    if(levels == null) levels = new double[NLEVELS];
    labels = new TextLine[levels.length];
    // Nice label steps not implemented yet
    //levelStep();

    if( logLevels ) {
        double inc = Math.log(zmax-zmin)/
            (double)(levels.length+1);
        try {
            for(i=0; i<levels.length; i++) levels[i] = zmin +
                Math.pow(Math.E, (double)(i+1)*inc);
        } catch (Exception e) {
            System.out.println("Error calculateing Log levels!");
            System.out.println("... calculating linear levels instead");
            logLevels = false;
            calcLevels();
        }
    } else {
        double inc = (zmax-zmin)/(double)(levels.length+1);
        for(i=0; i<levels.length; i++) levels[i] = zmin + (double)(i+1)*inc;
    }
}

/*
** calcLabels()
**     Calculate the labels
*/
private void calcLabels() {
    int i;
    if( !autoLabels ) return;

    if(levels==null || levels.length <= 0) return;

    labels = new TextLine[levels.length];
```

44/1/31

```
    for(i=0; i<labels.length; i++) {
        labels[i] = new TextLine();
        labels[i].parseDouble(levels[i],
            labelSignificant, labelPrecision, labelStyle);
    }
}

/*
** zrange()
**     Calculate the range of the grid
** */

private void zrange() {
    int i;

    zmin = grid[0];
    zmax = grid[1];
    for( i=0; i<grid.length; i++) {

        zmin = Math.min(zmin, grid[i]);
        zmax = Math.max(zmax, grid[i]);

    }

    System.out.println("Data range: zmin="+zmin+", zmax="+zmax);

    if(zmin == zmax) {
        System.out.println("Cannot produce contours of a constant surface!");
    }

    if(zmin <= 0 || zmax <= 0) logLevels = false;

}

/*
** paintFirst(Graphics g, Rectangle r)
**     before anything is painted calculate the contours.
** */

public void paintFirst(Graphics g, Rectangle r) {

    //System.out.println("paintFirst called");

    if( curves == null && !noContours ) {
        calculateCurves();
        calcLabels();
    }

    setContourColors();

    if(gridLimits && !userlimits ) {
        if( xaxis != null ) {
            if(xaxis.minimum > xmin ) xaxis.minimum = xmin;
            if(xaxis.maximum < xmax ) xaxis.maximum = xmax;
        }

        if( yaxis != null ) {
            if(yaxis.minimum > ymin ) yaxis.minimum = ymin;
            if(yaxis.maximum < ymax ) yaxis.maximum = ymax;
        }
    } else
}
```

45 / 131

```
    if( dataset.isEmpty() ) {
        if( xaxis != null ) {
            xaxis.minimum = xmin;
            xaxis.maximum = xmax;
        }

        if( yaxis != null ) {
            yaxis.minimum = ymin;
            yaxis.maximum = ymax;
        }
    }
}
```

/**

* Set the colors for the contour lines

```
*/
private void setContourColors() {
    int i;
    int j;
    Vector v;

    if( curves == null ||
        (contourColor == null && labelledColor == null) ) return;

    for( i=0; i < curves.length; i++ ) {
        setContourColors( curves[i], null );
    }

    if( contourColor != null ) {
        for( i=0; i < curves.length; i++ ) {
            setContourColors( curves[i], contourColor );
        }
    }

    if( labelledColor != null ) {
        for( i=0; i < curves.length; i++ ) {
            if( i % labelLevels == 0 ) {
                setContourColors( curves[i], labelledColor );
            }
        }
    }
}
```

/**

* Set the colors for the contour lines

```
*/
private void setContourColors( Vector v, Color c ) {
    int i;
    DataSet d;

    if( v == null ) return;

    for( i=0; i < v.size(); i++ ) {
        d = (DataSet) v.elementAt( i );
        if( d != null ) d.linecolor = c;
    }
}
```

/*

```
**      attachCurves()
**          Attach all the curves to the graph and to the axes
*/
private void attachCurves() {
    int i;
    if(curves == null) return;

    for(i=0; i<curves.length; i++) attachCurves(curves[i]);
}

/*
**      attachCurves(Vector v)
**          Attach all the curves from a given level to the graph and to the axes
*/

private void attachCurves(Vector v) {
    int j;
    if(v == null) return;
    for(j=0; j<v.size(); j++) {
        attachDataSet((DataSet)(v.elementAt(j)));
        if(xaxis != null)
            xaxis.attachDataSet((DataSet)(v.elementAt(j)));
        if(yaxis != null)
            yaxis.attachDataSet((DataSet)(v.elementAt(j)));
    }
}

/*
**      detachCurves()
**          Detach All the curves from the graph and the axes.
*/

private void detachCurves() {
    int i;
    if(curves == null) return;

    for(i=0; i<curves.length; i++) detachCurves(curves[i]);
}

/*
**      detachCurves()
**          Detach all the curves from a given level from
**          the graph and the axes.
*/

private void detachCurves(Vector v) {
    int j;
    if(v == null) return;
    for(j=0; j<v.size(); j++) {
        detachDataSet((DataSet)(v.elementAt(j)));
        if(xaxis != null)
            xaxis.detachDataSet((DataSet)(v.elementAt(j)));
        if(yaxis != null)
            yaxis.detachDataSet((DataSet)(v.elementAt(j)));
    }
}

/*
**      paintLast(Graphics g, Rectangle rect)
**          Last thing to be done is to draw the contour labels if required.
*/

public void paintLast(Graphics g, Rectangle rect) {
    int i, j;
    int points;
    int index;
    Vector v;
    DataSet ds;
    double point[] = new double[2];
```

```
int repoxmax, repoymax;
int x;
int y;
Color current = g.getColor();
Rectangle r = new Rectangle();
Rectangle vmark = new Rectangle();
Rectangle repomark = new Rectangle();
Color volcolor = Color.black;
Color repocolor = Color.yellow;
Volcano volcano = new Volcano();

// draw the repository outline
g.setColor(repocolor);
repomark.x = xaxis.getInteger(547198.0);
repoxmax = xaxis.getInteger(549655.0);
repomark.y = yaxis.getInteger(4080229.0);
repoymax = yaxis.getInteger(4076079.0);

repomark.width = repoxmax - repomark.x;
repomark.height = repoymax - repomark.y;

g.fillRect(repomark.x, repomark.y, repomark.width, repomark.height);

// draw the volcano locations
g.setColor(volcolor);
vmark.width = 5;
vmark.height = 5;
volcano.getVolcanoes();
for (i=0; i < Volcano.nvolcanoes*2; i+=2) {
    x = xaxis.getInteger(Volcano.vs[i] );
    y = yaxis.getInteger(Volcano.vs[i+1]);
    vmark.x = x - vmark.width/2;
    vmark.y = y - vmark.height/2;
    g.fillRect(vmark.x, vmark.y, vmark.width, vmark.height);
}

if( xaxis == null || yaxis == null || labels == null ||
    labelLevels == 0 || !drawlabels || curves == null ) {
    super.paintLast(g,rect);
    return;
}

for(i=0; i<levels.length; i++) {
    if( labels[i] != null && !labels[i].isNull() &&
        i%labelLevels == 0 ) {
        labels[i].setFont(labelfont);
        labels[i].setColor(labelcolor);
        v = curves[i];
        for(j=0; j<v.size(); j++) {
            ds = (DataSet)(v.elementAt(j));
            points = ds.dataPoints();
            index = (int)(Math.random()*(double)MINCELLS);
            while ( points > MINCELLS ) {
                point = ds.getPoint(index);
                x = xaxis.getInteger(point[0]);
                y = yaxis.getInteger(point[1]);

                r.width = labels[i].getWidth(g);
                r.height = labels[i].getAscent(g);
                r.x = x - r.width/2;
                r.y = y - r.height/2;
            }
        }
    }
}
```

```
+ );  
  
        //      System.out.println("X = "+point[0] + " Y = " + point[1]  
  
        g.setColor(DataBackground);  
        g.fillRect(r.x, r.y, r.width, r.height);  
  
        g.setColor(current);  
  
        labels[i].draw(g, r.x, r.y+r.height,  
                      TextLine.LEFT);  
  
        points -= MINCELLS;  
        index += MINCELLS;  
    }  
} }  
}  
  
        super.paintLast(g,rect);  
  
    }  
  
/*  
** calculateCurves()  
** Calculate the contours and attach them to the graph and axes.  
*/  
  
protected void calculateCurves() {  
    int i;  
    int j;  
    double data[];  
    double xscale = (xmax-xmin)/(double)(nx-1);  
    double yscale = (ymax-ymin)/(double)(ny-1);  
  
    IsoCurve isocurve;  
  
    isocurve = new IsoCurve(grid,nx,ny);  
  
    if( curves != null) {  
        detachCurves();  
        curves = null;  
    }  
    if( zmin == zmax ) return;  
  
    curves = new Vector(levels.length);  
  
    for(i=0; i<levels.length; i++) {  
        System.out.println("Calculating Contours: level="+levels[i]);  
        isocurve.setValue(levels[i]);  
  
        curves[i] = new Vector();  
  
        while( (data = isocurve.getCurve()) != null ) {  
            for(j=0; j<data.length; ) {  
                data[j] = xmin + data[j]*xscale;  
                j++;  
                data[j] = ymin + data[j]*yscale;  
                j++;  
            }  
  
            try {
```

49
/131

```
        curves[i].addElement(new DataSet(data, data.length/2));
    } catch (Exception e) {
        System.out.println("Error loading contour into DataSet!");
        System.out.println("...Contour Level "+levels[i]);
    }
}

attachCurves(curves[i]);

//repaint();
}
```

}

}

```
/* (C) 2000 Southwest Research Institute
All rights reserved */
```

```

import java.io.*;
import java.math.*;
/*
*****
**
**   Class Hestimate
**
*****
**
by Chuck Connor
Center for Nuclear Waste Regulatory Analyses
Southwest Research Institute
6220 Culebra Rd
San Antonio Texas. 78238-5166, USA
cconnor@swri.edu

Created: September, 1999
      (C) 2000 Southwest Research Institute
      All rights reserved

Purpose:
*Class to estimate the spatial recurrence rate
* aka the spatial weighting factor or spatial
* intensity, for a given volcano distribution
* originally written in TrueBasic by Chuck Connor (1993)
* translated to Java by Chuck Connor (1999)
*
* Contact
* Chuck Connor
* CNWRA, Southwest Research Inst.
* 6220 Culebra Rd
* San Antonio, Tx, 78238-5166, USA
* e-mail: cconnor@swri.org
*
* For more details about the algorithm used in this
* class, see Connor and Hill, 1995, Journal of
* Geophysical Research, 100 (B6) 10,107-10,125.
*****
**/

public class Hestimate {
    public double repox, repoy, nvolcanoes;

    /** where repox and repoy are the coordinates of the
        point for which the calculation is done (assumed to be
        in Universal Transverse Mercator coordinates / meters),
        nvolcanoes is the number of volcanoes
    **/

    public Hestimate(double repox, double repoy, int nvolcanoes) {

        this.repox = repox;
        this.repoy = repoy;
        this.nvolcanoes = nvolcanoes;

    }

    public double gaussh (double []vs, double smoothing_factor) {

        /** This method calculates the value of lambda
            for a given smoothing factor (h) using a gaussian
            kernel function.

```

Pass this method the single dimensional array *s*, containing the point locations of the volcanoes in Universal Transverse Mercator coordinates (meters). The format of array *s* is: *x1,y1,x2,y2,x3,y3,...,xn,yn*. Also pass the value of the smoothing factor, also assumed to have units of meters.

The method returns the value of lambda in units
 ***** 1/ km² *****
 about the point *repx,repoy*, for the given data set, and smoothing factor (*h*).

