ORNL/TM-12899

ornl

OAK RIDGE NATIONAL LABORATORY

MARTIN MARIETTA

MONTE CARLO TESTS OF THE ELIPGRID-PC ALGORITHM

J. R. Davidson

MANAGED BY
MARTIN MARIETTA ENERGY SYSTEMS, INC.
FOR THE UNITED STATES
DEPARTMENT OF ENERGY

ORNL/TM-12899

MONTE CARLO TESTS OF THE ELIPGRID-PC ALGORITHM

J.R. Davidson

Date Published: April, 1995

Prepared by
Oak Ridge National Laboratory
Oak Ridge, Tennessee 37831

managed by
Martin Marietta Energy Systems, Inc.
for the
U. S. Department of Energy
under contract No. DE-AC05-84OR21400

CONTENTS

FIGURES

TABLES

ACKNOWLEDGEMENTS

The author would like to recognize four individuals of ORNL Grand Junction for help relating to the Monte Carlo validation of the ELIPGRID-PC algorithm:

John E. Wilson for a key insight relating to a coordinate transformation equation used in the Monte Carlo algorithm.

Gloria H. Stevens, Project Manager for the ORNL Independent Verification Contract on the Grand Junction Project Office Remedial Action Project, for strong support of this project.

Joella Krall for her careful and thoughtful editorial work.

Phil V. Egidi for his timely help with graphics.

EXECUTIVE SUMMARY

The standard tool for calculating the probability of detecting pockets of contamination called hot spots has been the ELIPGRID computer code of Singer and Wickman. The ELIPGRID-PC program has recently made this algorithm available for an IBM® PC. However, no known independent validation of the ELIPGRID algorithm exists. This document describes a Monte Carlo simulation-based validation of a modified version of the ELIPGRID-PC code. The modified ELIPGRID-PC code is shown to match Monte Carlo-calculated hot-spot detection probabilities to within ±0.5% for 319 out of 320 test cases. The one exception, a very thin elliptical hot spot located within a rectangular sampling grid, differed from the Monte Carlo-calculated probability by about 1%. These results provide confidence in the ability of the modified ELIPGRID-PC code to accurately predict hot-spot detection probabilities within an acceptable range of error.

1. INTRODUCTION

Evaluating the need for remedial cleanup at a waste site can involve both finding average contaminant concentrations and identifying pockets of contamination called hot spots. The standard approach for calculating the probability of detecting elliptical hot spots is based on the ELIPGRID computer program developed by Singer (Singer 1972). Recently, Oak Ridge National Laboratory developed ELIPGRID-PC, making the ELIPGRID algorithm available for an IBM® personal computer (PC) or compatible (Davidson 1994).

Although ELIPGRID has been available for many years, no known independent validation of the algorithm is available. The ELIPGRID-MC program, documented here, provides a Monte Carlo simulation-based validation of the ELIPGRID-PC version of the ELIPGRID algorithm.

2. PREVIOUS WORK

The process of moving the ELIPGRID program to the PC environment uncovered two problems with the original algorithm. First, the results of running the ELIPGRID algorithm did not match Singer's (1972) published results for certain rectangular grid cases (Davidson 1994). Second, the original ELIPGRID algorithm was found to produce negative probabilities of missing a hot spot for a range of triangular grid cases (Davidson 1994, Appendix C). The approach to resolving these problems is briefly reviewed here.

The RECT subroutine in ELIPGRID transforms a rectangular grid to a square grid using an affine transformation described in detail by Singer and Wickman (1969). However, a small portion of the FORTRAN code in the RECT subroutine is not equivalent to the mathematical equation it is based on (Davidson 1994, Appendix A). By modifying the

FORTRAN code to correspond to the underlying equation, all of Singer's rectangular grid test case results were successfully matched by the ELIPGRID-PC version of ELIPGRID (Davidson 1994, Appendix B).

The triangular grid problem relates directly to the *L/G* ratio, the ellipse semi-major axis length, *L*, divided by the grid size, *G*, for a given ellipse and grid. For a significant portion of the *L/G* ratios between 0.5 and 0.6, the ELIPGRID algorithm produced negative probabilities of missing existing hot spots. Also, a sharp discontinuity near the *L/G* ratio of 0.58 was uncovered (Davidson 1994, Appendix C). The problem was found to be confined to ellipses with shapes greater than about 0.85 and less than 1.0. The shape of an ellipse is the ratio of the short axis length to the long axis length (or semi-minor axis ÷ semi-major axis).

ELIPGRID-PC did not provide a modified ELIPGRID algorithm to solve the triangular grid problem. Instead, a fourth-order polynomial regression equation of the probability of missing a hot spot versus the *L/G* ratio was supplied as a substitute for the ELIPGRID algorithm in the triangular grid problem region. Also, code was added to eliminate negative probabilities from being produced.

Since the ELIPGRID algorithm had never been independently validated and the ELIPGRID-PC program development revealed these rectangular and triangular grid problems, an independent test of ELIPGRID was undertaken. One independent test method would be to "hide" hot spots in a field and then simply count the number of hits for various sampling grids. The resulting detection probabilities found by direct count would be compared with the results analytically calculated by ELIPGRID. However, the economics of this empirical test method are certainly prohibitive, while a computer simulation using randomly placed hot spots could find the probability of detection very economically. Such a simulation method utilizing random numbers to mimic a process or system in nature or industry is commonly called a Monte Carlo simulation. This simulation, as related to the hot-spot probability problem, is described herein.

3. THE MONTE CARLO ALGORITHM

A Monte Carlo method that relates to the problem of determining hot spots is called the hit or miss Monte Carlo method. The following is a description of how this method works (Yakowitz 1977).

If $f(x)$ is a piecewise continuous real function such that $0 \le f(x) \le 1$ when x is between 0 and 1, the area A under the curve of $f(x)$ can be directly evaluated using $A = \int_0^1 f(x)\,dx$ (Fig. 1). However, the hit or miss Monte Carlo method provides an estimate of A even when the integral cannot be evaluated directly.

If n random points are generated on the unit square, a certain number, n_A, will fall on or below the $y = f(x)$ curve. The hit or miss Monte Carlo algorithm states that as n goes to infinity, the fraction $\frac{n_A}{n}$ approaches the true area as a limit, that is

$$A = \lim_{n \to \infty} \frac{n_A}{n}.$$

A simple example of the hit or miss Monte Carlo method is to calculate the area A of the region under the unit circle in the first quadrant (Fig. 2). The hit or miss Monte Carlo method gives $A \approx \frac{n_A}{n}$, where n_A is the number of random points such that $x^2 + y^2 \le 1$ and n is the total number of random points. In other words, the area A is determined by generating many random points on the unit square and finding the fraction that falls within the desired region.

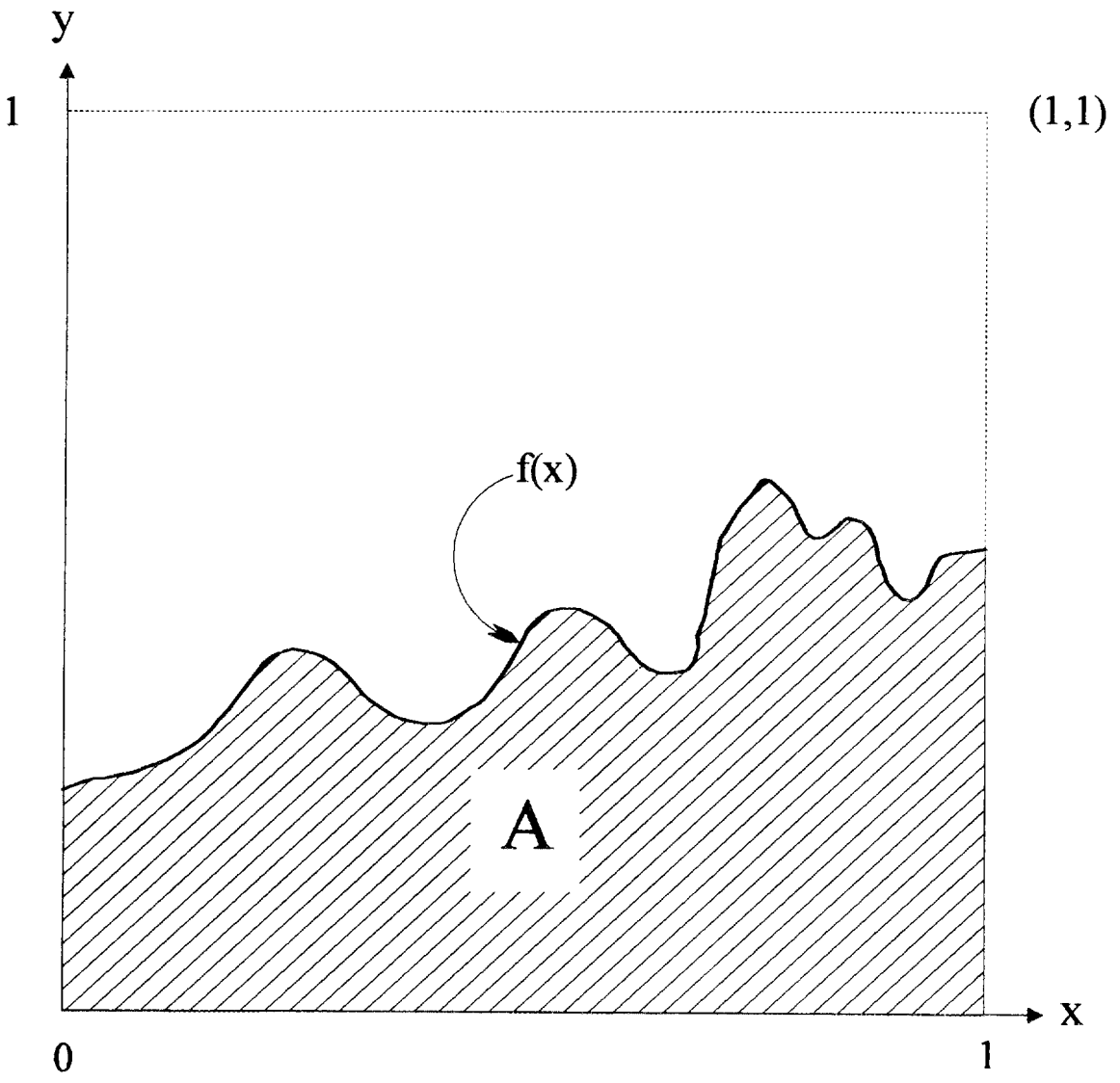

Fig. 1. The curve of *f(x)* on a unit square and *A*, the area under the curve.