```
*/
```

```
// let h be the smoothing factor in km
```

```
double h = smoothing_factor;
double dist1, dist2;
double kuu, sumn;
double lambda;
```

```
int i;
```

```
// make sure the summation is now zero
sumn = 0.0;
```

```
double xcenter = repox/1000.0;
double ycenter = repoy/1000.0;
```

```
for (i = 0; i < 2*nvolcanoes; i+=2) {
```

```
    // for each volcano calculate distance2 to repox,repoy (now converted to x,ycent
```

```
    + er)
    dist1 = (xcenter - vs[i]/1000.0)* (xcenter - vs[i]/1000.0) + (ycenter- vs[i+1]/10
    + 00.0)* (ycenter - vs[i+1]/1000.0);
    dist2 = dist1/(h*h);
```

```
    // use the Gaussian kernel
    kuu = 1/(2*Math.PI) * Math.exp(-0.5*dist2);
    sumn += (1.0/(h*h))* kuu;
    }
```

```
//estimate lambda in 1/km2
lambda = sumn/(double) nvolcanoes;
```

```
return lambda;
```

```
} // end of the method gaussh
```

```
public double epanh (double []vs, double smoothing_factor) {
```

```
    /** This method calculates the value of lambda
    for a given smoothing factor (h) using an
    Epanechnikov kernel function.
```

Pass this method the single dimensional array *s*, containing the point locations of the volcanoes in Universal Transverse Mercator coordinates (meters). The format of array *s* is: *x1,y1,x2,y2,x3,y3,...,xn,yn*. Also pass the value of the smoothing factor, also assumed to have units of meters.

The method returns the value of lambda in units
 ***** 1/ km² *****
 about the point *repx,repoy*, for the given data set, and smoothing factor (*h*).

```
/**
```

```
// let h be the smoothing factor in km
```

```
double h = smoothing_factor;  
double dist1, dist2;  
double kuu, sumn;  
double lambda;
```

```
int i;
```

```
// make sure the summation is now zero  
sumn = 0.0;
```

```
double xcenter = repox/1000.0;  
double ycenter = repoy/1000.0;
```

```
for (i = 0; i < 2*nvolcanoes; i+=2) {
```

```
    // for each volcano calculate distance^2 to repox,repoy (now converted to x,ycent
```

```
+ er)
```

```
    dist1 = (xcenter - vs[i]/1000.0)* (xcenter - vs[i]/1000.0) + (ycenter- vs[i+1]/10
```

```
+ 00.0)* (ycenter - vs[i+1]/1000.0);
```

```
    dist2 = dist1/(h*h);
```

```
    // use the Epanechnikov kernel
```

```
    if (dist2 < 1.0) {
```

```
        kuu = (2.0/Math.PI)*(1.0 - dist2);
```

```
        sumn += (1.0/(h*h))* kuu;
```

```
    }
```

```
}
```

```
//estimate lambda in 1/km^2
```

```
lambda = sumn/(double) nvolcanoes;
```

```
return lambda;
```

```
} // end of the method epanh
```

```
}
```

```
/* (C) 2000 Southwest Research Institute  
All rights reserved */
```

53/131

/**

Program Name: PVHA_YM

Class Name: Prob1Graph

Date: 02/11/00

Release Version: 1.0

Client Name: USNRC

U. S. Nuclear Regulatory Commission
NRC Office of Nuclear Material Safety and Safeguard
Division of Waste Management

NRC Contract: NRC 02-97-009

NRC Contact: Dr. John Trapp (301) 415-8063

CNWRA Contact: Dr. Charles Connor (210) 522-6649

Center For Nuclear Waste Regulatory Analyses
Southwest Research Institute
6220 Culebra Rd.
San Antonio, TX, 78238-5166, USA
cconnor@swri.edu

Documentation:
PVHA_YM version 1.0 - Probabilistic Volcanic
Hazard Assessment Methods for a Proposed
High-Level Radioactive Waste Repository
at Yucca Mountain, Nevada

CNWRA 2000-02

NUREG-Series Designator: N/A

DISCLAIMER

"This computer code / material was prepared as an account of work performed by the Center for Nuclear Waste Regulatory Analyses (CNWRA) for the Division of Waste Management of the Nuclear Regulatory Commission (NRC), an independent agency of the United States Government. Neither the developer(s) of the code nor any of their sponsors make any warranty, expressed or implied, or assume any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represent that its use would not infringe on privately-owned rights."

"In no event unless required by applicable law will the sponsors or those who have written or modified this code, be liable for damages, including any loss of profits, lost monies, or other special, incidental or consequential damages arising out of the use or inability to use the program (including but not limited to loss of data or data rendered inaccurate or losses sustained by third parties for failure of the program

54 / 131

to operate with other programs), even if you have been advised of the possibility of such damages or for any claim by any other party."

by C. Connor

*****/

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

```
import graph.*;
import graph.Volcano;
```

/*

```
**
**           Applet Prob1Graph
**
```

Based on code originally written in TrueBasic by C Connor, 1992-1994
Graphics portion of this code modified from code written by Leigh Brookshaw

(C) 2000 Southwest Research Institute
All rights reserved

Purpose:

```
**
** This program calculates the probability of volcanic eruptions
** in a given area
** given input parameters using an Gaussian kernel.
*****
```

```
* This applet uses the Hestimate class
* and plots the result.
```

*****/

```
class HminHmax extends Exception {}
class OddArea extends Exception {}
class OddTime extends Exception {}
class OddRecur extends Exception {}
class OddData extends Exception {}
```

```
public class Prob1Graph extends Applet {
```

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

```
    TextField repoxinput  = new TextField(10); // repository xlocation
    TextField repoyinput  = new TextField(10); // repository y location
    TextField hmininput   = new TextField(10); // min h input
    TextField hmaxinput   = new TextField(10); // max h input
    TextField areainput   = new TextField(10); // effective area input
    TextField timeinput   = new TextField(10); // time interval input
    TextField recurinput  = new TextField(10); // recurrence rate input
    TextField vdatainput  = new TextField(2); // data set to plot
```

```
Button plot = new Button("Calculate"); // Button to plot it.
```

```
public void init() {

    // define the title with a time date stamp and version ###
    Date d = new Date();
    int realmonth = d.getMonth() + 1;
    int realyear = d.getYear() + 1900;
    int realmin = d.getMinutes();
    String titlestring;

    if (realmin < 10) {
        titlestring = "Variation in Probability with H (Gaussian)          "
            + realmonth + "/" + d.getDate() + "/" + realyear
            + ", " + d.getHours() + ":" + "0" + realmin
            + "          PVHA_YM version 1.0";
    }
    else {titlestring = "Variation in Probability with H (Gaussian)          "
        + realmonth + "/" + d.getDate() + "/" + realyear
        + ", " + d.getHours() + ":" + realmin
        + "          PVHA_YM version 1.0";
    }

    Label title = new Label(titlestring, Label.CENTER);

    Panel panel = new Panel();
    GridBagLayout gridbag = new GridBagLayout();
    GridBagConstraints c = new GridBagConstraints();
    Font font = new Font("Helvetica", Font.PLAIN, 14);

    title.setFont(new Font("Helvetica", Font.PLAIN, 14));

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

    repoxinput.setText(getParameter("XLOC"));
    repoyinput.setText(getParameter("YLOC"));
    hmininput.setText(getParameter("HMIN"));
    hmaxinput.setText(getParameter("HMAX"));
    areainput.setText(getParameter("AREA"));
    timeinput.setText(getParameter("TIME"));
    recurinput.setText(getParameter("RECUR"));
    vdatainput.setText(getParameter("DATASET"));

    panel.setLayout(gridbag);

    Label repoxlabel = new Label("Easting");
    Label repoylabel = new Label("Northing");
    Label hminlabel = new Label("hmin");
    Label hmaxlabel = new Label("hmax");
    Label arealabel = new Label("Effective Area");
    Label timelabel = new Label("Time Interval");
    Label recurlabel = new Label("Recurrence Rate");
    Label vdatalabel = new Label("Data Set (1-5)");

    repoxlabel.setFont(font);
    repoylabel.setFont(font);
    hminlabel.setFont(font);
    hmaxlabel.setFont(font);
    arealabel.setFont(font);
```

```
timelabel.setFont(font);
recurlabel.setFont(font);
vdatalabel.setFont(font);
repoxinput.setFont(font);
repoxinput.setBackground(Color.lightGray);
repyinput.setFont(font);
repyinput.setBackground(Color.lightGray);
hmininput.setFont(font);
hmininput.setBackground(Color.lightGray);
hmaxinput.setFont(font);
hmaxinput.setBackground(Color.lightGray);
areainput.setFont(font);
areainput.setBackground(Color.lightGray);
timeinput.setFont(font);
timeinput.setBackground(Color.lightGray);
recurinput.setFont(font);
recurinput.setBackground(Color.lightGray);
vdatainput.setFont(font);
vdatainput.setBackground(Color.lightGray);
plot.setFont(font);
plot.setBackground(Color.cyan);

/*
** First row of the GridBag contains the plot
*/
c.weightx = 1.0;
c.weighty = 1.0;
c.gridwidth = 4;
c.gridwidth=GridBagConstraints.REMAINDER;
c.fill = GridBagConstraints.BOTH;

gridbag.setConstraints(graph,c);

/*
** Second row of the gridBag contains the function input and the
** plotit button
*/
c.fill = GridBagConstraints.NONE;
c.weightx=0.0;
c.weighty=0.0;
c.gridheight=1;

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

c.anchor = GridBagConstraints.WEST;
gridbag.setConstraints(vdatainput,c);

c.fill = GridBagConstraints.WEST;
c.gridwidth=GridBagConstraints.REMAINDER;

gridbag.setConstraints(plot,c);

/*
** Third row contains the
*/
c.gridwidth=1;
c.anchor = GridBagConstraints.EAST;
gridbag.setConstraints(repoxlabel,c);

c.anchor = GridBagConstraints.WEST;
gridbag.setConstraints(repoxinput,c);
```

```
c.anchor = GridBagConstraints.EAST;
gridbag.setConstraints(repoylabel,c);

c.anchor = GridBagConstraints.WEST;
c.gridwidth=GridBagConstraints.REMAINDER;
gridbag.setConstraints(repoyinput,c);

/*
** Fourth row contains the
*/
c.gridwidth=1;
c.anchor = GridBagConstraints.EAST;
gridbag.setConstraints(hminlabel,c);

c.anchor = GridBagConstraints.WEST;
gridbag.setConstraints(hmininput,c);

c.anchor = GridBagConstraints.EAST;
gridbag.setConstraints(hmaxlabel,c);

c.anchor = GridBagConstraints.WEST;
c.gridwidth=GridBagConstraints.REMAINDER;
gridbag.setConstraints(hmaxinput,c);

/*
** Fifth row contains the area and time interval of the grid
*/

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

c.anchor = GridBagConstraints.WEST;
gridbag.setConstraints(areainput,c);

c.anchor = GridBagConstraints.EAST;
gridbag.setConstraints(timelabel,c);

c.anchor = GridBagConstraints.WEST;
c.gridwidth=GridBagConstraints.REMAINDER;
gridbag.setConstraints(timeinput,c);