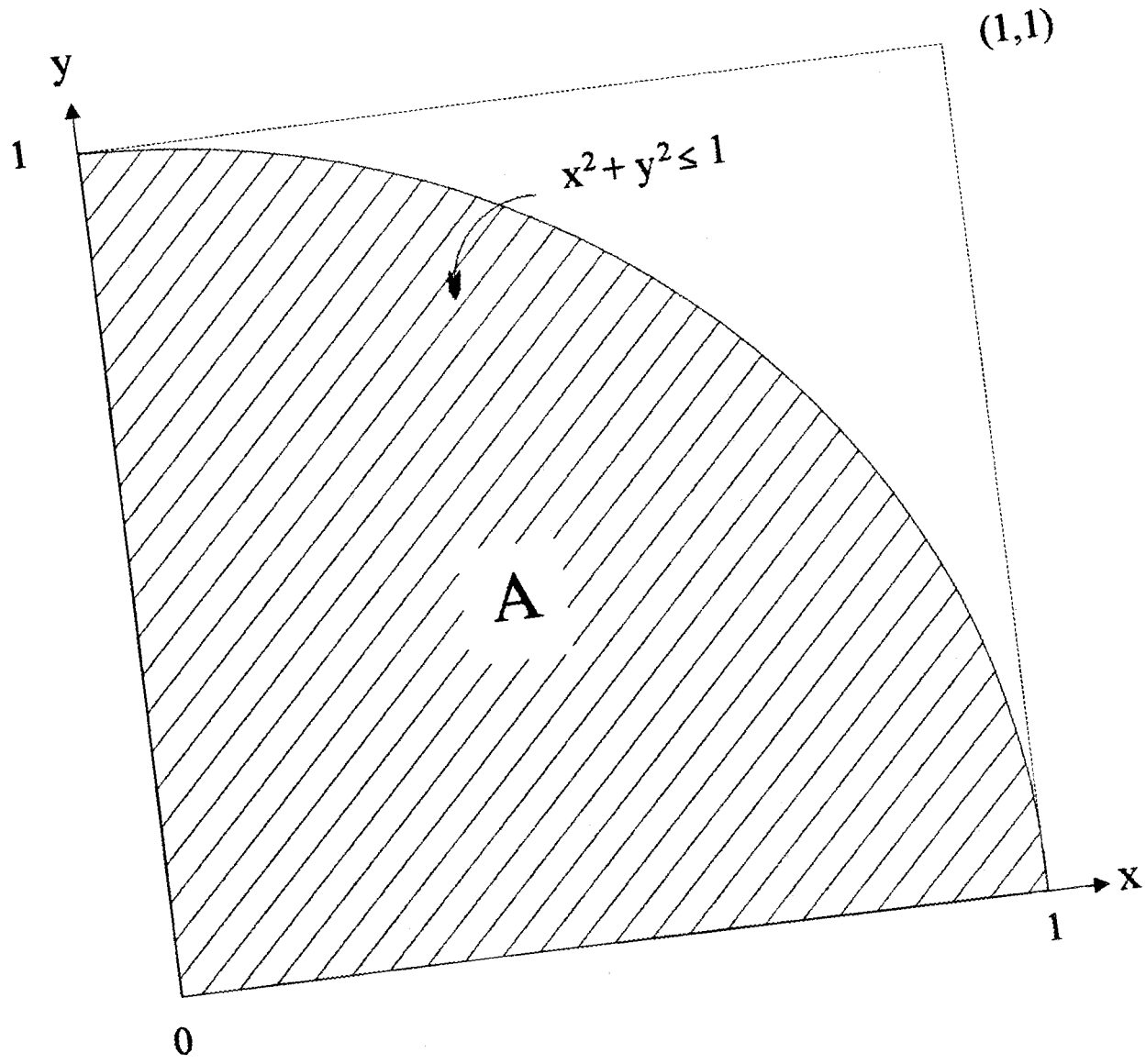

Fig. 2. Simple example of the hit or miss Monte Carlo method.

4. THE ELIPGRID-MC ALGORITHM

4.1 OVERVIEW

The ELIPGRID-MC Monte Carlo method is similar to the hit or miss Monte Carlo technique just described. A square, rectangular, or triangular grid of sampling locations, such as illustrated in Fig. 3, is used. It is assumed that elliptical hot spots with random center points, specified shapes and orientation angles, and minimum sizes are located on a grid and by symmetry, that the random center points all fall in one grid cell. Detection of a hot spot (a hit) occurs when any portion of an ellipse covers one or more sampling locations (grid nodes).

Thus, if n random centers, all within one cell (Fig. 4), are generated, a certain number of the ellipses, n_H, will cover at least one grid node and should be considered hits. The ELIPGRID-MC algorithm assumes that as n goes to infinity, the fraction $\frac{n_H}{n}$ approaches the true hit probability, P(hit), as a limit: $P(\text{hit}) = \lim_{n \to \infty} \frac{n_H}{n}$.

An ellipse with a random center (x, y) is sampled by any node if

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} \leq 1,$$

where a is the semi-major axis of the ellipse and b is the semi-minor axis of the ellipse.

The formula for an ellipse in standard position, i.e., an ellipse at the origin of the x, y coordinate system with its semi-major axis on the x axis, is $\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$. Since the hot spot's center need not be at the origin and the semi-major axis can be tilted with respect to the x axis, this equation must be transformed through standard transformation of coordinates equations involving translation and rotation of coordinate systems.

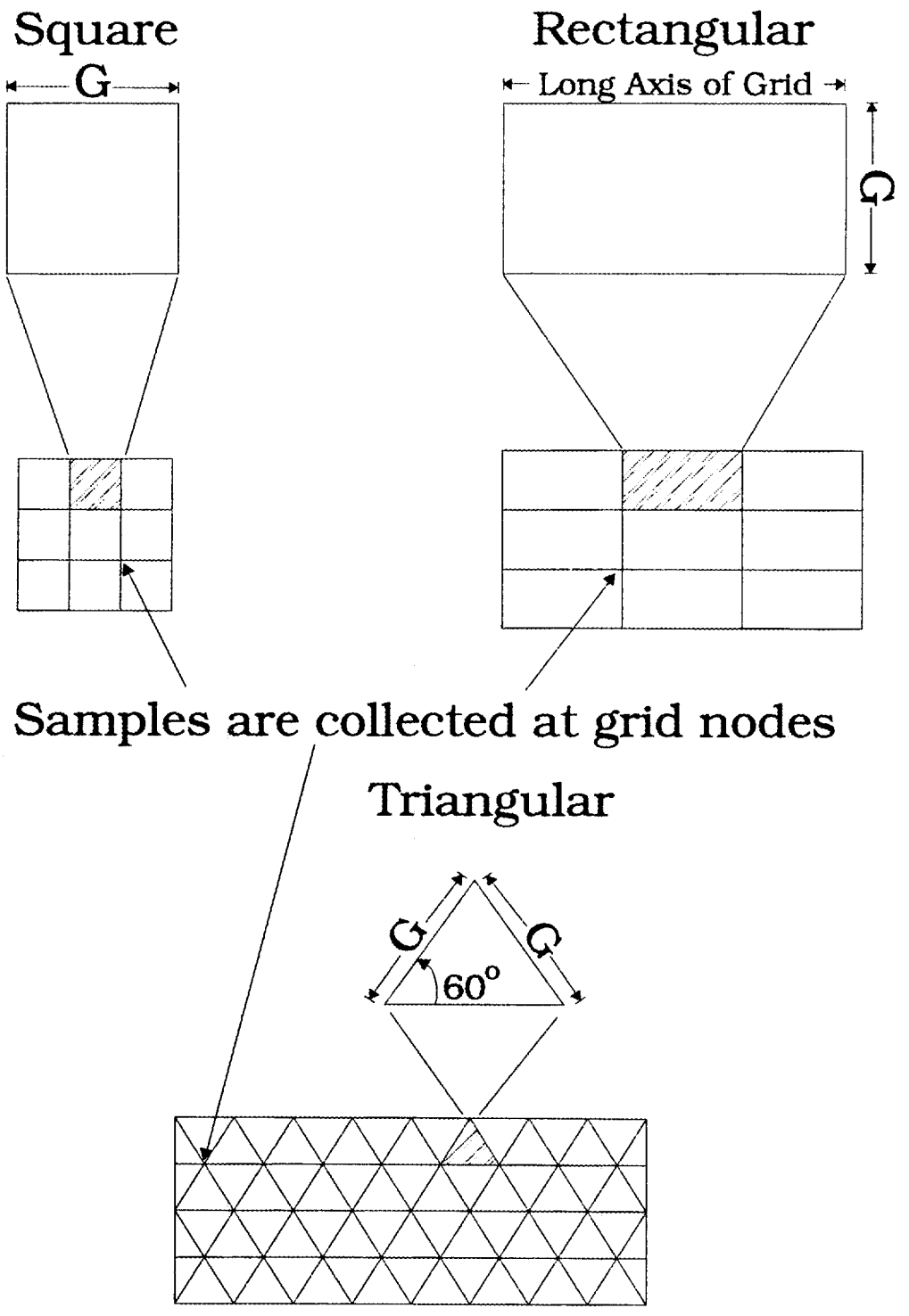

Fig. 3. Grid configuration for finding hot spots.

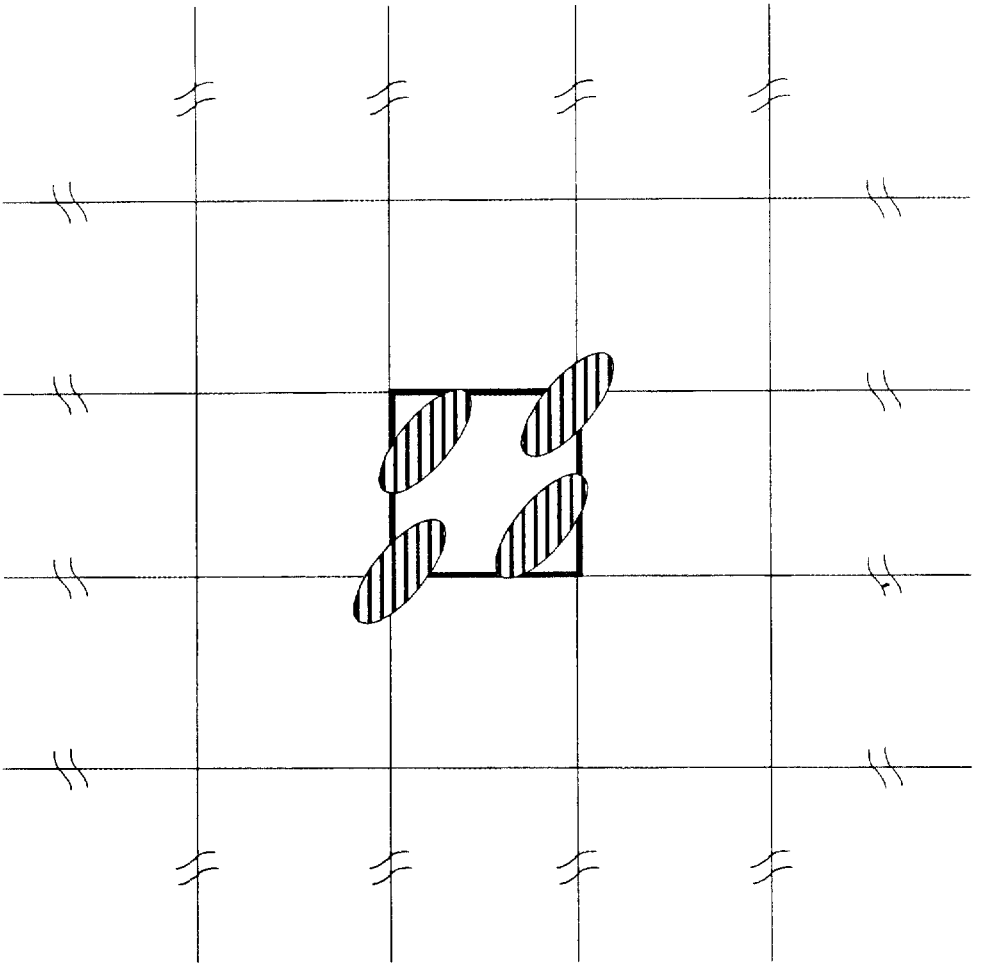

Fig. 4. Four random ellipses, two of them overlapping sampling nodes.

4.2 THE RANDOM NUMBER GENERATOR

The basis for the Monte Carlo simulation is a valid random number generator that can uniformly place the center of the ellipse on a grid cell. Many system-supplied random number generators and various published algorithms for random number generators have serious flaws. Creating and verifying a random function for ELIPGRID-MC was a first step in producing a viable program.

Park and Miller (1988) have provided an algorithm for a minimal standard that they believe should always be used, unless a random number generator *known* to be better is available. The minimal standard proposed by Park and Miller (1988) is a multiplicative linear congruential generator with multiplier 16,807 and prime modulus $2^{31} - 1$. This generator has the following form in Pascal:

```
var seed : integer;
seed := 1;          (* Note: seed can be any integer from 1 to m - 1 *)
function Random : real;
                    (* Integer Version 1 *)
const
  a = 16807;              (* multiplier *)
  m = 2147483647;  (* prime modulus = 2^31 - 1 *)

begin
  seed := (a * seed) mod m;
  Random := seed / m;
end;
```

Park and Miller (1988) state that this minimal standard "should be tested for correctness, *not* randomness. If the implementation is correct, tests for randomness are redundant." It need not be tested for randomness since "the generator has been exhaustively tested and

its characteristics are well understood" (Park and Miller 1988). However, any implementation is dependent on the arithmetic characteristics of the host computer (such as overflow for integer products) and, therefore, should be checked for correctness.

A simple but effective test for correctness is to call the random number generator 10,000 times and then check the current value of *seed* (Park and Miller 1988). If this value is exactly 1,043,618,065, the generator is correct. In ELIPGRID-MC, this test is performed the first time the program is run on a computer. If the result is acceptable, a small file, RandTest.Ok is written to the default drive as a note for future program runs. If the result is not acceptable, an error message is displayed and the program aborts. For details of this process, see the code for function TestUnifRand() in file EGMCMCAl.Prg in Appendix C.

CA-Clipper®, the language used for ELIPGRID-MC, does not provide integer numeric variables as the Pascal function Integer Version 1 calls for. However, CA-Clipper® stores numeric values in a standard double precision floating point format with numeric precision guaranteed up to 16 significant digits (Computer Associates 1992). The use of these numeric values in the ELIPGRID-MC random function UnifRand() has been entirely satisfactory. All trials have passed the correctness test based on generating the test number. The actual code for UnifRand() with comments omitted follows:

```
Function UnifRand()
#define nMULTIPLIER 16807
#define nMODULUS 2147483647
nSeed := (nMULTIPLIER * nSeed) % nMODULUS
return (nSeed/nMODULUS)
```

Note that % is the modulus operator in CA-Clipper®. Previous to the first call of UnifRand(), nSeed is initialized to 1 with the following statement:

```
static nSeed := 1.
```

Thus, the first random number generated will always be

$$((16{,}807 * 1) \% 2{,}147{,}483{,}647) \div 2{,}147{,}483{,}647 = 16{,}807 \div 2{,}147{,}483{,}647 \approx 7.826 \times 10^{-6}.$$

4.3 THE NODE CHECKING ALGORITHM

Another key process of the ELIPGRID-MC program is to determine how many nodes and which nodes to check for possible hot-spot detection. An algorithm was written to accomplish this.

Figure 5 illustrates the node checking process with a random ellipse on a square grid. With a box drawn around the ellipse, it is visually clear that no node above point A can possibly detect the current ellipse. Similarly, no node to the right of point B, no node below point D, and no node to the left of point C can detect the ellipse. These maximum and minimum box points lead to the maximum and minimum grid rows and columns that need to be checked for possible hot-spot detection. The node checking algorithm that codifies this intuitive approach is summarized below:

1. The maximum grid row and maximum grid column that can contain a node that could detect the current ellipse is determined.
2. The minimum grid row and minimum grid column that can contain a node that could detect the current ellipse is found.
3. Each candidate grid node, beginning with the node at the intersection of the minimum grid row and minimum grid column, is checked. The coordinates of the current candidate grid node are transformed into coordinates based on the x_T, y_T coordinate system, with origin at the ellipse center and x_T axis on the semi-major axis of the ellipse. The ellipse is in standard position with respect to the x_T, y_T coordinate system. The transformed grid node coordinates are labeled (x_T, y_T).

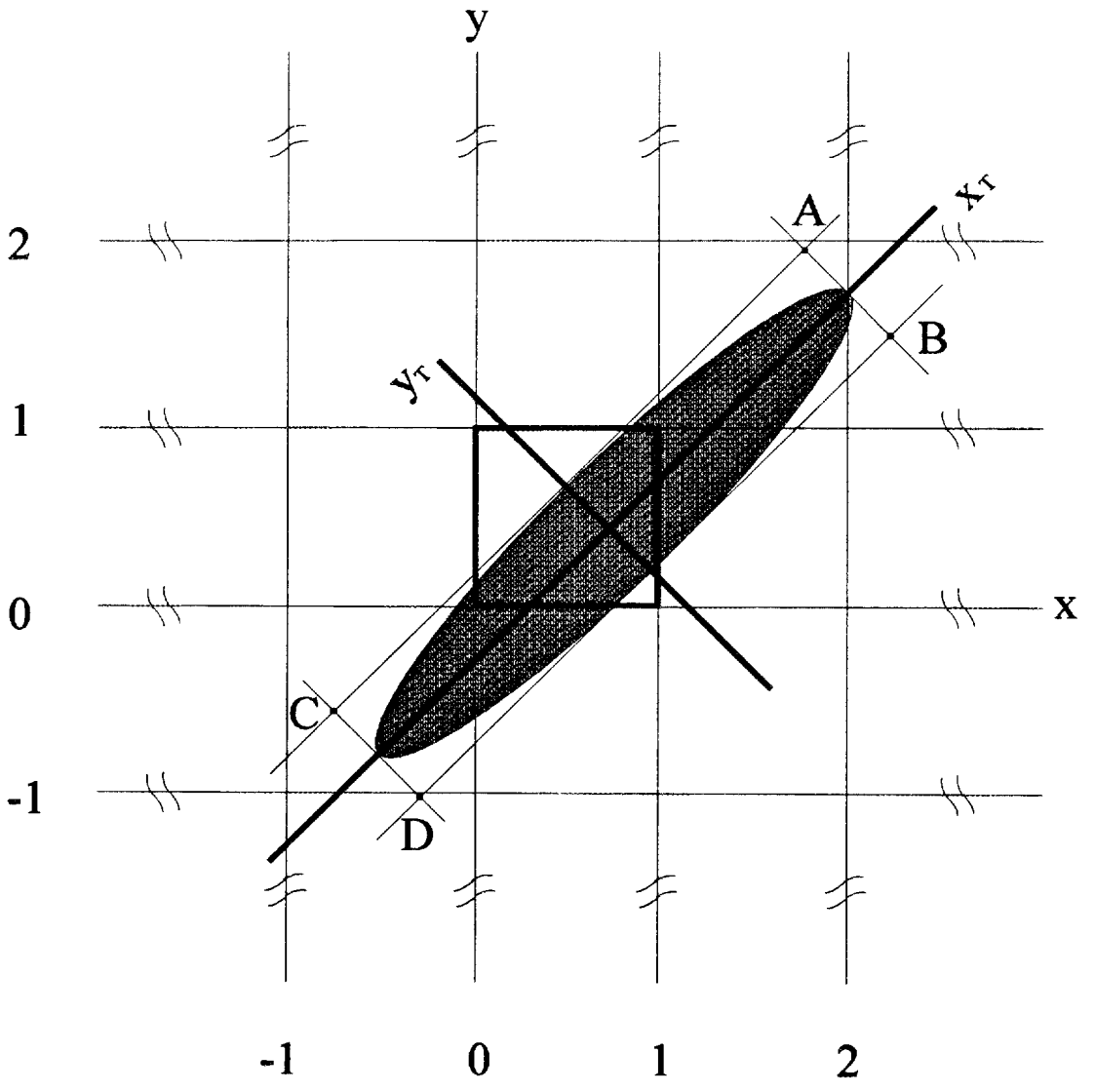

Fig. 5. Illustration of the node checking process.

4. The transformed node coordinates are substituted into the following equation:

$$\frac{x_T}{a^2} + \frac{y_T}{b^2} \leq 1 ,$$

where a is the semi-major axis of ellipse and b is the semi-minor axis of ellipse. If the equation is satisfied, the current grid node would detect the elliptical hot spot. The number of hits for this simulation run are incremented and the node checking loop is exited. If the equation is not satisfied, the next candidate grid node is checked. When all candidate grid nodes have been checked, the node checking algorithm is completed, and the next random ellipse is called up.

To consider Steps 1 and 2 in more detail, the process of finding the maximum grid row consists of finding the x, y coordinates of point A in Fig. 5 by first finding the transformed coordinates x_T, y_T of A and then converting them back into grid coordinate system coordinates x, y. The transformation equations for converting from the ellipse coordinate system back to the grid coordinate system are (Spiegel 1968):

$$x = x_T \cos \alpha - y_T \sin \alpha + x_0$$
$$y = x_T \sin \alpha + y_T \cos \alpha + y_0$$

where

x_T, y_T = coordinates in the ellipse coordinate system,

x, y = coordinates in the grid coordinate system,

x_0, y_0 = coordinates of the ellipse random center point,

α = angle of the ellipse (and the ellipse coordinate system) to the grid.

By definition of the semi-minor axis of the ellipse, the y_T coordinate of A is simply the length of the semi-minor axis. Using the above transformation equations, it is possible to find from y_T the y coordinate of point A. The closest grid row below this coordinate is

the maximum grid row whose nodes must be checked. A similar process is used to find all the minimum and maximum grid rows and columns.

Step 3 is implemented with two **for-next** loops that step through all possible grid nodes, from the node with the minimum row and column to the node with the maximum row and column.

Step 4 transforms the x, y coordinates of the current grid nodes into the ellipse-based coordinate system using the following transformation equations (Spiegel 1968):

$$x_T = (x - x_0) \cos \alpha + (y - y_0) \sin \alpha$$
$$y_T = (y - y_0) \cos \alpha - (x - x_0) \sin \alpha,$$

where

x, y = coordinates in the grid coordinate system,

x_T, y_T = coordinates in the ellipse coordinate system,

x_0, y_0 = coordinates of the ellipse random center point,

α = angle of the ellipse (and the ellipse coordinate system) to the grid.