/*
** Sixth row contains the recurrence rate of the grid
*/
c.gridwidth=1;

c.anchor = GridBagConstraints.EAST;
gridbag.setConstraints(recurlabel,c);

c.anchor = GridBagConstraints.WEST;
c.gridwidth=GridBagConstraints.REMAINDER;
gridbag.setConstraints(recurinput,c);

// display the fields
panel.add(graph);
panel.add(vdatalabel);
panel.add(vdatainput);
panel.add(plot);

panel.add(repoxlabel);
panel.add(repoxinput);
panel.add(repoylabel);
```

```
panel.add(repoyinput);
panel.add(hminlabel);
panel.add(hmininput);
panel.add(hmaxlabel);
panel.add(hmaxinput);
panel.add(arealabel);
panel.add(areainput);
panel.add(timelabel);
panel.add(timeinput);
panel.add(recurlabel);
panel.add(recurinput);

//label the graph axes
xaxis = graph.createXAxis();
xaxis.setTitleText("Smoothing Factor (km)");
xaxis.setTitleFont(new Font("Helvetica",Font.PLAIN,18));
xaxis.setLabelFont(new Font("Helvetica",Font.PLAIN,14));

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

data = new DataSet();

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

graph.setDataBackground(new Color(230,240,240));
graph.setBackground(new Color(240,250,250));

plot();
}

void plot() {

double repox, repoy, hmin, hmax, area, timeint, recur;
int kount;
int j;
int setnum;
double d[] = new double [200];
boolean error = false;

try {
    repox = Double.valueOf(repoxinput.getText()).doubleValue();
} catch(Exception e) {
    this.showStatus("Error with x-location");
    System.out.println("x-location error "+e.getMessage());
    return;
}

try {
    repoy = Double.valueOf(repoyinput.getText()).doubleValue();
} catch(Exception e) {
    this.showStatus("Error with y-location!");
    System.out.println("y location error "+e.getMessage());
    return;
}
```

59
/131

```
try {
    setnum = Double.valueOf(vdatainput.getText()).intValue();
} catch(Exception e) {
    this.showStatus("Illegal data set value: enter an integer!");
    System.out.println("Dataset error "+e.getMessage());
    return;
}

// check for illegal data sets
try {
    if (setnum < 1 || setnum > 5) throw (new OddData() );
} catch(Exception e4) {
    this.showStatus("Error: Data Set between 1 and 5");
    System.out.println("Data Set error ");
    return;
}

try {
    hmin = (Double.valueOf(hmininput.getText()).doubleValue());
} catch(Exception e) {
    this.showStatus("Error with min smoothing factor");
    System.out.println("min smoothing factor error "+e.getMessage());
    return;
}
// illegal for hmin to be negative
if (hmin < 0.0) hmin = 0.0;

try {
    hmax = Double.valueOf(hmaxinput.getText()).doubleValue();
} catch(Exception e) {
    this.showStatus("Error with max smoothing factor");
    System.out.println("max smoothing factor error "+e.getMessage());
    return;
}

//illegal for hmax to be less than hmin
try {
    if (hmax <= hmin) throw (new HminHmax() );
} catch(Exception e1) {
    this.showStatus("Error: Make hmax > hmin");
    System.out.println("hmax < hmin error ");
    return;
}

try {
    area = Double.valueOf(areainput.getText()).doubleValue();
} catch(Exception e) {
    this.showStatus("Error with effective area");
    System.out.println("Effective area error "+e.getMessage());
    return;
}

// illegal for area to be negative
try {
    if (area <= 0.0) throw (new OddArea() );
} catch(Exception e2) {
    this.showStatus("Error: Make Effective Area > 0");
    System.out.println("Effective area error ");
    return;
}

try {
    timeint= Double.valueOf(timeinput.getText()).doubleValue();
} catch(Exception e) {
```

```
        this.showStatus("Error with time interval");
        System.out.println(" time interval error "+e.getMessage());
        return;
    }
    // illegal for timeint to be negative
    try {
        if (timeint <= 0.0) throw (new OddTime() );
    } catch(Exception e3) {
        this.showStatus("Error: Make Time Interval > 0");
        System.out.println("Time interval error ");
        return;
    }

    try {
        recur= Double.valueOf(recurinput.getText()).doubleValue();
    } catch(Exception e) {
        this.showStatus("Error with recurrence interval");
        System.out.println(" recurrence interval error "+e.getMessage());
        return;
    }
    // illegal for recur to be negative
    try {
        if (recur <= 0.0) throw (new OddRecur() );
    } catch(Exception e4) {
        this.showStatus("Error: Make Recurrence Rate > 0");
        System.out.println("Recurrence rate error ");
        return;
    }
    //the volcano data set is determined from setnum
    volcano.setVolcanoset(setnum);

    // instantiate the volcanodata set
    volcano.getVolcanoes();

    Hestimate lambda1 = new Hestimate(repox, repoy, Volcano.nvolcanoes);

    j =0;
    int kountmax = 100;

    for (kount=1; kount <= kountmax; kount++, j+=2) {
        try {

            d[j] = hmin + (hmax-hmin)/((double)kountmax) *(double)kount;
            d[j+1] = 1.0 - Math.exp(-area*timeint*recur*lambda1.gaussh(Volcano.vs, d[j]))

            System.out.println("h = " + d[j]);
            System.out.println("lambda = " + d[j+1]);

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

    data.deleteData();

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

```
graph.repaint();
}

public boolean action(Event e, Object a) {
    if(e.target instanceof Button) {
        if( plot.equals(e.target) ) {
            plot();
            return true;
        }
    }

    return false;
}

}

/* (C) 2000 Southwest Research Institute
All rights reserved */
```

62/131

/**

Program Name: PVHA_YM

Class Name: Prob1Graph

Date: 02/11/00

Release Version: 1.0

Client Name: USNRC

U. S. Nuclear Regulatory Commission
NRC Office of Nuclear Material Safety and Safeguard
Division of Waste Management

NRC Contract: NRC 02-97-009

NRC Contact: Dr. John Trapp (301) 415-8063

CNWRA Contact: Dr. Charles Connor (210) 522-6649

Center For Nuclear Waste Regulatory Analyses
Southwest Research Institute
6220 Culebra Rd.
San Antonio, TX, 78238-5166, USA
cconnor@swri.edu

Documentation:
PVHA_YM version 1.0 - Probabilistic Volcanic
Hazard Assessment Methods for a Proposed
High-Level Radioactive Waste Repository
at Yucca Mountain, Nevada

CNWRA 2000-02

NUREG-Series Designator: N/A

DISCLAIMER

"This computer code / material was prepared as an account of work performed by the Center for Nuclear Waste Regulatory Analyses (CNWRA) for the Division of Waste Management of the Nuclear Regulatory Commission (NRC), an independent agency of the United States Government. Neither the developer(s) of the code nor any of their sponsors make any warranty, expressed or implied, or assume any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represent that its use would not infringe on privately-owned rights."

"In no event unless required by applicable law will the sponsors or those who have written or modified this code, be liable for damages, including any loss of profits, lost monies, or other special, incidental or consequential damages arising out of the use or inability to use the program (including but not limited to loss of data or data rendered inaccurate or losses sustained by third parties for failure of the program

to operate with other programs), even if you have been advised of the possibility of such damages or for any claim by any other party."

```
*****
*****
by C. Connor
```

```
*****
*****/
import java.awt.*;
import java.applet.*;
import java.util.Date;

import graph.*;
import graph.Volcano;
/*
**
**           Applet Prob2Graph
**
*****
```

Based on code originally written in TrueBasic by C Connor, 1992-1994
 Graphics portion of this code modified from code written by Leigh Brookshaw

(C) 2000 Southwest Research Institute
 All rights reserved

Purpose:

```
**
** This program calculates the probability of volcanic eruptions
** in a given area
** given input parameters using an Epanechnikov kernel.
*****
*
* This applet uses the Hestimate class
* and plots the result.
*
*****/
```

```
// classes to handle defined exceptions
class HminHmax extends Exception {}
class OddArea extends Exception {}
class OddTime extends Exception {}
class OddRecur extends Exception {}
class OddData extends Exception {}

public class Prob2Graph extends Applet {

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

    TextField repoxinput    = new TextField(10); // repository xlocation
    TextField repoyinput    = new TextField(10); //repository y location
    TextField hmininput     = new TextField(10); // min h input
    TextField hmaxinput     = new TextField(10); // max h input
    TextField areainput     = new TextField(10); // effective area input
    TextField timeinput     = new TextField(10); // time interval input
    TextField recurinput    = new TextField(10); // recurrence rate input
```

```
TextField vdatainput      = new TextField(2);      // data set to plot
Button plot               = new Button("Calculate"); // Button to plot it.

public void init() {

    // define the title with a time date stamp and version ###
    Date d = new Date();
    int realmonth = d.getMonth() + 1;
    int realyear = d.getYear() + 1900;
    int realmin = d.getMinutes();
    String titlestring;

    if (realmin < 10) {
        titlestring = "Variation in Probability with H (Epanechnikov) "
            + realmonth + "/" + d.getDate() + "/" + realyear
            + ", " + d.getHours() + ":" + "0" + realmin
            + "          PVHA_YM version 1.0";
    }
    else {titlestring = "Variation in Probability with H (Epanechnikov) "
        + realmonth + "/" + d.getDate() + "/" + realyear
        + ", " + d.getHours() + ":" + realmin
        + "          PVHA_YM version 1.0";
    }

    Label title = new Label(titlestring, Label.CENTER);

    Panel panel      = new Panel();
    GridBagLayout gridbag = new GridBagLayout();
    GridBagConstraints c = new GridBagConstraints();
    Font font        = new Font("Helvetica",Font.PLAIN,14);

    title.setFont(new Font("Helvetica",Font.PLAIN,14));

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

    repoxinput.setText(getParameter("XLOC"));
    repoyinput.setText(getParameter("YLOC"));
    hmininput.setText(getParameter("HMIN"));
    hmaxinput.setText(getParameter("HMAX"));
    areainput.setText(getParameter("AREA"));
    timeinput.setText(getParameter("TIME"));
    recurinput.setText(getParameter("RECUR"));
    vdatainput.setText(getParameter("DATASET"));

    panel.setLayout(gridbag);

    Label repoxlabel = new Label("Easting");
    Label repoylabel = new Label("Northing");
    Label hminlabel = new Label("hmin");
    Label hmaxlabel = new Label("hmax");
    Label arealabel = new Label("Effective Area");
    Label timelabel = new Label("Time Interval");
    Label recurlabel = new Label("Recurrence Rate");
    Label vdatalabel = new Label("Data Set (1-5)");

    repoxlabel.setFont(font);
    repoylabel.setFont(font);
```

```
hminlabel.setFont(font);
hmaxlabel.setFont(font);
arealabel.setFont(font);
timelabel.setFont(font);
recurlabel.setFont(font);
vdatalabel.setFont(font);
repoxinput.setFont(font);
repoxinput.setBackground(Color.lightGray);
repyinput.setFont(font);
repyinput.setBackground(Color.lightGray);
hmininput.setFont(font);
hmininput.setBackground(Color.lightGray);
hmaxinput.setFont(font);
hmaxinput.setBackground(Color.lightGray);
areainput.setFont(font);
areainput.setBackground(Color.lightGray);
timeinput.setFont(font);
timeinput.setBackground(Color.lightGray);
recurinput.setFont(font);
recurinput.setBackground(Color.lightGray);
vdatainput.setFont(font);
vdatainput.setBackground(Color.lightGray);
plot.setFont(font);
plot.setBackground(Color.cyan);

/*
** First row of the GridBag contains the plot
*/
c.weightx = 1.0;
c.weighty = 1.0;
c.gridwidth = 4;
c.gridwidth=GridBagConstraints.REMAINDER;
c.fill = GridBagConstraints.BOTH;

gridbag.setConstraints(graph,c);

/*
** Second row of the gridBag contains the function input and the
** plotit button
*/
c.fill = GridBagConstraints.NONE;
c.weightx=0.0;
c.weighty=0.0;
c.gridheight=1;

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

c.anchor = GridBagConstraints.WEST;
gridbag.setConstraints(vdatainput,c);

c.fill = GridBagConstraints.WEST;
c.gridwidth=GridBagConstraints.REMAINDER;

gridbag.setConstraints(plot,c);

/*
** Third row contains the
*/
c.gridwidth=1;
c.anchor = GridBagConstraints.EAST;
gridbag.setConstraints(repoxlabel,c);
```

```
c.anchor = GridBagConstraints.WEST;
gridbag.setConstraints(repoxinput,c);

c.anchor = GridBagConstraints.EAST;
gridbag.setConstraints(repoylabel,c);

c.anchor = GridBagConstraints.WEST;
c.gridwidth=GridBagConstraints.REMAINDER;
gridbag.setConstraints(repoyinput,c);

/*
** Fourth row contains the
*/
c.gridwidth=1;
c.anchor = GridBagConstraints.EAST;
gridbag.setConstraints(hminlabel,c);

c.anchor = GridBagConstraints.WEST;
gridbag.setConstraints(hmininput,c);

c.anchor = GridBagConstraints.EAST;
gridbag.setConstraints(hmaxlabel,c);

c.anchor = GridBagConstraints.WEST;
c.gridwidth=GridBagConstraints.REMAINDER;
gridbag.setConstraints(hmaxinput,c);

/*
** Fifth row contains the area and time interval of the grid
*/
c.gridwidth=1;
c.anchor = GridBagConstraints.EAST;
gridbag.setConstraints(arealabel,c);

c.anchor = GridBagConstraints.WEST;
gridbag.setConstraints(areainput,c);

c.anchor = GridBagConstraints.EAST;
gridbag.setConstraints(timelabel,c);

c.anchor = GridBagConstraints.WEST;
c.gridwidth=GridBagConstraints.REMAINDER;
gridbag.setConstraints(timeinput,c);

/*
** Sixth row contains the recurrence rate of the grid
*/
c.gridwidth=1;

c.anchor = GridBagConstraints.EAST;
gridbag.setConstraints(recurlabel,c);

c.anchor = GridBagConstraints.WEST;
c.gridwidth=GridBagConstraints.REMAINDER;
gridbag.setConstraints(recurinput,c);

// display the fields
panel.add(graph);
panel.add(vdatalabel);
panel.add(vdatainput);
panel.add(plot);
```