Step 5 completes the node checking algorithm. When a node is found that would detect the current ellipse, the CA-Clipper® break() function is used to leave both **for-next** loops of the node checking algorithm. The code then determines the next random ellipse, and the node checking process is repeated. If no node is found that detects the ellipse, the **for-next** loops are completed.

The node checking algorithm is essentially the same for rectangular grids as the algorithm just described for square grids. For details, see code in file EGMCMCAl.Prg in Appendix C. The node checking code for triangular grids is separated from that for rectangular or square grids. Triangular grids have the extra complexity of node columns not being in a straight line. For details of how the algorithm deals with the alternating x coordinates for a given triangular grid node column, see the detailed pseudocode in Sect. 4.5 or the actual code in Appendix C.

4.4 STATISTICAL ANALYSIS OF MONTE CARLO-CALCULATED VALUES

In regard to the statistical analysis of Monte Carlo simulation output data, Law and Kelton state that simulation output data are almost never independent and that classical statistical analysis based on independent identically distributed observations is not directly applicable (Law and Kelton 1982). However, in a series of n simulation runs, if each simulation uses a different set of random numbers, it is possible to calculate a confidence interval for the mean, μ, of the results using

$$\bar{X}(n) \pm t_{n-1,1-\alpha/2} \sqrt{\frac{s^2(n)}{n}},$$

where

n = number of simulation runs;

$\bar{X}(n)$ = sample mean of the n simulation run results;

$t_{n-1,1-\alpha/2}$ = t score with n-1 degrees of freedom for a 100 (1 - α) percent confidence interval for the true mean, μ;

$s^2(n)$ = sample standard deviation of the n simulation run results.

In the Monte Carlo code for ELIPGRID-MC, n is set to a minimum of 10 runs. Each run consists of exactly 1,000 random trials, and the t-score value is based on the value required for a 99% confidence interval, i.e., α = 0.01. After 10 runs have been completed, the current calculated 99% confidence half-interval for n = 10 runs is compared to 0.005. If it exceeds 0.005, another run of 1,000 trials is completed. Otherwise, the results are printed out. A maximum of 100 simulation runs is allowed. Thus, the maximum total number of trials for a series of simulation runs is 100,000. For the 320 cases considered in this document, 91,000 trials was the maximum number required to reach the 99% confidence half-interval bound of 0.005.

4.5 PSEUDOCODE FOR ELIPGRID-MC MONTE CARLO ALGORITHM

Pseudocode for ELIPGRID-MC, a summary or overview of the logic of the code, is presented below. The following pseudocode style is adapted from *Mathematical Modeling* (Meerschaert 1993).

Algorithm: ELIPGRID-MC provides Monte Carlo simulation of ELIPGRID type hot-spot hit probabilities.

Summary: On a square, rectangular, or triangular sampling grid of unlimited size, thousands of ellipses are placed at random within one grid cell. The number of ellipses that cover a grid sampling node are counted and divided by the total number of ellipses to obtain an estimate of the probability that the sampling grid will detect a random elliptical hot spot on at least one sampling node. The probability of detecting larger hot spots using the same sampling grid will be as good or better than this estimate.

Variables:

SemiMajor	=	Length of semi-major axis of hot spot
Shape	=	Elliptical hot spot minor/major axis ratio
Angle	=	Orientation angle of hot spot to grid in degrees
GSize	=	Grid size (for rectangular grid, short side)
RecRatio	=	Long to short side ratio for rectangular grids
GTyp	=	Grid type: 1 = square, 2 = triangle, 3 = rectangle
NumRuns	=	Number of runs actually needed to achieve acceptable error

Constants: *MaxRuns* = Maximum number of simulation runs = 100
NumTrials = Number of trials per simulation run = 1,000
DsrdBound = Desired 99% confidence half-interval = 0.005

Input: *SemiMajor, Shape, Angle, GSize, RecRatio, Gtyp*

The algorithm is given in summary, then in detailed form. An asterisk or a double forward slash indicate a comment that is not part of the actual algorithm. The following operator definitions apply:

:= or = = Assignment operator
== = Test of equality operator
!= = Test of inequality operator
.or. = Logical "or" operator
.and. = Logical "and" operator

Process:

* Initialize variables, counters, etc.
* Loop through as many simulation runs as needed, up to MaxRuns
for *Run* = 1 to *MaxRuns* // *MaxRuns* is set to 100 in previous code
* Do each trial for one simulation run
for *Trial* = 1 to *NumTrials* // *NumTrials* is set to 1,000
* Get coordinates of random center
* Get node minimums and maximums
* Check nodes for a hit and count hits
next *Trial*
* If 10 or more runs have been completed, do statistics
* Exit if bound criteria has been reached
next *Run*

Output: *MeanProb, StdDev, CurBound, TotalTrials*

99% Confidence Interval = *MeanProb* ± *CurBound*

Detailed Process:

```
Initialize variables, (only key variables listed here)
* Set array of hit probabilities, one for each run, to null
* The ith run hit probability is NumHits/NumTrials
aProbHit := {}       // array of hit probabilities
* LtoG is hot spot semi-major axis scaled by grid size
LtoG := SemiMajor/GSize
* Initialize array of Student's t scores for 99% conf. intervals
InitTScores(aTScores)
* Seed random number generator
SeedUnifRand(1)      // 1 is initial seed
* ASq is square of the scaled semi-major axis
ASq := LtoG * LtoG
* BSq is square of the scaled semi-minor axis
BSq := (LtoG * Shape) * (LtoG * Shape)
* Constants needed to speed up finding min/max X,Y nodes
* These equations from coordinate transformation equations (Spiegel 1968)
MinXCnst := -LtoG * cos(Angle) - LtoG * Shape * sin(Angle)
MaxXCnst :=  LtoG * cos(Angle) + LtoG * Shape * sin(Angle)
MinYCnst := -LtoG * sin(Angle) - LtoG * Shape * cos(Angle)
MaxYCnst :=  LtoG * sin(Angle) + LtoG * Shape * cos(Angle)

* Loop through as many simulation runs as needed
for Run = 1 to MaxRuns       // MaxRuns is set to 100 in earlier code
    * NumHits is reset here for each run
    NumHits = 0
    * Do each trial for a simulation run
    for Trial = 1 to NumTrials    // NumTrials is set to 1,000 in earlier code
        * Increment total trials counter
        TotTrials++
        * Get coordinates of random center, (RandX, RandY)
        if GTyp != TRI_GRID
            * For square and rectangular grids
            RandX := UnifRand() * nRecRatio
            RandY := UnifRand()
```

```
else
    * For triangular grids
    * RandX will vary from = 0.0 to = 1.0
    * RandY will vary from = 0.0 to = 0.866, i.e., sin(60)
    RandX := UnifRand()
    RandY := sin(60) * UnifRand()
endif
* Get node minimums and maximums
if GTyp == SQR_GRID
    * For square grids
    MinX := ceiling(MinXCnst + RandX)
    MaxX := floor(MaxXCnst + RandX)
    MinY := ceiling(MinYCnst + RandY)
    MaxY := floor(MaxYCnst + RandY)
elseif GTyp == nTRI_GRID
    * For tri grids
    MinYRow := int((MinYCnst + RandY)/sin(60))
    MinY    := MinYRow * sin(60)
    MaxYRow := int((MaxYCnst + RandY)/sin(60))
    MaxY    := MaxYRow * sin(60)
    MinX    := int((MinXCnst + RandX)/0.5) * 0.5
    * If min Y row is even and min X is not an integer, add 0.5.
    if MinYRow % 2 == 0 .and. int(MinX) != MinX
       MinX += 0.5
    endif
    MaxX := int((MaxXCnst + RandX)/0.5) * 0.5
elseif GTyp == REC_GRID
    * For rect grids
    MinX := ceiling((MinXCnst + RandX) / RecRatio) * RecRatio
    MaxX := floor(  (MaxXCnst + RandX) / RecRatio ) * RecRatio
    MinY := ceiling(MinYCnst + RandY)
    MaxY := floor(MaxYCnst + RandY)
endif
* Check nodes for a hit; first see if a hit is possible
if MaxX < MinX .or. MaxY < MinY
    * If here, hot spot is between grid lines, cannot be hit
    * Just loop back to next trial
    loop
endif
* Now sequence through all possible nodes, looking for a hit
* Break out of sequence when a hit occurs
if GTyp != TRI_GRID
    * For square and rectangular grids
```

```
    begin sequence
        for XNode = MinX to MaxX step RecRatio
            for YNode = MinY to MaxY
                * Transform nodes x-coord. using transformation equation
                XNodeT := ((XNode - RandX) * Cos)+((YNode - RandY) *
                Sin(Angle))
                XNodeTSq := XNodeT * XNodeT
                * Transform nodes y-coord. using transformation equation
                YNodeT := ((YNode - RandY) * Cos)-((XNode - RandX) *
                Sin(Angle))
                YNodeTSq := YNodeT * YNodeT
                * Check if current node is inside hot spot
                if XNodeTSq/ASq + YNodeTSq/BSq <= 1
                    * If here, had a hit, increment hit counter
                    NumHits++
                    * Break from node checking sequence, do another trial
                    break
                endif
            next YNode
        next XNode
    end sequence
else
    * For triangular grids
    * Now sequence through all possible nodes, looking for a hit
    * Break out of sequence when a hit occurs
    RowsChecked := 0
    begin sequence
        for YNode = MinY to MaxY step SIN60
            * As we move up the grid from row to row, the X coordinates of
            the nodes alternate by 0.5.  On first pass, when RowsChecked
            is 0, MinX is not altered.  Thereafter, it alternates by 0.5.
            if nRowsChecked > 0
                if nRowsChecked % 2 == 0
                    MinX += 0.5
                else
                    MinX -= 0.5
                endif
            endif
            RowsChecked++
            for XNode = MinX to MaxX
                * Transform nodes x-coord. using transformation equation
                XNodeT := ((XNode - RandX) * Cos)+((YNode - RandY) *
                Sin(Angle))
                XNodeTSq := XNodeT * XNodeT
                * Transform nodes y-coord. using transformation equation
```

```
                    YNodeT := ((YNode - RandY) * Cos)-((XNode - nRandX) *
                    Sin(Angle))
                    YNodeTSq := YNodeT * YNodeT
                    * Check if current node is inside hot spot
                    if  XNodeTSq/ASq + YNodeTSq/BSq <= 1
                        NumHits++
                        * Break from node checking sequence, do another trial
                        break
                    endif
                next XNode
            next YNode
        end sequence
    endif
next Trial
    * Add current hit ratio to array, anProbHit
    aadd(anProbHit, NumHits/NumTrial)     // aadd() adds an element to an array
    * Add current hit ratio to summing variable, ProbHitSum
    ProbHitSum += anProbHit[Run]
    * Check if we have reached bound criteria
    if Run >= 10
        * Get current mean prob. of a hit
        MeanProb := ProbHitSum/Run
        * Get current sum of squared differences
        SumOfSqrs := 0
        for i = 1 to Run
            Diff := (anProbHit[i] - MeanProb)
            SumOfSqrs += Diff * Diff
        next i
        * Get current standard dev. of hit probabilities
        StdDev := sqrt(SumOfSqrs/(Run-1))
        * Get degrees of freedom for t score array
        DF := see code for details, generally Run - 1
        CurBound := anTScores[DF] * StdDev/sqrt(Run)
        * Check for bound convergence
        if nCurBound <= nDsrdBound .and. nRun >= 10
            * Done, no more runs needed
            exit
        endif
    endif
next nRun
```

Output: *MeanProb, StdDev, CurBound, TotTrials*

5. MONTE CARLO TESTS OF ELIPGRID-PC RESULTS

5.1 TEST OF SINGER'S 90 SINGLE-ANGLE CASES

As an example of the use of his ELIPGRID program, Singer provided 100 cases of input data and the ELIPGRID calculated probabilities (Singer 1972). These cases were used as the basis for testing the ELIPGRID-PC code (Davidson 1994). Since the Monte Carlo procedure is much more time consuming than the ELIPGRID algorithm, Singer's 10 random-angle cases were omitted from the Monte Carlo tests. However, many single-angle tests beyond Singer's 90 cases were also tested. The input test file, TestS90.SIF, was taken directly from the Test100.SIF file used for testing ELIPGRID-PC. Table A.1 in Appendix A is a listing of TestS90.SIF. Table A.2 in Appendix A is a listing of the output file, Test90.Out, produced by ELIPGRID-MC.

All Monte Carlo tests described here involved these four steps:

1. Input the test data.
2. Calculate the probability of not hitting the hot spot using ELIPGRID-PC code.
3. Calculate the probability of not hitting the hot spot using the Monte Carlo algorithm.
4. Print out both results, their difference, the Monte Carlo-calculated 99% confidence half-interval, and Y if the ELIPGRID-PC result was within the Monte Carlo 99% confidence interval. Print N if the ELIPGRID-PC result was *not* within the Monte Carlo 99% confidence interval.

The test results using Singer's 90 cases were excellent:

1. All 90 ELIPGRID-PC-calculated probabilities fell within the Monte Carlo-calculated 99% confidence interval.

2. The maximum absolute value of the difference in the percent probabilities of missing a hot spot was 0.470%.
3. The average absolute value of the difference in the percent probabilities of missing a hot spot was 0.105%.

See Fig. 6 for a histogram of the absolute values of the differences for 90 Singer cases.

5.2 TEST OF 150 RANDOM CASES

To further test the ELIPGRID-PC algorithm, 150 random cases were generated using the Minitab® random function. Fifty cases each for square, rectangular, and triangular grid were generated. The random values for *L/G* (semi-major axis ÷ grid size) came from a uniform distribution with a range of 0.01 to 3.0. The random values for the shape ratios (semi-minor axis ÷ semi-major axis) came from a uniform distribution with a range of 0.05 to 1.0. The random values for the hot-spot orientation angles came from a uniform distribution with a range of 0 to 45° for square grids, 0 to 90° for rectangular grids, and 0 to 30° for triangular grids. The rectangular grid long to short side ratios came from a log-normal distribution with a location value of 1.3 and a scale value 0.5. Table A.3 in Appendix A lists the input file, TestD150.SIF. Table A.4 (Appendix A) lists the output file, TestD150.Out.

The testing involved the same steps listed in Sect. 5.1. The results were excellent with one exception:

1. All 150 ELIPGRID-PC-calculated probabilities except one fell within the Monte Carlo-calculated 99% confidence interval.
2. The maximum absolute value of the difference in the percent probabilities of missing a hot spot was 0.460% with the one result excluded.

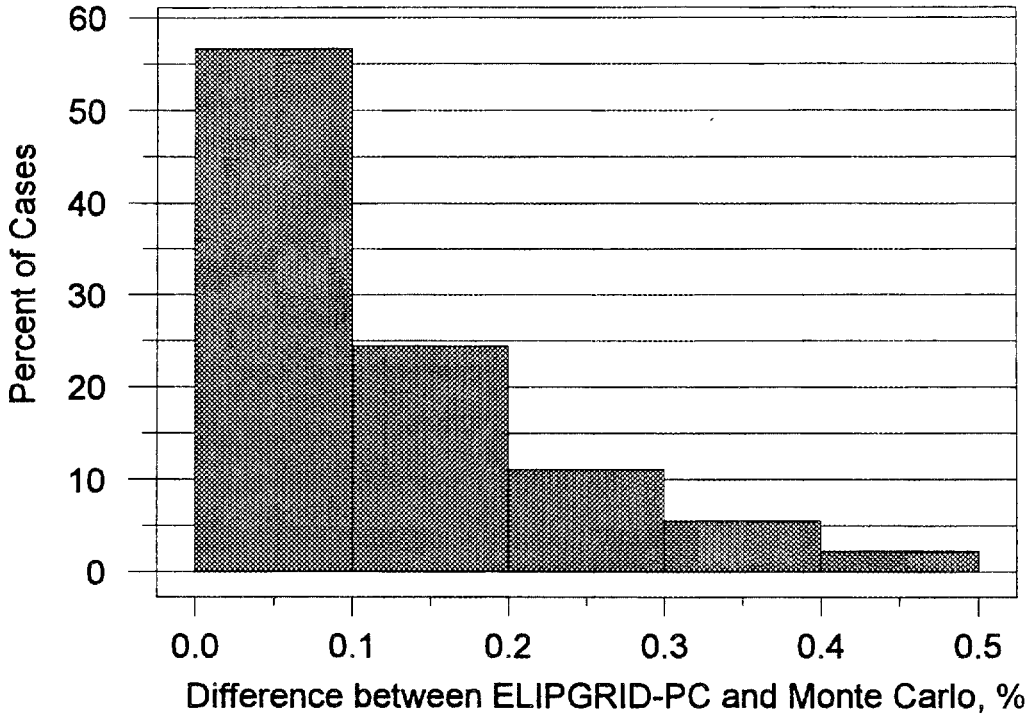

Fig. 6. Distribution of results for 90 Singer cases.

3. The average absolute value of the difference in the percent probabilities of missing a hot spot was 0.090%.

See Fig. 7 for a histogram of the absolute values of the differences for the 150 random cases.

The one exception can be found in Table A.4 as Target #86. It is for an ellipse with a shape ratio of 0.05. This is an extremely thin ellipse, on the minimum limit allowed by ELIPGRID-PC for shape ratios. The ELIPGRID-PC-calculated probability for this case differed from the Monte Carlo-calculated value by 1.01%. Although this is not within the goal of ± 0.5%, it may not represent a large difference from a practical stand-point. However, to ensure that the calculated probability for very thin ellipses is within ± 0.5% of the Monte Carlo result, ELIPGRID-MC should be used instead of ELIPGRID-PC. Further work with ELIPGRID-PC could probably reduce the difference to ± 0.5% for even very thin ellipses.

5.3 TESTS OF TRIANGULAR GRID CASES NEAR DISCONTINUITY

5.3.1 Test Using Original ELIPGRID-PC Regression Equation

The original ELIPGRID code exhibited a sharp discontinuity for triangular grids when L/G was between 0.85 and 1.0 (Davidson 1994, Appendix C). To deal with this problem, ELIPGRID-PC replaced the ELIPGRID algorithm with a series of fourth-order polynomial regressions based on the single independent variable L/G for triangular grids in the discontinuity region. The series consisted of regressions for seven different ranges of hot-spot shapes. ELIPGRID-MC was used to test the results of this regression technique. Table A.5 in Appendix A lists the input file, TestTO80.SIF. Table A.6 (Appendix A) lists the output file, TestTO80.Out.

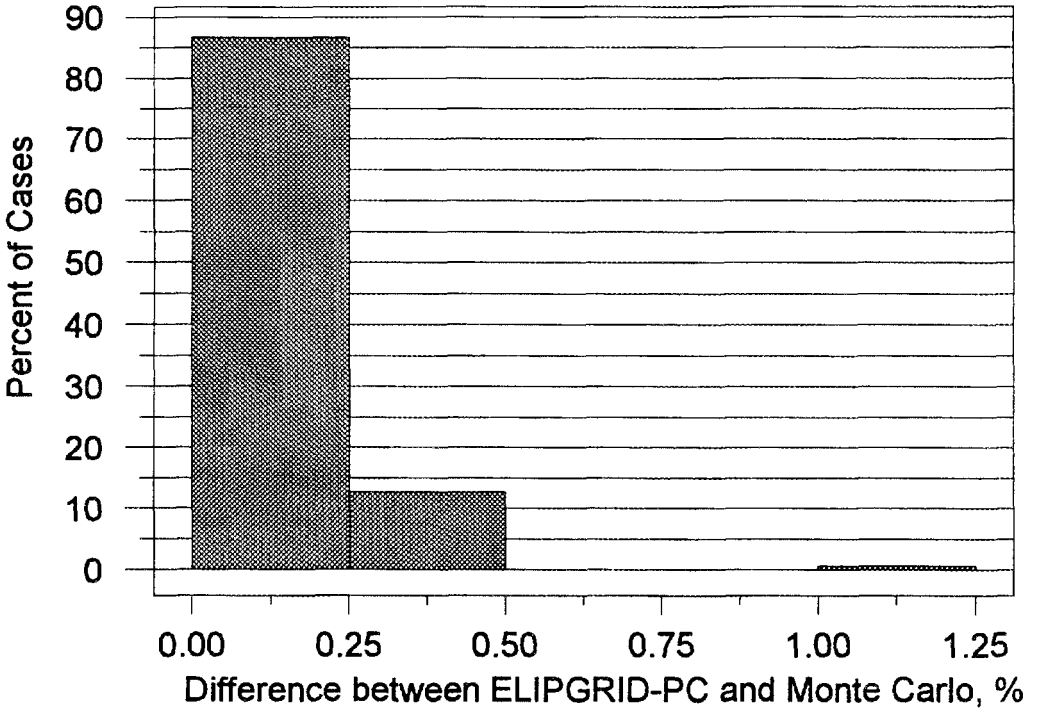

Fig. 7. Distribution of results for 150 random cases.

The testing involved the same steps listed in Sect. 5.1. The results were fair but not on a par with square and rectangular grid results:

1. Only 53 of the 80 ELIPGRID-PC-calculated probabilities fell within the Monte Carlo-calculated 99% confidence interval.
2. The maximum absolute value of the difference in the percent probabilities of missing a hot spot was 2.100%.
3. The average absolute value of the difference in the percent probabilities of missing a hot spot was 0.464%.
4. Twelve absolute values of the differences in percent probabilities of missing a hot spot were greater than 1.0%.

See Fig. 8 for a histogram of the absolute values of the differences for these 80 triangular grid cases.