67 / 131

```
panel.add(repoxlabel);
panel.add(repoxinput);
panel.add(repoylabel);
panel.add(repoyinput);
panel.add(hminlabel);
panel.add(hmininput);
panel.add(hmaxlabel);
panel.add(hmaxinput);
panel.add(arealabel);
panel.add(areainput);
panel.add(timelabel);
panel.add(timeinput);
panel.add(recurlabel);
panel.add(recurinput);

//label the graph axes
xaxis = graph.createXAxis();
xaxis.setTitleText("Smoothing Factor (km)");
xaxis.setTitleFont(new Font("Helvetica",Font.PLAIN,18));
xaxis.setLabelFont(new Font("Helvetica",Font.PLAIN,14));

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

data = new DataSet();

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

graph.setDataBackground(new Color(230,240,240));
graph.setBackground(new Color(240,250,250));

plot();
}

void plot() {

double repox, repoy, hmin, hmax, area, timeint, recur;
int kount;
int j;
int setnum;
double d[] = new double [200];
boolean error = false;

try {
    repox = Double.valueOf(repoxinput.getText()).doubleValue();
} catch(Exception e) {
    this.showStatus("Error with x-location");
    System.out.println("x-location error "+e.getMessage());
    return;
}

try {
    repoy = Double.valueOf(repoyinput.getText()).doubleValue();
} catch(Exception e) {
    this.showStatus("Error with y-location!");
}
```

48/131

```
        System.out.println("y location error "+e.getMessage());
        return;
    }
    try {
        setnum = Double.valueOf(vdatainput.getText()).intValue();
    } catch(Exception e) {
        this.showStatus("Illegal data set value: enter an integer!");
        System.out.println("Dataset error "+e.getMessage());
        return;
    }

    // check for illegal data sets
    try {
        if (setnum < 1 || setnum > 5) throw (new OddData() );
    } catch(Exception e4) {
        this.showStatus("Error: Data Set between 1 and 5");
        System.out.println("Data Set error ");
        return;
    }

    try {
        hmin = (Double.valueOf(hmininput.getText()).doubleValue());
    } catch(Exception e) {
        this.showStatus("Error with min smoothing factor");
        System.out.println("min smoothing factor error "+e.getMessage());
        return;
    }
    // illegal for hmin to be negative
    if (hmin < 0.0) hmin = 0.0;

    try {
        hmax = Double.valueOf(hmaxinput.getText()).doubleValue();
    } catch(Exception e) {
        this.showStatus("Error with max smoothing factor");
        System.out.println("max smoothing factor error "+e.getMessage());
        return;
    }

    //illegal for hmax to be less than hmin
    try {
        if (hmax <= hmin) throw (new HminHmax() );
    } catch(Exception e1) {
        this.showStatus("Error: Make hmax > hmin");
        System.out.println("hmax < hmin error ");
        return;
    }

    try {
        area = Double.valueOf(areainput.getText()).doubleValue();
    } catch(Exception e) {
        this.showStatus("Error with effective area");
        System.out.println("Effective area error "+e.getMessage());
        return;
    }

    // illegal for area to be negative
    try {
        if (area <= 0.0) throw (new OddArea() );
    } catch(Exception e2) {
        this.showStatus("Error: Make Effective Area > 0");
        System.out.println("Effective area error ");
        return;
    }
}
```

```
try {
    timeint= Double.valueOf(timeinput.getText()).doubleValue();
} catch(Exception e) {
    this.showStatus("Error with time interval");
    System.out.println(" time interval error "+e.getMessage());
    return;
}

// illegal for timeint to be negative
try {
    if (timeint <= 0.0) throw (new OddTime() );
} catch(Exception e3) {
    this.showStatus("Error: Make Time Interval > 0");
    System.out.println("Time interval error ");
    return;
}

try {
    recur= Double.valueOf(recurinput.getText()).doubleValue();
} catch(Exception e) {
    this.showStatus("Error with recurrence interval");
    System.out.println(" recurrence interval error "+e.getMessage());
    return;
}

// illegal for recur to be negative
try {
    if (recur <= 0.0) throw (new OddRecur() );
} catch(Exception e4) {
    this.showStatus("Error: Make Recurrence Rate > 0");
    System.out.println("Recurrence rate error ");
    return;
}

//the volcano data set is determined from setnum
volcano.setVolcanoset(setnum);

// instantiate the volcanodata set
volcano.getVolcanoes();

Hestimate lambda1 = new Hestimate(repox, repoy, Volcano.nvolcanoes);

j =0;
int kountmax = 100;

for (kount=1; kount <= kountmax; kount++, j+=2) {
    try {

        d[j] = hmin + (hmax-hmin)/(((double)kountmax) *((double)kount);
        d[j+1] = 1.0 - Math.exp(-area*timeint*recur*lambda1.epanh(Volcano.vs, d[j]));
        System.out.println("h = " + d[j]);
        System.out.println("lambda = " + d[j+1]);

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

data.deleteData();

try {
    data.append(d, kountmax);
} catch(Exception e) {
    this.showStatus("Error while appending data!");
}
```

70 / 131

```
        System.out.println("Error while appending data!");  
        return;  
    }
```

```
graph.repaint();  
}
```

```
public boolean action(Event e, Object a) {
```

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

```
    return false;  
}
```

```
}
```

```
/* (C) 2000 Southwest Research Institute  
All rights reserved */
```

71/131

/**

Program Name: PVHA_YM

Class Name: Prob1Graph

Date: 02/11/00

Release Version: 1.0

Client Name: USNRC

U. S. Nuclear Regulatory Commission
NRC Office of Nuclear Material Safety and Safeguard
Division of Waste Management

NRC Contract: NRC 02-97-009

NRC Contact: Dr. John Trapp (301) 415-8063

CNWRA Contact: Dr. Charles Connor (210) 522-6649

Center For Nuclear Waste Regulatory Analyses
Southwest Research Institute
6220 Culebra Rd.
San Antonio, TX, 78238-5166, USA
cconnor@swri.edu

Documentation:
PVHA_YM version 1.0 - Probabilistic Volcanic
Hazard Assessment Methods for a Proposed
High-Level Radioactive Waste Repository
at Yucca Mountain, Nevada

CNWRA 2000-02

NUREG-Series Designator: N/A

DISCLAIMER

"This computer code / material was prepared as an account of work performed by the Center for Nuclear Waste Regulatory Analyses (CNWRA) for the Division of Waste Management of the Nuclear Regulatory Commission (NRC), an independent agency of the United States Government. Neither the developer(s) of the code nor any of their sponsors make any warranty, expressed or implied, or assume any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represent that its use would not infringe on privately-owned rights."

"In no event unless required by applicable law will the sponsors or those who have written or modified this code, be liable for damages, including any loss of profits, lost monies, or other special, incidental or consequential damages arising out of the use or inability to use the program (including but not limited to loss of data or data rendered inaccurate or losses sustained by third parties for failure of the program

72/131

to operate with other programs), even if you have been advised of the possibility of such damages or for any claim by any other party."

by C. Connor

```
*****  
*****/  
import java.awt.*;  
import java.applet.*;  
import java.util.Date;  
  
import graph.*;  
import graph.ContourProb;  
import graph.Volcano;  
/*  
*****  
**  
**           Applet ProbMap1  
**  
*****
```

Based on code originally written in TrueBasic by C Connor, 1992-1994
Graphics portion of this code modified from code written by Leigh Brookshaw

(C) 2000 Southwest Research Institute
All rights reserved

Purpose:

applet ProbMap1 calculates the conditional probability of a new volcano forming within a 1 km x 1 km region, given that a new volcano forms in the region

This applet uses the Restimate class to estimate probability and contours the results

This applet uses the Volcano class to select data sets

This applet uses the ContourProb class to plot the map

*****/

```
// classes to handle defined exceptions  
class OddSF extends Exception {}  
class OddData extends Exception {}
```

```
public class ProbMap1 extends Applet {  
  
    ContourProb graph      = new ContourProb(); // Graph class to do the plotting  
    Axis xaxis;  
    Axis yaxis;  
    DataSet data;  
    Volcano volcano = new Volcano();  
  
    TextField cinput      = new TextField(4); // Number of contours  
    TextField xinput      = new TextField(4); // X grid dimension  
    TextField yinput      = new TextField(4); // Y grid dimension  
    TextField xmininput   = new TextField(10); // Minimum x value input
```

73/131

```

TextField xmaxinput  = new TextField(10);      // Maximum x value input
TextField ymininput  = new TextField(10);      // Minimum y value input
TextField ymaxinput  = new TextField(10);      // Maximum y value input
TextField vdatainput = new TextField(2);       // data set to plot
TextField sfinput    = new TextField(4);       // smoothing factor input
Button plot          = new Button("Calculate"); // Button to plot it.

```

```
public void init() {
```

```
    // define the title with a time date stamp and version ###
```

```
    Date d = new Date();
```

```
    int realmonth = d.getMonth() + 1;
```

```
    int realyear = d.getYear() + 1900;
```

```
    int realmin = d.getMinutes();
```

```
    String titlestring;
```

```
    if (realmin < 10) {
```

```
        titlestring = "Gaussian Probability Map"
            + realmonth + "/" + d.getDate() + "/" + realyear
            + ", " + d.getHours() + ":" + "0" + realmin
            + "          PVHA_YM version 1.0";
    }
```

```
    else {titlestring = "Gaussian Probability Map"
        + realmonth + "/" + d.getDate() + "/" + realyear
        + ", " + d.getHours() + ":" + realmin
        + "          PVHA_YM version 1.0";
    }
```

```
}
```

```
Label title = new Label(titlestring, Label.CENTER);
```

```
Panel panel = new Panel();
```

```
GridBagLayout gridbag = new GridBagLayout();
```

```
GridBagConstraints c = new GridBagConstraints();
```

```
Font font = new Font("Helvetica", Font.PLAIN, 14);
```

```
title.setFont(new Font("Helvetica", Font.PLAIN, 14));
```

```
setLayout(new BorderLayout());
```

```
add("North", title);
```

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

```
// Data from .html
```

```
vdatainput.setText(getParameter("DATASET"));
```

```
sfinput.setText(getParameter("SMOOTH"));
```

```
cinput.setText(getParameter("CONTOURS"));
```

```
xinput.setText(getParameter("XGRID"));
```

```
yinput.setText(getParameter("YGRID"));
```

```
xmininput.setText(getParameter("XMIN"));
```

```
xmaxinput.setText(getParameter("XMAX"));
```

```
ymininput.setText(getParameter("YMIN"));
```

```
ymaxinput.setText(getParameter("YMAX"));
```

```
panel.setLayout(gridbag);
```

```
/*
```

```
** create labels for all the text input fields
```

```
*/
```

```
Label clabel = new Label("Number of Contours");
```

```
Label xlabel = new Label("X grid dimension");
```

```
Label ylabel = new Label("Y grid dimension");
```

```
Label xminlabel = new Label("Minimum Easting");
```

```
Label xmaxlabel = new Label("Maximum Easting");
```

```
Label yminlabel = new Label("Minimum Northing");
```

```
Label ymaxlabel = new Label("Maximum Northing");
```

```
Label vdatalabel = new Label("Data Set (1-5)");
```

```
Label sflabel    = new Label("Smoothing Factor (km)");
/*
** Set the fonts and colors to use
*/
clabel.setFont(font);
xlabel.setFont(font);
ylabel.setFont(font);
vdatalabel.setFont(font);
sflabel.setFont(font);
xminlabel.setFont(font);
xmaxlabel.setFont(font);
yminlabel.setFont(font);
ymaxlabel.setFont(font);

cinput.setFont(font);
cinput.setBackground(Color.lightGray);
xinput.setFont(font);
xinput.setBackground(Color.lightGray);
yinput.setFont(font);
yinput.setBackground(Color.lightGray);
vdatainput.setFont(font);
sfinput.setBackground(Color.lightGray);
sfinput.setFont(font);
vdatainput.setBackground(Color.lightGray);
xmininput.setFont(font);
xmininput.setBackground(Color.lightGray);
xmaxinput.setFont(font);
xmaxinput.setBackground(Color.lightGray);
ymininput.setFont(font);
ymininput.setBackground(Color.lightGray);
ymaxinput.setFont(font);
ymaxinput.setBackground(Color.lightGray);
plot.setFont(font);
plot.setBackground(Color.cyan);

/*
** First row of the GridBag contains the plot
*/
c.weightx = 1.0;
c.weighty = 1.0;
c.gridwidth = 4;
c.gridwidth=GridBagConstraints.REMAINDER;
c.fill = GridBagConstraints.BOTH;

gridbag.setConstraints(graph,c);

/*
** Second row of the gridBag contains the function input and the
** plotit button
*/
c.fill = GridBagConstraints.NONE;
c.weightx=0.0;
c.weighty=0.0;
c.gridheight=1;

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

c.anchor = GridBagConstraints.WEST;
gridbag.setConstraints(vdatainput,c);
```

```
c.fill = GridBagConstraints.WEST;
c.gridwidth=GridBagConstraints.REMAINDER;

gridbag.setConstraints(plot,c);

/*
** Third row contains the Number of contours and smoothing factor
*/
c.gridwidth=1;
c.anchor = GridBagConstraints.EAST;
gridbag.setConstraints(clabel,c);

c.anchor = GridBagConstraints.WEST;
gridbag.setConstraints(cinput,c);

c.anchor = GridBagConstraints.EAST;
gridbag.setConstraints(sflabel,c);

c.anchor = GridBagConstraints.WEST;
c.gridwidth=GridBagConstraints.REMAINDER;
gridbag.setConstraints(sfinput,c);

/*
** Fourth row contains the grid dimensions
*/
c.gridwidth=1;
c.anchor = GridBagConstraints.EAST;
gridbag.setConstraints(xlabel,c);

c.anchor = GridBagConstraints.WEST;
gridbag.setConstraints(xinput,c);

c.anchor = GridBagConstraints.EAST;
gridbag.setConstraints(ylabel,c);

c.anchor = GridBagConstraints.WEST;
c.gridwidth=GridBagConstraints.REMAINDER;
gridbag.setConstraints(yinput,c);

/*
** Fifth row contains the x range of the grid
*/

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

c.anchor = GridBagConstraints.WEST;
gridbag.setConstraints(xmininput,c);

c.anchor = GridBagConstraints.EAST;
gridbag.setConstraints(xmaxlabel,c);

c.anchor = GridBagConstraints.WEST;
c.gridwidth=GridBagConstraints.REMAINDER;
gridbag.setConstraints(xmaxinput,c);

/*
** Sixth row contains the y range of the grid
*/

c.gridwidth=1;
c.anchor = GridBagConstraints.EAST;
```

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

c.anchor = GridBagConstraints.WEST;
gridbag.setConstraints(ymininput,c);

c.anchor = GridBagConstraints.EAST;
gridbag.setConstraints(ymaxlabel,c);

c.anchor = GridBagConstraints.WEST;
c.gridwidth=GridBagConstraints.REMAINDER;
gridbag.setConstraints(ymaxinput,c);

// display the fields
panel.add(graph);
panel.add(vdatalabel);
panel.add(vdatainput);

panel.add(plot);
panel.add(clabel);
panel.add(cinput);
panel.add(sflabel);
panel.add(sfinput);
panel.add(xlabel);
panel.add(xinput);
panel.add(ylabel);
panel.add(yinput);
panel.add(xminlabel);
panel.add(xmininput);
panel.add(xmaxlabel);
panel.add(xmaxinput);
panel.add(yminlabel);
panel.add(ymininput);
panel.add(ymaxlabel);
panel.add(ymaxinput);

//label the graph axes
xaxis = graph.createXAxis();
xaxis.setTitleText("Easting");

yaxis = graph.createYAxis();
yaxis.setTitleText("Northing");

// set the graph colors
graph.setDataBackground(new Color(230,240,240));
graph.setBackground(new Color(240,250,250));
graph.setContourColor(Color.black);
graph.setLabelledContourColor(Color.red);

// set the label interval
// default condition -> contours not labelled
graph.setLabelLevels(5);
graph.setDrawLabels(false);

// since its a map, make it square
graph.square = true;

plot();

}

// the plot class does the work of creating the array to contour
// and interpreting the parameters from input
```

```
void plot() {
    int levels;
    int nx;
    int ny;
    int i, j;
    int setnum;
    double xmax;
    double xmin;
    double ymax;
    double ymin;
    double x, y;
    int count = 0;
    boolean error = false;
    double smoothing_factor;

    try {
        graph.setNLevels( Integer.parseInt(cinput.getText()) );
    } catch(Exception e) {
        this.showStatus("Error with number of contour levels!");
        System.out.println(
            "Number of contour levels error "+e.getMessage());
        return;
    }

    try {
        nx = Integer.parseInt(xinput.getText());
        ny = Integer.parseInt(yinput.getText());
    } catch(Exception e) {
        this.showStatus("Error parsing grid dimensions!");
        System.out.println(
            "Error parsing grid dimensions! "+e.getMessage());
        return;
    }

    try {
        xmax = Double.valueOf(xmaxinput.getText()).doubleValue();
        xmin = Double.valueOf(xmininput.getText()).doubleValue();
    } catch(Exception e) {
        this.showStatus("Error with X range!");
        System.out.println("X range error "+e.getMessage());
        return;
    }

    try {
        ymax = Double.valueOf(ymaxinput.getText()).doubleValue();
        ymin = Double.valueOf(ymininput.getText()).doubleValue();
    } catch(Exception e) {
        this.showStatus("Error with Y range!");
        System.out.println("Y range error "+e.getMessage());
        return;
    }

    try {
        setnum = Double.valueOf(vdatainput.getText()).intValue();
    } catch(Exception e) {
        this.showStatus("Illegal data set value: enter an integer!");
        System.out.println("Dataset error "+e.getMessage());
        return;
    }

    // check for illegal data sets
    try {
        if (setnum < 1 || setnum > 5) throw (new OddData() );
    }
```

```
    } catch(Exception e4) {
        this.showStatus("Error: Data Set between 1 and 5");
        System.out.println("Data Set error ");
        return;
    }

    try {
        smoothing_factor = Double.valueOf(sfinput.getText()).doubleValue();
    } catch(Exception e) {
        this.showStatus("Illegal smoothing factor");
        System.out.println("smoothing factor error "+e.getMessage());
        return;
    }

    // Smoothing factor must be positive real number
    try {
        if (smoothing_factor <= 0.0) throw (new OddSF() );
    } catch(Exception e3) {
        this.showStatus("Error: Make Smoothing Factor > 0");
        System.out.println("Smoothing Factor error ");
        return;
    }

    if(xmin>=xmax || ymin>= ymax ) {
        this.showStatus("Error with Grid range!");
        System.out.println("grid range error! ");
        return;
    }
    if( nx < 2 || ny < 2 ) {
        this.showStatus("Error with Grid dimensions!");
        System.out.println("grid dimension error!");
        return;
    }

    graph.setRange(xmin,xmax,ymin,ymax);

    //the volcano data set is determined from setnum
    volcano.setVolcanoset(setnum);

    //array is the grid to plot
    double array[] = new double[nx*ny];

    this.showStatus("Calculating Grid Values!");
    count = 0;

    // instantiate the volcanodata set
    volcano.getVolcanoes();

    // instaniate lambda1 - a new instance of Restimate
    Restimate lambda1 = new Restimate(smoothing_factor, Volcano.nvolcanoes);

    // fill the array
    for(j=0; j<ny; j++) {
        y = ymin + j*(ymax-ymin)/(ny-1);
        for(i=0; i<nx; i++) {
            x = xmin + i*(xmax-xmin)/(nx-1);
            try {

                // use the gaussian method in Restimate
                array[count++] = lambda1.gaussh(Volcano.vs, x,y);
            } catch(Exception e) {
                array[count++] = 0.0;
                error = true; }
        }
    }
}
```

79/131