5.3.2 The New Regression Equation

These results indicated the need for a better regression equation than the one used by ELIPGRID-PC. To meet this need, 500 random cases in the discontinuity region were generated using the Minitab® random uniform distribution function. The range of random *L/G* values was 0.49 to 0.60, and the range of random shapes was 0.84 to 0.99. (Random orientation angles were also tried; the angle independent variable did not contribute significantly to the regression and was removed.) The final fourth-order polynomial regression equation with two independent variables is

$$P(0) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_{11} x_1^2 + \beta_{22} x_2^2 + \beta_{12} x_1 x_2 + \beta_{111} x_1^3 + \beta_{112} x_1^2 x_2 + \beta_{122} x_1 x_2^2 + \beta_{1111} x_1^4 + \beta_{1122} x_1^2 x_2^2 + \beta_{1112} x_1^3 x_2$$

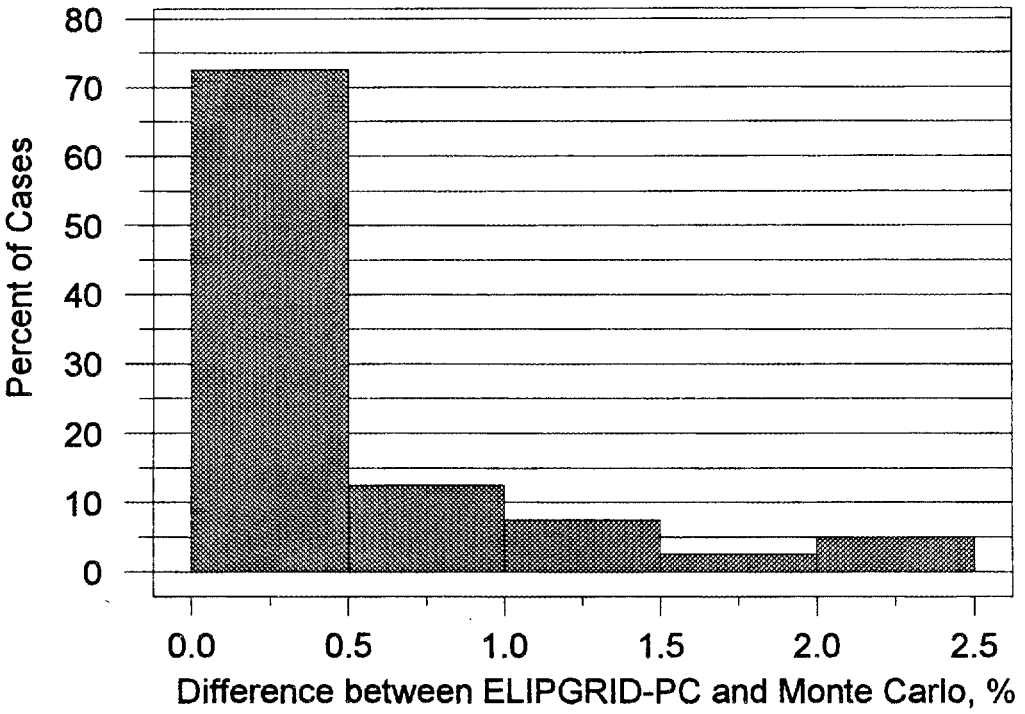

Fig. 8. Distribution of results for 80 triangular cases near discontinuity using original ELIPGRID-PC regression coefficients.

where

$P(0)$ = probability of missing the hot spot;

β = one of the 12 regression parameters. The subscripts enumerate for each independent variable the current terms factors;

x_1 = $X_1 - \bar{X}_1$, the deviation of the *L/G* ratio X_1 from its mean;

x_2 = $X_2 - \bar{X}_2$, the deviation of the hot-spot shape X_2 from its mean;

$\bar{X}_1$ = the mean *L/G* ratio for the 500 random cases;

$\bar{X}_2$ = the mean shape for the 500 random cases.

Note that each independent variable is expressed as a deviation from its mean. Expressing the independent variables this way reduces the problem of highly correlated higher power terms (Neter, Wasserman, and Kutner 1989). Estimated regression parameters were determined using Minitab® release 10.2 on 500 Monte Carlo-calculated probabilities from the triangular grid discontinuity region. The actual values and the means may be found in the code in function Prob0_Regr(), listed in file EGMCFort.Prg in Appendix B.

All fourth-order terms included in the equation were shown to be significantly different from 0.0 (all p-values were ≤ 0.001). The adjusted R^2 value, the coefficient of determination, was 0.99, and the estimated standard deviation about the regression line was 0.0015.

5.3.3 Test Using New Regression Equation

ELIPGRID-MC was used to test the results of using these new estimated parameters in the relevant ELIPGRID-PC code. The same 80 cases in or near the discontinuity region were used to test the new parameters. Table A.7 (Appendix A) lists the output file, TestT80.Out.

The testing involved the same steps listed in Sect. 5.1. The results were greatly improved over the previous regression results:

1. Seventy-seven of the 80 ELIPGRID-PC-calculated probabilities fell within the Monte Carlo-calculated 99% confidence interval.
2. The maximum absolute value of the difference in the percent probabilities of missing a hot spot was 0.480%.
3. The average absolute value of the difference in percent probabilities of missing a hot spot was 0.122%.
4. None of the absolute values of the differences in percent probabilities of missing a hot spot were greater than 0.5%.

See Fig. 9 for a histogram of the absolute values of the differences for these 80 triangular grid cases.

In conclusion, even the three cases that did not fall within the Monte Carlo-calculated 99% confidence interval had absolute differences of less than 0.5%. Therefore, the new regression solved the triangular grid discontinuity problem to within ± 0.5% of the true value. This should be satisfactory for most, if not all, practical problems involving triangular grids.

5.3.4 The New Regression Equation Code

The ELIPGRID-PC function Prob0_Regr() in file EGPCFort.Prg using the previous estimated regression parameters is now obsolete (Davidson 1994, Appendix E). The modified function with the new regression parameters is provided in Appendix B. It is designed to be a drop-in replacement for the original function.

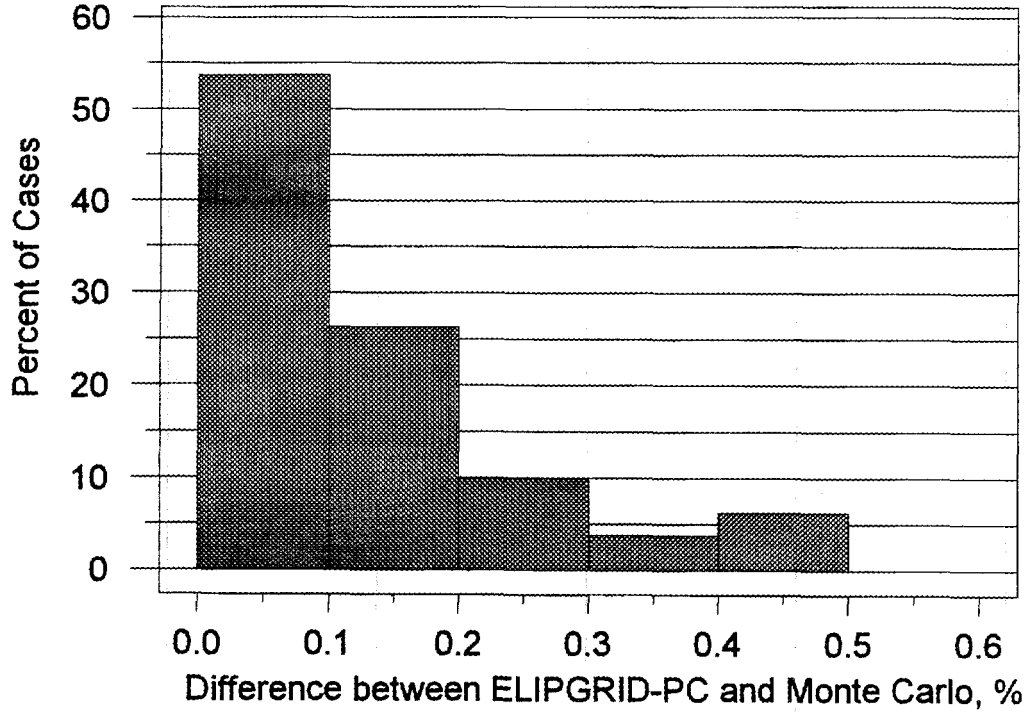

Fig. 9. Distribution of results for 80 triangular cases near discontinuity using new ELIPGRID-PC regression coefficients.

6. LIMITATIONS AND FURTHER WORK

The ELIPGRID-PC algorithm and the Monte Carlo tests are limited to a restricted range of elliptical hot-spot shapes and sizes. The elliptical shape range is from 0.05 to 1.0, i.e., thin ellipses to full circles. The ellipse size is limited by a maximum *L/G* ratio of 3.0, i.e., the ellipse semi-major axis can be no more than three times the grid cell size. These limitations, which grew out of the original ELIPGRID algorithm, will probably be of no consequence for most practical situations. For example, the nomographs based on ELIPGRID provided by Gilbert for environmental professionals are considerably more limited in ellipse size and shape (Gilbert 1987). However, it should noted that the Monte Carlo algorithm itself is not limited in these ways. The Monte Carlo code in ELIPGRID-MC could be isolated from the ELIPGRID-PC code, allowing it to calculate detection probabilities for very long and thin hot spots.

A limitation of the test data is related to the long-to-short-side ratio for rectangular grids. Theoretically, this ratio may be unlimited in size. Actually, for pragmatic testing reasons, the greatest ratio actually tested was 11.5. However, since rectangular grids with even a 2 to 1 long-to-short-side ratio are inefficient compared to square grids, it is unlikely that rectangular grids with ratios beyond 11.5 will be of practical concern. Also, note that neither the ELIPGRID-PC code nor the ELIPGRID-MC code is bound by this test data limit. Hence, rectangular grid cases could easily be further tested with very large long-to-short-side ratios.

The ELIPGRID-PC program not only includes a modified version of ELIPGRID but also provides options related to estimating the number of samples needed for a given area and the resulting sample cost. The Monte Carlo tests described here do not address this aspect of the ELIPGRID-PC program.

The full source code for ELIPGRID-PC is available in the CA-Clipper® language (Davidson 1994, Appendix E). However, CA-Clipper® is a proprietary, platform-specific language that is not easily portable to other computer platforms. In contrast, the

American National Standards Institute (ANSI) and the International Organization for Standardization (ISO) have adopted a C language standard that is not proprietary nor platform specific (Schildt 1990). The key algorithms in ELIPGRID-PC could be translated into the ANSI C language to provide portability benefits not available with CA-Clipper®.

7. SUMMARY

Singer's (1972) ELIPGRID algorithm as modified in the ELIPGRID-PC code has been successfully tested against an independent algorithm based on Monte Carlo simulation. The ELIPGRID-PC-calculated results for square grids have been shown to be within ± 0.5% of the Monte Carlo-calculated results for all cases tested. The ELIPGRID-PC-calculated results for rectangular grids have been shown to be within ± 0.5% of the Monte Carlo-calculated results for all but one (probably rare) test case. The ELIPGRID-PC-calculated results for triangular grids in the discontinuity region have been brought to within ± 0.5% of the Monte Carlo-calculated results through the use of a new fourth-order regression equation. All other triangular grid cases tested were within ± 0.5% of the Monte Carlo-calculated results.