```
    }  
  }  
  
  //instantiate graph with array and array dimensions  
  graph.setGrid(array,nx,ny);  
  
  this.showStatus("Calculating Contours!");  
  
  graph.repaint();  
}  
  
//keep track of Calculate events  
public boolean action(Event e, Object a) {  
    if(e.target instanceof Button) {  
        if( plot.equals(e.target) ) {  
            plot();  
            return true;  
        }  
    }  
    return false;  
}  
}  
  
/* (C) 2000 Southwest Research Institute  
All rights reserved */
```

80/131

/**

Program Name: PVHA_YM

Class Name: Prob1Graph

Date: 02/11/00

Release Version: 1.0

Client Name: USNRC

U. S. Nuclear Regulatory Commission
NRC Office of Nuclear Material Safety and Safeguard
Division of Waste Management

NRC Contract: NRC 02-97-009

NRC Contact: Dr. John Trapp (301) 415-8063

CNWRA Contact: Dr. Charles Connor (210) 522-6649

Center For Nuclear Waste Regulatory Analyses
Southwest Research Institute
6220 Culebra Rd.
San Antonio, TX, 78238-5166, USA
cconnor@swri.edu

Documentation:
PVHA_YM version 1.0 - Probabilistic Volcanic
Hazard Assessment Methods for a Proposed
High-Level Radioactive Waste Repository
at Yucca Mountain, Nevada

CNWRA 2000-02

NUREG-Series Designator: N/A

DISCLAIMER

"This computer code / material was prepared as an account of work performed by the Center for Nuclear Waste Regulatory Analyses (CNWRA) for the Division of Waste Management of the Nuclear Regulatory Commission (NRC), an independent agency of the United States Government. Neither the developer(s) of the code nor any of their sponsors make any warranty, expressed or implied, or assume any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represent that its use would not infringe on privately-owned rights."

"In no event unless required by applicable law will the sponsors or those who have written or modified this code, be liable for damages, including any loss of profits, lost monies, or other special, incidental or consequential damages arising out of the use or inability to use the program (including but not limited to loss of data or data rendered inaccurate or losses sustained by third parties for failure of the program

81/131

to operate with other programs), even if you have been advised of the possibility of such damages or for any claim by any other party."

by C. Connor

```
*****  
*****  
import java.awt.*;  
import java.applet.*;  
import java.util.Date;  
  
import graph.*;  
import graph.ContourProb;  
import graph.Volcano;  
/*  
**  
**           Applet ProbMap2  
**  
*****
```

Based on code originally written in TrueBasic by C Connor, 1992-1994
Graphics portion of this code modified from code written by Leigh Brookshaw

(C) 2000 Southwest Research Institute
All rights reserved

Purpose:

applet ProbMap2 calculates the conditional probability of a new volcano forming within a 1 km x 1 km region, given that a new volcano forms in the region, using an epanechnikov kernel

This applet uses the Restimate class to estimate probability and contours the results

This applet uses the Volcano class to select data sets

This applet uses the ContourProb class to plot the map

*****/

```
// classes to handle defined exceptions  
class OddSF extends Exception {}  
class OddData extends Exception {}
```

```
public class ProbMap2 extends Applet {  
  
    ContourProb graph          = new ContourProb();    // Graph class to do the plotting  
    Axis xaxis;  
    Axis yaxis;  
    DataSet data;  
    Volcano volcano = new Volcano();  
  
    TextField cinput           = new TextField(4);    // Number of contours  
    TextField xinput           = new TextField(4);    // X grid dimension  
    TextField yinput           = new TextField(4);    // Y grid dimension  
    TextField xmininput        = new TextField(10);   // Minimum x value input
```

```

TextField xmaxinput  = new TextField(10);      // Maximum x value input
TextField ymininput  = new TextField(10);      // Minimum y value input
TextField ymaxinput  = new TextField(10);      // Maximum y value input
TextField vdatainput = new TextField(2);       // data set to plot
TextField sfinput    = new TextField(4);       // smoothing factor input
Button plot          = new Button("Calculate"); // Button to plot it.

```

```

public void init() {

    // define the title with a time date stamp and version ###
    Date d = new Date();
    int realmonth = d.getMonth() + 1;
    int realyear  = d.getYear() + 1900;
    int realmin   = d.getMinutes();
    String titlestring;

    if (realmin < 10) {
        titlestring = "Epanechnikov Probability Map          "
            + realmonth + "/" + d.getDate() + "/" + realyear
            + ", " + d.getHours() + ":" + "0" + realmin
            + "          PVHA_YM version 1.0";
    }
    else {titlestring = "Epanechnikov Probability Map          "
        + realmonth + "/" + d.getDate() + "/" + realyear
        + ", " + d.getHours() + ":" + realmin
        + "          PVHA_YM version 1.0";
    }

    Label title = new Label(titlestring, Label.CENTER);

    Panel panel = new Panel();
    GridBagLayout gridbag = new GridBagLayout();
    GridBagConstraints c = new GridBagConstraints();
    Font font = new Font("Helvetica", Font.PLAIN, 14);

    title.setFont(new Font("Helvetica", Font.PLAIN, 14));

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

    // Data from .html
    vdatainput.setText(getParameter("DATASET"));
    sfinput.setText(getParameter("SMOOTH"));
    cinput.setText(getParameter("CONTOURS"));
    xininput.setText(getParameter("XGRID"));
    yininput.setText(getParameter("YGRID"));
    xmininput.setText(getParameter("XMIN"));
    xmaxinput.setText(getParameter("XMAX"));
    ymininput.setText(getParameter("YMIN"));
    ymaxinput.setText(getParameter("YMAX"));

    panel.setLayout(gridbag);
    /*
    ** create labels for all the text input fields
    */
    Label clabel = new Label("Number of Contours");
    Label xlabel = new Label("X grid dimension");
    Label ylabel = new Label("Y grid dimension");
    Label xminlabel = new Label("Minimum Easting");
    Label xmaxlabel = new Label("Maximum Easting");
    Label yminlabel = new Label("Minimum Northing");
    Label ymaxlabel = new Label("Maximum Northing");
    Label vdatalabel = new Label("Data Set (1-5)");

```

```
Label sflabel      = new Label("Smoothing Factor (km)");
/*
** Set the fonts and colors to use
*/
clabel.setFont(font);
xlabel.setFont(font);
ylabel.setFont(font);
vdatalabel.setFont(font);
sflabel.setFont(font);
xminlabel.setFont(font);
xmaxlabel.setFont(font);
yminlabel.setFont(font);
ymaxlabel.setFont(font);

cinput.setFont(font);
cinput.setBackground(Color.lightGray);
xinput.setFont(font);
xinput.setBackground(Color.lightGray);
yinput.setFont(font);
yinput.setBackground(Color.lightGray);
vdatainput.setFont(font);
sfinput.setBackground(Color.lightGray);
sfinput.setFont(font);
vdatainput.setBackground(Color.lightGray);
xmininput.setFont(font);
xmininput.setBackground(Color.lightGray);
xmaxinput.setFont(font);
xmaxinput.setBackground(Color.lightGray);
ymininput.setFont(font);
ymininput.setBackground(Color.lightGray);
ymaxinput.setFont(font);
ymaxinput.setBackground(Color.lightGray);
plot.setFont(font);
plot.setBackground(Color.cyan);

/*
** First row of the GridBag contains the plot
*/
c.weightx = 1.0;
c.weighty = 1.0;
c.gridwidth = 4;
c.gridwidth=GridBagConstraints.REMAINDER;
c.fill = GridBagConstraints.BOTH;

gridbag.setConstraints(graph,c);

/*
** Second row of the gridBag contains the function input and the
** plotit button
*/
c.fill = GridBagConstraints.NONE;
c.weightx=0.0;
c.weighty=0.0;
c.gridheight=1;

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

c.anchor = GridBagConstraints.WEST;
gridbag.setConstraints(vdatainput,c);
```

```
c.fill = GridBagConstraints.WEST;
c.gridwidth=GridBagConstraints.REMAINDER;

gridbag.setConstraints(plot,c);

/*
** Third row contains the Number of contours and smoothinf factor
*/
c.gridwidth=1;
c.anchor = GridBagConstraints.EAST;
gridbag.setConstraints(clabel,c);

c.anchor = GridBagConstraints.WEST;
gridbag.setConstraints(cinput,c);

c.anchor = GridBagConstraints.EAST;
gridbag.setConstraints(sflabel,c);

c.anchor = GridBagConstraints.WEST;
c.gridwidth=GridBagConstraints.REMAINDER;
gridbag.setConstraints(sfinput,c);

/*
** Fourth row contains the grid dimensions
*/
c.gridwidth=1;
c.anchor = GridBagConstraints.EAST;
gridbag.setConstraints(xlabel,c);

c.anchor = GridBagConstraints.WEST;
gridbag.setConstraints(xinput,c);

c.anchor = GridBagConstraints.EAST;
gridbag.setConstraints(ylabel,c);

c.anchor = GridBagConstraints.WEST;
c.gridwidth=GridBagConstraints.REMAINDER;
gridbag.setConstraints(yinput,c);

/*
** Fifth row contains the x range of the grid
*/
c.gridwidth=1;
c.anchor = GridBagConstraints.EAST;
gridbag.setConstraints(xminlabel,c);

c.anchor = GridBagConstraints.WEST;
gridbag.setConstraints(xmininput,c);

c.anchor = GridBagConstraints.EAST;
gridbag.setConstraints(xmaxlabel,c);

c.anchor = GridBagConstraints.WEST;
c.gridwidth=GridBagConstraints.REMAINDER;
gridbag.setConstraints(xmaxinput,c);