The ELIPGRID algorithm, as modified by ELIPGRID-PC, has now been independently validated against a set of cases, including all 90 of Singer's (1972) single angle cases, 150 random cases, and 80 cases in the triangular grid discontinuity region. Furthermore, the Monte Carlo algorithm in ELIPGRID-MC is now available for further development and use on the difficult or exceptional cases that tend to arise in the "real" world.

REFERENCES

Computer Associates. 1992. *CA-Clipper® Programming and Utilities Guide*. Version 5.2. Computer Associates International.

Davidson, J. R. 1994. *ELIPGRID-PC: A PC Program for Calculating Hot Spot Probabilities*. ORNL/TM-12774. Oak Ridge National Laboratory, Oak Ridge, Tennessee.

Gilbert, R. O. 1987. *Statistical Methods for Environmental Pollution Monitoring*. Van Nostrand Reinhold, New York.

Law, A. M., and W. D. Kelton. 1982. *Simulation and Modeling Analysis*. McGraw-Hill, New York.

Meerschaert, M. M. 1993. *Mathematical Modeling*. Academic Press, Boston.

Neter, J., W. Wasserman, and M. H. Kutner. 1989. *Applied Linear Regression Models*. 2nd Edition. Irwin, Homewood, Illinois.

Park, S. K., and K. W. Miller. 1988. Random number generators: Good ones are hard to find. *Communications of the ACM*, **31**:1192-1201.

Schildt, H. 1990. *The Annotated ANSI C Standard*. Osborne McGraw-Hill, Berkeley, California.

Singer, D. A. 1972. ELIPGRID, a FORTRAN IV program for calculating the probability of success in locating elliptical targets with square, rectangular, and hexagonal grids. *Geocom Programs*, **4**:1-16.

Singer, D. A., and F. E. Wickman. 1969. *Probability Tables for Locating Elliptical Targets with Square, Rectangular and Hexagonal Point Nets*. Pennsylvania State University, University Park, Pennsylvania.

Spiegel, M. R. 1968. *Mathematical Handbook of Formulas and Tables*. McGraw-Hill, New York.

Yakowitz, S. J. 1977. *Computational Probability and Simulation*. Addison-Wesley, Reading, Massachusetts.

APPENDIX A

TEST DATA AND RESULTS

TEST DATA AND RESULTS

Table A.1. Input file listing of Singer's 90 cases

```
TestS90.SIF, an SIF format input test file for Monte Carlo testing of ELIPGRID.
* This file was taken directly from ELIPGRID-PC test file Test100.Sif, 2/6/94.
* The ten random angle cases are deleted leaving 30 sq, 30 rec, and 30 tri.
* These 90 cases are originally from Table 1 (Singer 1972) and were used in the
* ELIPGRID-PC TM to compare Singer's published results with ELIPGRID-PC, see
* Table B.7.
```

* Semimajor	Shape	Angle	GridSize	Type	Orient.	TargetID
1000.0	0.38	22.0	800.0	1	0	#261
1250.0	0.30	6.0	800.0	1	0	#187
1250.0	0.50	38.0	800.0	1	0	#190
300.0	0.25	24.0	800.0	1	0	#147
625.0	0.50	35.0	800.0	1	0	#10
875.0	0.31	7.0	800.0	1	0	#19
625.0	0.20	18.0	800.0	1	0	#26
125.0	0.50	24.0	800.0	1	0	#30
1625.0	0.15	11.0	800.0	1	0	#49
1250.0	0.50	0.0	800.0	1	0	#104
1000.0	0.38	22.0	1000.0	1	0	#261
1250.0	0.30	6.0	1000.0	1	0	#187
1250.0	0.50	38.0	1000.0	1	0	#190
300.0	0.25	24.0	1000.0	1	0	#147
625.0	0.50	35.0	1000.0	1	0	#10
875.0	0.31	7.0	1000.0	1	0	#19
625.0	0.20	18.0	1000.0	1	0	#26
125.0	0.50	24.0	1000.0	1	0	#30
1625.0	0.15	11.0	1000.0	1	0	#49
1250.0	0.50	0.0	1000.0	1	0	#104
1000.0	0.38	22.0	1500.0	1	0	#261
1250.0	0.30	6.0	1500.0	1	0	#187
1250.0	0.50	38.0	1500.0	1	0	#190
300.0	0.25	24.0	1500.0	1	0	#147
625.0	0.50	35.0	1500.0	1	0	#10
875.0	0.31	7.0	1500.0	1	0	#19
625.0	0.20	18.0	1500.0	1	0	#26
125.0	0.50	24.0	1500.0	1	0	#30
1625.0	0.15	11.0	1500.0	1	0	#49
1250.0	0.50	0.0	1500.0	1	0	#104
1000.0	0.38	22.0	859.66	2	0	#261
1250.0	0.30	6.0	859.66	2	0	#187
1250.0	0.50	22.0	859.66	2	0	#190

Table A.1. (continued)

625.0	0.50	35.0	565.69	3	0	#10
2.0						
875.0	0.31	7.0	565.69	3	0	#19
2.0						
625.0	0.20	18.0	565.69	3	0	#26
2.0						
125.0	0.50	24.0	565.69	3	0	#30
2.0						
1625.0	0.15	11.0	565.69	3	0	#49
2.0						
1250.0	0.50	0.0	565.69	3	0	#104
2.0						
1000.0	0.38	22.0	707.11	3	0	#261
2.0						
1250.0	0.30	6.0	707.11	3	0	#187
2.0						
1250.0	0.50	38.0	707.11	3	0	#190
2.0						
300.0	0.25	66.0	707.11	3	0	#147
2.0						
625.0	0.50	35.0	707.11	3	0	#10
2.0						
875.0	0.31	7.0	707.11	3	0	#19
2.0						
625.0	0.20	18.0	707.11	3	0	#26
2.0						
125.0	0.50	24.0	707.11	3	0	#30
2.0						
1625.0	0.15	11.0	707.11	3	0	#49
2.0						
1250.0	0.50	0.0	707.11	3	0	#104
2.0						
1000.0	0.38	22.0	1060.66	3	0	#261
2.0						
1250.0	0.30	6.0	1060.66	3	0	#187
2.0						
300.0	0.25	6.0	859.66	2	0	#147
625.0	0.50	25.0	859.66	2	0	#10
875.0	0.31	7.0	859.66	2	0	#19
625.0	0.20	18.0	859.66	2	0	#26
125.0	0.50	24.0	859.66	2	0	#30

Table A.1. (continued)

1625.0	0.15	11.0	859.66	2	0	#49
1250.0	0.50	0.0	859.66	2	0	#104
1000.0	0.38	22.0	1074.57	2	0	#261
1250.0	0.30	6.0	1074.57	2	0	#187
1250.0	0.50	22.0	1074.57	2	0	#190
300.0	0.25	6.0	1074.57	2	0	#147
625.0	0.50	25.0	1074.57	2	0	#10
875.0	0.31	7.0	1074.57	2	0	#19
625.0	0.2	18.0	1074.57	2	0	#26
125.0	0.50	24.0	1074.57	2	0	#30
1625.0	0.15	11.0	1074.57	2	0	#49
1250.0	0.50	0.0	1074.57	2	0	#104
1000.0	0.38	22.0	1611.86	2	0	#261
1250.0	0.30	6.0	1611.86	2	0	#187
1250.0	0.50	22.0	1611.86	2	0	#190
300.0	0.25	6.0	1611.86	2	0	#147
625.0	0.50	25.0	1611.86	2	0	#10
875.0	0.31	7.0	1611.86	2	0	#19
625.0	0.20	18.0	1611.86	2	0	#26
125.0	0.50	24.0	1611.86	2	0	#30
1625.0	0.15	11.0	1611.86	2	0	#49
1250.0	0.50	0.0	1611.86	2	0	#104
1000.0	0.38	22.0	565.69	3	0	#261
2.0						
1250.0	0.30	6.0	565.69	3	0	#187
2.0						
1250.0	0.50	38.0	565.69	3	0	#190
2.0						
300.0	0.25	66.0	565.69	3	0	#147
2.0						
1250.0	0.50	38.0	1060.66	3	0	#190
2.0						
300.0	0.25	66.0	1060.66	3	0	#147
2.0						
625.0	0.50	35.0	1060.66	3	0	#10
2.0						
875.0	0.31	7.0	1060.66	3	0	#19
2.0						
625.0	0.20	18.0	1060.66	3	0	#26
2.0						
125.0	0.50	24.0	1060.66	3	0	#30
2.0						

Table A.1. (continued)

1625.0	0.15	11.0	1060.66	3	0	#49
2.0						
1250.0	0.50	0.0	1060.66	3	0	#104
2.0						
9.9	9.9	9.9	9.9	9	9	EOF

Table A.2. Output file listing of Singer's 90 cases

Output from ORNL ELIPGRID-MC Monte Carlo Test Program Version: 11/14/94
File Name.: C:\CLIPPER2\EDITOR\EGMC\VALIDTST\TEST890.OUT
Created on: 11/14/94
Input file: TEST890.SIF using SIF format.
Title line: Test890.SIF, an SIF format input test file for Monte Carlo testing of ELIPGRID.