/*
** Sixth row contains the y range of the grid
*/
c.gridwidth=1;
c.anchor = GridBagConstraints.EAST;
```

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

c.anchor = GridBagConstraints.WEST;
gridbag.setConstraints(ymininput,c);

c.anchor = GridBagConstraints.EAST;
gridbag.setConstraints(ymaxlabel,c);

c.anchor = GridBagConstraints.WEST;
c.gridwidth=GridBagConstraints.REMAINDER;
gridbag.setConstraints(ymaxinput,c);

// display the fields
panel.add(graph);
panel.add(vdatalabel);
panel.add(vdatainput);

panel.add(plot);
panel.add(clabel);
panel.add(cinput);
panel.add(sflabel);
panel.add(sfinput);
panel.add(xlabel);
panel.add(xinput);
panel.add(ylabel);
panel.add(yinput);
panel.add(xminlabel);
panel.add(xmininput);
panel.add(xmaxlabel);
panel.add(xmaxinput);
panel.add(yminlabel);
panel.add(ymininput);
panel.add(ymaxlabel);
panel.add(ymaxinput);

//label the graph axes
xaxis = graph.createXAxis();
xaxis.setTitleText("Easting");

yaxis = graph.createYAxis();
yaxis.setTitleText("Northing");

// set the graph colors
graph.setDataBackground(new Color(230,240,240));
graph.setBackground(new Color(240,250,250));
graph.setContourColor(Color.black);
graph.setLabelledContourColor(Color.red);

// set the label interval
// default condition -> contours not labelled
graph.setLabelLevels(5);
graph.setDrawLabels(false);

// since its a map, make it square
graph.square = true;

plot();

}

// the plot class does the work of creating the array to contour
// and interpreting the parameters from input
```

```
void plot() {
    int levels;
    int nx;
    int ny;
    int i, j;
    int setnum;
    double xmax;
    double xmin;
    double ymax;
    double ymin;
    double x, y;
    int count = 0;
    boolean error = false;
    double smoothing_factor;

    try {
        graph.setNLevels( Integer.parseInt(cinput.getText()) );
    } catch(Exception e) {
        this.showStatus("Error with number of contour levels!");
        System.out.println(
            "Number of contour levels error "+e.getMessage());
        return;
    }

    try {
        nx = Integer.parseInt(xinput.getText());
        ny = Integer.parseInt(yinput.getText());
    } catch(Exception e) {
        this.showStatus("Error parsing grid dimensions!");
        System.out.println(
            "Error parsing grid dimensions! "+e.getMessage());
        return;
    }

    try {
        xmax = Double.valueOf(xmaxinput.getText()).doubleValue();
        xmin = Double.valueOf(xmininput.getText()).doubleValue();
    } catch(Exception e) {
        this.showStatus("Error with X range!");
        System.out.println("X range error "+e.getMessage());
        return;
    }

    try {
        ymax = Double.valueOf(ymaxinput.getText()).doubleValue();
        ymin = Double.valueOf(ymininput.getText()).doubleValue();
    } catch(Exception e) {
        this.showStatus("Error with Y range!");
        System.out.println("Y range error "+e.getMessage());
        return;
    }

    try {
        setnum = Double.valueOf(vdatainput.getText()).intValue();
    } catch(Exception e) {
        this.showStatus("Illegal data set value: enter an integer!");
        System.out.println("Dataset error "+e.getMessage());
        return;
    }

    // check for illegal data sets
    try {
        if (setnum < 1 || setnum > 5) throw (new OddData() );
    }
```

```
    } catch(Exception e4) {
        this.showStatus("Error: Data Set between 1 and 5");
        System.out.println("Data Set error ");
        return;
    }

    try {
        smoothing_factor = Double.valueOf(sfinput.getText()).doubleValue();
    } catch(Exception e) {
        this.showStatus("Illegal smoothing factor");
        System.out.println("smoothing factor error "+e.getMessage());
        return;
    }

    // Smoothing factor must be positive real number
    try {
        if (smoothing_factor <= 0.0) throw (new OddSF() );
    } catch(Exception e3) {
        this.showStatus("Error: Make Smoothing Factor > 0");
        System.out.println("Smoothing Factor error ");
        return;
    }

    if(xmin>=xmax || ymin>= ymax ) {
        this.showStatus("Error with Grid range!");
        System.out.println("grid range error! ");
        return;
    }
    if( nx < 2 || ny < 2 ) {
        this.showStatus("Error with Grid dimensions!");
        System.out.println("grid dimension error!");
        return;
    }

    graph.setRange(xmin,xmax,ymin,ymax);

    //the volcano data set is determined from setnum
    volcano.setVolcanoset(setnum);

    //array is the grid to plot
    double array[] = new double[nx*ny];

    this.showStatus("Calculating Grid Values!");
    count = 0;

    // instantiate the volcanodata set
    volcano.getVolcanoes();

    // instaniate lambda1 - a new instance of Restimate
    Restimate lambda1 = new Restimate(smoothing_factor, Volcano.nvolcanoes);

    // fill the array
    for(j=0; j<ny; j++) {
        y = ymin + j*(ymax-ymin)/(ny-1);
        for(i=0; i<nx; i++) {
            x = xmin + i*(xmax-xmin)/(nx-1);
            try {

                // use the epanechnikov method in Restimate
                array[count++] = lambda1.epanh(Volcano.vs, x,y);
            } catch(Exception e) {
                array[count++] = 0.0;
                error = true; }
        }
    }
}
```

88/131

```
    }  
  }  
  
  //instantiate graph with array and array dimensions  
  graph.setGrid(array,nx,ny);  
  
  this.showStatus("Calculating Contours!");  
  
  graph.repaint();  
}  
  
//keep track of Calculate events  
public boolean action(Event e, Object a) {  
    if(e.target instanceof Button) {  
        if( plot.equals(e.target) ) {  
            plot();  
            return true;  
        }  
    }  
    return false;  
}  
}  
  
/* (C) 2000 Southwest Research Institute  
All rights reserved */
```

```

import java.io.*;
import java.math.*;
/*
*****
**
**   Class Restimate
**
*****
**
by Chuck Connor
Center for Nuclear Waste Regulatory Analyses
Southwest Research Institute
6220 Culebra Rd
San Antonio Texas. 78238-5166, USA
cconnor@swri.edu

```

```

Created: September, 1999
      (C) 2000 Southwest Research Institute
      All rights reserved

```

Purpose:

```

*Class to estimate the spatial recurrence rate
* aka the spatial weighting factor or spatial
* intensity, for a given volcano distribution
* results to be plotted on a map
* originally written in TrueBasic by Chuck Connor (1993)
* translated to Java by Chuck Connor (1999)
*
* Contact
* Chuck Connor
* CNWRA, Southwest Research Inst.
* 6220 Culebra Rd
* San Antonio, Tx, 78238-5166, USA
* e-mail: cconnor@swri.org
*
* For more details about the algorithm used in this
* class, see Connor and Hill, 1995, Journal of
* Geophysical Research, 100 (B6) 10,107-10,125.
*****
**/

```

```

public class Restimate {
    public double smoothing_factor, nvolcanoes;

    /** where smoothing_factor is given in kilometers,
        nvolcanoes is the number of volcanoes
    **/

    public Restimate(double smoothing_factor, int nvolcanoes) {

        this.smoothing_factor = smoothing_factor;
        this.nvolcanoes = nvolcanoes;

    }

    public double gaussh (double []vs, double x, double y) {

        /** This method calculates the value of lambda
            for a given smoothing factor (h) using a gaussian
            kernel function.

            Pass this method the single dimensional array vs,

```

containing the point locations of the volcanoes in Universal Transverse Mercator coordinates (meters). The format of array *s* is: *x1,y1,x2,y2,x3,y3,...,xn,yn*. Also pass location for which the estimate is made (*x,y*) also in units of Universal Transverse Mectator coordinates (meters).

The method returns the value of lambda in units
 ***** 1/ km² *****
 about the point *x,y*, for the given data set, and smoothing factor (*h*).

**/

// let h be the smoothing factor in km

double h = smoothing_factor;
 double dist1, dist2;
 double kuu, sumn;
 double lambda;

int i;

// make sure the summation is now zero
 sumn = 0.0;

double xcenter = x/1000.0;
 double ycenter = y/1000.0;

for (i = 0; i < 2*nvolcanoes; i+=2) {

// for each volcano calculate distance² to repox, repoy (now converted to x, ycent

+ er)

dist1 = (xcenter - vs[i]/1000.0)* (xcenter - vs[i]/1000.0) + (ycenter- vs[i+1]/10
 + 00.0)* (ycenter - vs[i+1]/1000.0);
 dist2 = dist1/(h*h);

// use the Gaussian kernel
 kuu = 1/(2*Math.PI) * Math.exp(-0.5*dist2);
 sumn += (1.0/(h*h))* kuu;
 }

//estimate lambda in 1/km²
 lambda = sumn/(double) nvolcanoes;

return lambda;

} // end of the method gaussh

public double epanh (double []vs, double x, double y) {
 /** This method calculates the value of lambda
 for a given smoothing factor (*h*) using an Epanechnikov
 kernel function.

Pass this method the single dimensional array *vs*,
 containing the point locations of the volcanoes in Universal
 Transverse Mercator coordinates (meters). The format of
 array *s* is: *x1,y1,x2,y2,x3,y3,...,xn,yn*. Also pass location for
 which the estimate is made (*x,y*) also in units of Universal
 Transverse Mectator coordinates (meters).

The method returns the value of lambda in units
 ***** 1/ km² *****
 about the point *x,y*, for the given data set, and smoothing factor (*h*).

**/

91/131

```
// let h be the smoothing factor in km
```

```
double h = smoothing_factor;
double dist1, dist2;
double kuu, sumn;
double lambda;
```

```
int i;
```

```
// make sure the summation is now zero
sumn = 0.0;
```

```
double xcenter = x/1000.0;
double ycenter = y/1000.0;
```

```
for (i = 0; i < 2*nvolcanoes; i+=2) {
```

```
    // for each volcano calculate distance^2 to repox, repoy (now converted to x, ycent
```

```
+ er)
```

```
    dist1 = (xcenter - vs[i]/1000.0)* (xcenter - vs[i]/1000.0) + (ycenter - vs[i+1]/10
+ 00.0)* (ycenter - vs[i+1]/1000.0);
    dist2 = dist1/(h*h);
```

```
    // use the Epanechnikov kernel
```

```
    if (dist2 < 1.0) {
        kuu = (2.0/Math.PI)*(1.0 - dist2);
        sumn += (1.0/(h*h))* kuu;
    }
```

```
}
```

```
//estimate lambda in 1/km^2
```

```
lambda = sumn/(double) nvolcanoes;
```

```
return lambda;
```

```
} // end of the method epanh
```

```
}
```

```
/* (C) 2000 Southwest Research Institute
```

```
All rights reserved */
```

```
package graph;
import java.io.*;
```

```
/*
*****
**
**   Class Volcano
**
*****
**
```

```
by Chuck Connor
Center for Nuclear Waste Regulatory Analyses
Southwest Research Institute
6220 Culebra Rd
San Antonio Texas. 78238-5166, USA
cconnor@swri.edu
```

```
Created: December, 1999
      (C) 2000 Southwest Research Institute
      All rights reserved
```

```
Purpose: this class includes datasets 1-5 and defines these data
sets as static variables to be used by other applets in PVHA_YM
```

```
*****
```

```
public class Volcano {

    protected static int volcanoset;
    public static int nvolcanoes;
    public static double [] vs;

    /*   public static int nvolcanoes2;
    public static double [] vs2;
    */

    static {
        volcanoset = 0;
    }

    public void getVolcanoes() {
        switch (volcanoset) {
            case 1:

                // first volcano dataset
                // includes 47 miocene to quaternary volcanic events
                nvolcanoes = 47;
                vs = new double [nvolcanoes*2];

                // Lathrop Wells
                vs[0] = 543780;
                vs[1] = 4060380;

                // Little Black Peak
                vs[2] = 522120;
                vs[3] = 4110340;

                // Hidden Cone
                vs[4] = 523400;
                vs[5] = 4112600;

                // Northern Cone
                vs[6] = 540350;
```

```
vs[7] = 4079360;

// Black Cone
vs[8] = 538840;
vs[9] = 4074120;

// Red Cone
vs[10] = 537580;
vs[11] = 4071880;

// Little Cone 1
vs[12] = 535200;
vs[13] = 4069360;

// Little cone 2
vs[14] = 535480;
vs[15] = 4069560;

//Buckboard Mesa 1
vs[16] = 555180;
vs[17] = 4109200;

// Buckboard Mesa 2
vs[18] = 555500;
vs[19] = 4108500;

// Crater Flat a
vs[20] = 540330;
vs[21] = 4070050;

//Crater Flat b
vs[22] = 540420;
vs[23] = 4068780;

//Crater Flat c
vs[24] = 540360;
vs[25] = 4068440;

// Crater Flat d
vs[26] = 540680;
vs[27] = 4068820;

//Crater Flat e
vs[28] = 540700;
vs[29] = 4068260;

//Crater Flat f
vs[30] = 540300;
vs[31] = 4071600;

//magnetic anomalies
// AAB
vs[32] = 553700;
vs[33] = 4052900;

//AAA 1
vs[34] = 546100;
vs[35] = 4055100;

//AAA 2
vs[36] = 546100;
vs[37] = 4053100;

//AAA 3
```

```
vs[38] = 544500;
vs[39] = 4051400;

//AAC
vs[40] = 547000;
vs[41] = 4042900;

// AAD
vs[42] = 549400;
vs[43] = 4040000;

// AAE
vs [44] = 538300;
vs[45] = 4047200;

// SW Crater Flat
vs[46] =535000;
vs[47] =4067800;

//Thirsty Mesa 1
vs[48] = 529520;
vs[49] = 4112150;

// Thirsty Mesa 2
vs[50] =529480;;
vs[51] =4112040;

//Thirsty Mesa 3
vs[52] =529540;
vs[53] =4111680;

// Nye Canyon N
vs[54] =604680;
vs[55] =4094260;

// Nye Canyon Middle
vs[56] =602170;
vs[57] =4088960;

//Nye Canyon S 1
vs[58] =600950;
vs[59] =4085920;

// Nye Canyon S 2
vs[60] =600550;
vs[61] =4085450;

//Nye canyon ring dike
vs[62] =599160;
vs[63] =4085820;

//Nye Canyon scarp canyon vent
vs[64] =597930;
vs[65] =4082470;

//Frenchman Flat Ue5I
vs[66] =595260;
vs[67] =4080980;

// Frenchman Flat Ue5K
vs[68] =593520;
vs[69] =4081480;

//Yucca Flat UE1h
```

```
vs[70] =582980;
vs[71] =4095280;

//Yucca Flat UE1j
vs[72] =582440;
vs[73] =4096580;

// Yucca Flat UE6d
vs[74] =583740;
vs[75] =4093400;

// Rocket Wash
vs[76] = 536100;
vs[77] = 4109100;

// Paiute Ridge 1
vs[78] = 592400;
vs[79] = 4106800;

// Paiute Ridge 2
vs[80] = 592800;
vs[81] = 4105900;

// Paiute Ridge 3
vs[82] = 593400;
vs[83] = 4105500;

// Paiute Ridge 4
vs[84] = 594800;
vs[85] = 4107900;

// Paiute Ridge 5
vs[86] = 595800;
vs[87] = 4106300;

// Pahute Mesa 1
vs[88] = 548900;
vs[89] = 4133300;

// Pahute Mesa 2
vs[90] = 554100;
vs[91] = 4134500;

// Pahute Mesa 3
vs[92] = 562400;
vs[93] = 4132700;

break;
case 2:

    //second volcano data set
    // includes pliocene and Quaternary
    // little cones considered 1 event
    // buckboard considered 1 event
    // 5 amargosa anomalies included
    // plio crater flat events e and f not included

nvolcanoes = 20;
vs = new double [nvolcanoes*2];

// Lathrop Wells
vs[0] = 543780;
vs[1] = 4060380;
```

```
// Little Black Peak
vs[2] = 522120;
vs[3] = 4110340;

// Hidden Cone
vs[4] = 523400;
vs[5] = 4112600;

// Northern Cone
vs[6] = 540350;
vs[7] = 4079360;

// Black Cone
vs[8] = 538840;
vs[9] = 4074120;

// Red Cone
vs[10] = 537580;
vs[11] = 4071880;

// Little Cone 1
vs[12] = 535200;
vs[13] = 4069360;

//Buckboard Mesa
vs[14] = 555180;
vs[15] = 4109200;

// Crater Flat a
vs[16] = 540330;
vs[17] = 4070050;

//Crater Flat b
vs[18] = 540420;
vs[19] = 4068780;

//Crater Flat c
vs[20] = 540360;
vs[21] = 4068440;

// Crater Flat d
vs[22] = 540680;
vs[23] = 4068820;

//Crater Flat e
vs[24] = 540700;
vs[25] = 4068260;

//magnetic anomalies
// AAB
vs[26] = 553700;
vs[27] = 4052900;

//AAA 1
vs[28] = 546100;
vs[29] = 4055100;

//AAA 2
vs[30] = 546100;
vs[31] = 4053100;

//AAA 3
vs[32] = 544500;
```

97/131

```
vs[33] = 4051400;

//AAC
vs[34] = 547000;
vs[35] = 4042900;

// AAD
vs[36] = 549400;
vs[37] = 4040000;

// AAE
vs [38] = 538300;
vs[39] = 4047200;

break;
case 3:

    // The third volcano data set
    // includes all known Quaternary vents
    // mapped in the Yucca Mountain region.
    // Magnetic anomalies are not included in this data set.

    nvolcanoes = 8;
    vs = new double [nvolcanoes*2];

    // Lathrop Wells
    vs[0] = 543780;
    vs[1] = 4060380;

    // Little Black Peak
    vs[2] = 522120;
    vs[3] = 4110340;

    // Hidden Cone
    vs[4] = 523400;
    vs[5] = 4112600;

    // Northern Cone
    vs[6] = 540350;
    vs[7] = 4079360;

    // Black Cone
    vs[8] = 538840;
    vs[9] = 4074120;

    // Red Cone
    vs[10] = 537580;
    vs[11] = 4071880;

    // Little Cone 1
    vs[12] = 535200;
    vs[13] = 4069360;

    // Little Cone 2
    vs[14] = 535480;
    vs[15] = 4069560;

    break;
case 4:

    // The fourth volcano data set
    // includes three events.
    // These are the Quaternary Crater Flat volcano alignment,
    // taken as centered on Red Cone,
```

```
// the Sleeping Butte alignment,
// taken as centered on Hidden Cone,
// and Lathrop Wells volcano.

nvolcanoes = 3;
vs = new double [nvolcanoes*2];

// Lathrop Wells
vs[0] = 543780;
vs[1] = 4060380;

// Hidden Cone
vs[2] = 523400;
vs[3] = 4112600;

// Red Cone
vs[4] = 537580;
vs[5] = 4071880;

break;

case 5:

    // fifth volcano dataset
    // includes all vent locations reported in Data Table
    // some basalt outcrops with unknown vent locations
    // are not included

    nvolcanoes = 64;
    vs = new double [nvolcanoes*2];

    // Lathrop Wells
    vs[0] = 543780;
    vs[1] = 4060380;

    // Little Black Peak
    vs[2] = 522120;
    vs[3] = 4110340;

    // Hidden Cone
    vs[4] = 523400;
    vs[5] = 4112600;

    // Northern Cone
    vs[6] = 540350;
    vs[7] = 4079360;

    // Black Cone
    vs[8] = 538840;
    vs[9] = 4074120;

    // Red Cone
    vs[10] = 537580;
    vs[11] = 4071880;

    // Little Cone 1
    vs[12] = 535200;
    vs[13] = 4069360;

    // Little cone 2
    vs[14] = 535480;
    vs[15] = 4069560;

    //Buckboard Mesa 1
```

```
vs[16] = 555180;
vs[17] = 4109200;

// Buckboard Mesa 2
vs[18] = 555500;
vs[19] = 4108500;

// Crater Flat a
vs[20] = 540330;
vs[21] = 4070050;

//Crater Flat b
vs[22] = 540420;
vs[23] = 4068780;

//Crater Flat c
vs[24] = 540360;
vs[25] = 4068440;

// Crater Flat d
vs[26] = 540680;
vs[27] = 4068820;

//Crater Flat e
vs[28] = 540700;
vs[29] = 4068260;

//Crater Flat f
vs[30] = 540300;
vs[31] = 4071600;

//magnetic anomalies
// AAB
vs[32] = 553700;
vs[33] = 4052900;

//AAA 1
vs[34] = 546100;
vs[35] = 4055100;

//AAA 2
vs[36] = 546100;
vs[37] = 4053100;

//AAA 3
vs[38] = 544500;
vs[39] = 4051400;

//AAC
vs[40] = 547000;
vs[41] = 4042900;

// AAD
vs[42] = 549400;
vs[43] = 4040000;

// AAE
vs [44] = 538300;
vs[45] = 4047200;

// SW Crater Flat
vs[46] =535000;
vs[47] =4067800;
```

100 / 131

```
//Thirsty Mesa 1
vs[48] = 529520;
vs[49] = 4112150;

// Thirsty Mesa 2
vs[50] =529480;;
vs[51] =4112040;

//Thirsty Mesa 3
vs[52] =529540;
vs[53] =4111680;

// Nye Canyon N
vs[54] =604680;
vs[55] =4094260;

// Nye Canyon Middle
vs[56] =602170;
vs[57] =4088960;

//Nye Canyon S 1
vs[58] =600950;
vs[59] =4085920;

// Nye Canyon S 2
vs[60] =600550;
vs[61] =4085450;

//Nye canyon ring dike
vs[62] =599160;
vs[63] =4085820;

//Nye Canyon scarp canyon vent
vs[64] =597930;
vs[65] =4082470;

//Frenchman Flat Ue5I
vs[66] =595260;
vs[67] =4080980;

// Frenchman Flat Ue5K
vs[68] =593520;
vs[69] =4081480;

//Yucca Flat UE1h
vs[70] =582980;
vs[71] =4095280;

//Yucca Flat UE1j
vs[72] =582440;
vs[73] =4096580;

// Yucca Flat UE6d
vs[74] =583740;
vs[75] =4093400;

// Rocket Wash
vs[76] = 536100;
vs[77] = 4109100;

// Paiute Ridge 1
vs[78] = 592400;
vs[79] = 4106800;
```

```
// Paiute Ridge 2
vs[80] = 592800;
vs[81] = 4105900;

// Paiute Ridge 3
vs[82] = 593400;
vs[83] = 4105500;

// Paiute Ridge 4
vs[84] = 594800;
vs[85] = 4107900;

// Paiute Ridge 5
vs[86] = 595800;
vs[87] = 4106300;

// Pahute Mesa 1
vs[88] = 548900;
vs[89] = 4133300;

// Pahute Mesa 2
vs[90] = 554100;
vs[91] = 4134500;

// Pahute Mesa 3
vs[92] = 562400;
vs[93] = 4132700;

// basalt of sleeping butte 1
vs[94] = 525700;
vs[95] = 4112100;

//basalt of sleeping butte 2
vs[96] = 524300;
vs[97] = 4113600;

// Solitario canyon
vs[98] = 546800;
vs[99] = 4082400;

// Miocene basalt of SW crater flat 1
vs[100] = 536400;
vs[101] = 4064000;

// Miocene basalt of SW Crater Flat 2
vs[102] = 534700;
vs[103] = 4066500;

// basalt of VH-2
vs[104] = 537900;
vs[105] = 4072950;

// basalt of Kiwi Mesa
vs[106] = 568940;
vs[107] = 4078740;
vs[108] = 568820;
vs[109] = 4079000;

// basalt of NE Amargosa Desert
vs[110] = 563300;
vs[111] = 4046500;

// beatty basalt
vs[112] = 525300;
```

```
vs[113] = 4085600;
vs[114] = 527400;
vs[115] = 4085200;
vs[116] = 514800;
vs[117] = 4090800;
```

```
// grapevine
vs[118] = 476400;
vs[119] = 4101800;
vs[120] = 476800;
vs[121] = 4102700;
vs[122] = 477900;
vs[123] = 4106600;
```

```
// southern Death Valley
// Cinder Hill
vs[124] = 523900;
vs[125] = 3977100;
// Shoreline Butte
vs[126] = 526200;
vs[127] = 3973700;
```

```
break;
```

```
default:
    nvolcanoes = 1;
    vs = new double[nvolcanoes*2];
    vs[0] = 0.0;
    vs[1] = 0.0;
}
```

```
}

public void setVolcanoset( int n) {
    volcanoset = n;
}
```

```
}
/* (C) 2000 Southwest Research Institute
All rights reserved */
```

Information on Pages 103 through 131 contains GSA Bulletin copyright information and is therefore not included in this file.