Target	Grid Type	Semi-major Axis in Relative Units	Gridspace in Orig Units	Shape	Angle	ELIPGRID Prob(0)	MCarlo Prob(0)	EG - MC Diff	MCarlo NumTrials	MC 99% ½-C.I.	EG in 99% C.I.
#261	Square	1.2500	800.00	0.38	22.0	0.0000	0.0000	0.0000	10,000	0.0000	Y
#187	Square	1.5625	800.00	0.30	6.0	0.0311	0.0329	-0.0018	11,000	0.0046	Y
#190	Square	1.5625	800.00	0.50	38.0	0.0000	0.0000	0.0000	10,000	0.0000	Y
#147	Square	0.3750	800.00	0.25	24.0	0.8896	0.8914	-0.0019	21,000	0.0050	Y
#10	Square	0.7813	800.00	0.50	35.0	0.0906	0.0911	-0.0004	14,000	0.0048	Y
#19	Square	1.0938	800.00	0.31	7.0	0.3059	0.3080	-0.0021	62,000	0.0050	Y
#26	Square	0.7813	800.00	0.20	18.0	0.6165	0.6189	-0.0024	78,000	0.0050	Y
#30	Square	0.1563	800.00	0.50	24.0	0.9617	0.9609	0.0007	12,000	0.0048	Y
#49	Square	2.0313	800.00	0.15	11.0	0.0470	0.0489	-0.0019	14,000	0.0046	Y
#104	Square	1.5625	800.00	0.50	0.0	0.0000	0.0000	0.0000	10,000	0.0000	Y
#261	Square	1.0000	1000.00	0.38	22.0	0.0619	0.0656	-0.0037	15,000	0.0049	Y
#187	Square	1.2500	1000.00	0.30	6.0	0.2337	0.2349	-0.0012	44,000	0.0049	Y
#190	Square	1.2500	1000.00	0.50	38.0	0.0000	0.0000	0.0000	10,000	0.0000	Y
#147	Square	0.3000	1000.00	0.25	24.0	0.9293	0.9283	0.0011	16,000	0.0050	Y
#10	Square	0.6250	1000.00	0.50	35.0	0.3864	0.3888	-0.0024	66,000	0.0050	Y
#19	Square	0.8750	1000.00	0.31	7.0	0.4591	0.4595	-0.0004	82,000	0.0050	Y
#26	Square	0.6250	1000.00	0.20	18.0	0.7546	0.7591	-0.0045	43,000	0.0049	Y
#30	Square	0.1250	1000.00	0.50	24.0	0.9755	0.9758	-0.0003	10,000	0.0035	Y
#49	Square	1.6250	1000.00	0.15	11.0	0.2499	0.2519	-0.0020	52,000	0.0050	Y
#104	Square	1.2500	1000.00	0.50	0.0	0.0000	0.0000	0.0000	10,000	0.0000	Y
#261	Square	0.6667	1500.00	0.38	22.0	0.4694	0.4708	-0.0014	73,000	0.0050	Y
#187	Square	0.8333	1500.00	0.30	6.0	0.5107	0.5093	0.0014	91,000	0.0049	Y
#190	Square	0.8333	1500.00	0.50	38.0	0.0266	0.0272	-0.0006	13,000	0.0046	Y
#147	Square	0.2000	1500.00	0.25	24.0	0.9686	0.9687	-0.0001	12,000	0.0049	Y
#10	Square	0.4167	1500.00	0.50	35.0	0.7273	0.7290	-0.0017	68,000	0.0050	Y
#19	Square	0.5833	1500.00	0.31	7.0	0.6783	0.6805	-0.0021	73,000	0.0050	Y
#26	Square	0.4167	1500.00	0.20	18.0	0.8909	0.8920	-0.0011	31,000	0.0050	Y
#30	Square	0.0833	1500.00	0.50	24.0	0.9891	0.9891	0.0000	10,000	0.0017	Y
#49	Square	1.0833	1500.00	0.15	11.0	0.5307	0.5311	-0.0004	85,000	0.0050	Y
#104	Square	0.8333	1500.00	0.50	0.0	0.2198	0.2208	-0.0010	56,000	0.0050	Y
#261	Triangular	1.1633	859.66	0.38	22.0	0.0000	0.0000	0.0000	10,000	0.0000	Y
#187	Triangular	1.4541	859.66	0.30	6.0	0.0000	0.0000	0.0000	10,000	0.0000	Y
#190	Triangular	1.4541	859.66	0.50	22.0	0.0000	0.0000	0.0000	10,000	0.0000	Y
#10	Rectangular, 2.0/1	1.1048	565.69	0.50	35.0	0.1518	0.1536	-0.0017	52,000	0.0049	Y
#19	Rectangular, 2.0/1	1.5468	565.69	0.31	7.0	0.0646	0.0669	-0.0023	12,000	0.0046	Y
#26	Rectangular, 2.0/1	1.1048	565.69	0.20	18.0	0.6165	0.6176	-0.0011	71,000	0.0050	Y
#30	Rectangular, 2.0/1	0.2210	565.69	0.50	24.0	0.9617	0.9609	0.0008	10,000	0.0048	Y
#49	Rectangular, 2.0/1	2.8726	565.69	0.15	11.0	0.0000	0.0000	0.0000	10,000	0.0000	Y
#104	Rectangular, 2.0/1	2.2097	565.69	0.50	0.0	0.0000	0.0000	0.0000	10,000	0.0000	Y
#261	Rectangular, 2.0/1	1.4142	707.11	0.38	22.0	0.0022	0.0018	0.0004	10,000	0.0011	Y
#187	Rectangular, 2.0/1	1.7678	707.11	0.30	6.0	0.0000	0.0000	0.0000	10,000	0.0000	Y
#190	Rectangular, 2.0/1	1.7678	707.11	0.50	38.0	0.0000	0.0000	0.0000	10,000	0.0000	Y
#147	Rectangular, 2.0/1	0.4243	707.11	0.25	66.0	0.9293	0.9340	-0.0047	15,000	0.0048	Y
#10	Rectangular, 2.0/1	0.8839	707.11	0.50	35.0	0.3882	0.3880	0.0002	72,000	0.0050	Y
#19	Rectangular, 2.0/1	1.2374	707.11	0.31	7.0	0.2989	0.3011	-0.0021	67,000	0.0049	Y
#26	Rectangular, 2.0/1	0.8839	707.11	0.20	18.0	0.7546	0.7545	0.0000	55,000	0.0049	Y

Table A.2. (continued)

#30	Rectangular,	2.0/1	0.1768	707.11	0.50	24.0	0.9755	0.9756	-0.0001	10,000	0.0041	Y
#49	Rectangular,	2.0/1	2.2981	707.11	0.15	11.0	0.0090	0.0094	-0.0004	10,000	0.0030	Y
#104	Rectangular,	2.0/1	1.7678	707.11	0.50	0.0	0.0000	0.0000	0.0000	10,000	0.0000	Y
#261	Rectangular,	2.0/1	0.9428	1060.66	0.38	22.0	0.4694	0.4724	-0.0030	84,000	0.0050	Y
#187	Rectangular,	2.0/1	1.1785	1060.66	0.30	6.0	0.3721	0.3737	-0.0016	82,000	0.0049	Y
#147	Triangular		0.3490	859.66	0.25	6.0	0.8896	0.8913	-0.0017	37,000	0.0050	Y
#10	Triangular		0.7270	859.66	0.50	25.0	0.1113	0.1122	-0.0010	32,000	0.0049	Y
#19	Triangular		1.0178	859.66	0.31	7.0	0.2594	0.2611	-0.0017	53,000	0.0050	Y
#26	Triangular		0.7270	859.66	0.20	18.0	0.6165	0.6167	-0.0002	61,000	0.0050	Y
#30	Triangular		0.1454	859.66	0.50	24.0	0.9617	0.9606	0.0010	11,000	0.0048	Y
#49	Triangular		1.8903	859.66	0.15	11.0	0.0023	0.0030	-0.0007	10,000	0.0017	Y
#104	Triangular		1.4541	859.66	0.50	0.0	0.0000	0.0000	0.0000	10,000	0.0000	Y
#261	Triangular		0.9306	1074.57	0.38	22.0	0.0134	0.0140	-0.0006	10,000	0.0046	Y
#187	Triangular		1.1633	1074.57	0.30	6.0	0.1807	0.1836	-0.0029	34,000	0.0049	Y
#190	Triangular		1.1633	1074.57	0.50	22.0	0.0000	0.0000	0.0000	10,000	0.0000	Y
#147	Triangular		0.2792	1074.57	0.25	6.0	0.9293	0.9294	-0.0001	24,000	0.0049	Y
#10	Triangular		0.5816	1074.57	0.50	25.0	0.3864	0.3898	-0.0034	76,000	0.0050	Y
#19	Triangular		0.8143	1074.57	0.31	7.0	0.4257	0.4250	0.0006	91,000	0.0050	Y
#26	Triangular		0.5816	1074.57	0.20	18.0	0.7546	0.7582	-0.0037	56,000	0.0050	Y
#30	Triangular		0.1163	1074.57	0.50	24.0	0.9755	0.9753	0.0002	10,000	0.0034	Y
#49	Triangular		1.5122	1074.57	0.15	11.0	0.1998	0.2022	-0.0025	46,000	0.0050	Y
#104	Triangular		1.1633	1074.57	0.50	0.0	0.0000	0.0000	0.0000	10,000	0.0000	Y
#261	Triangular		0.6204	1611.86	0.38	22.0	0.4694	0.4723	-0.0029	72,000	0.0050	Y
#187	Triangular		0.7755	1611.86	0.30	6.0	0.4810	0.4814	-0.0003	85,000	0.0050	Y
#190	Triangular		0.7755	1611.86	0.50	22.0	0.0677	0.0691	-0.0014	31,000	0.0049	Y
#147	Triangular		0.1861	1611.86	0.25	6.0	0.9686	0.9690	-0.0004	18,000	0.0049	Y
#10	Triangular		0.3878	1611.86	0.50	25.0	0.7273	0.7282	-0.0009	61,000	0.0050	Y
#19	Triangular		0.5429	1611.86	0.31	7.0	0.6695	0.6705	-0.0011	81,000	0.0050	Y
#26	Triangular		0.3878	1611.86	0.20	18.0	0.8909	0.8912	-0.0003	30,000	0.0049	Y
#30	Triangular		0.0776	1611.86	0.50	24.0	0.9891	0.9889	0.0002	10,000	0.0030	Y
#49	Triangular		1.0082	1611.86	0.15	11.0	0.5058	0.5054	0.0004	89,000	0.0050	Y
#104	Triangular		0.7755	1611.86	0.50	0.0	0.1712	0.1749	-0.0037	37,000	0.0050	Y
#261	Rectangular,	2.0/1	1.7678	565.69	0.38	22.0	0.0000	0.0000	0.0000	10,000	0.0000	Y
#187	Rectangular,	2.0/1	2.2097	565.69	0.30	6.0	0.0000	0.0000	0.0000	10,000	0.0000	Y
#190	Rectangular,	2.0/1	2.2097	565.69	0.50	38.0	0.0000	0.0000	0.0000	10,000	0.0000	Y
#147	Rectangular,	2.0/1	0.5303	565.69	0.25	66.0	0.8896	0.8906	-0.0010	19,000	0.0050	Y
#190	Rectangular,	2.0/1	1.1785	1060.66	0.50	38.0	0.0969	0.0977	-0.0008	32,000	0.0048	Y
#147	Rectangular,	2.0/1	0.2828	1060.66	0.25	66.0	0.9686	0.9690	-0.0004	10,000	0.0041	Y
#10	Rectangular,	2.0/1	0.5893	1060.66	0.50	35.0	0.7273	0.7281	-0.0008	70,000	0.0050	Y
#19	Rectangular,	2.0/1	0.8250	1060.66	0.31	7.0	0.6686	0.6689	-0.0003	85,000	0.0050	Y
#26	Rectangular,	2.0/1	0.5893	1060.66	0.20	18.0	0.8909	0.8917	-0.0008	24,000	0.0050	Y
#30	Rectangular,	2.0/1	0.1179	1060.66	0.50	24.0	0.9891	0.9898	-0.0007	10,000	0.0020	Y
#49	Rectangular,	2.0/1	1.5321	1060.66	0.15	11.0	0.4469	0.4487	-0.0017	66,000	0.0050	Y
#104	Rectangular,	2.0/1	1.1785	1060.66	0.50	0.0	0.0600	0.0614	-0.0014	26,000	0.0049	Y

END OF RUN (OR ERROR IN SHAPE OR L/G RATIO > 3)-

Table A.3 Input file listing of 150 random cases

TestD150.SIF input file, 11/14/94, 50 each Sq Rec Tri grid cases for ELIPGRID-MC.
* Random values generated by Minitab's RANDOM function.
* Random value distributions and ranges are:
* Semimajor/Grid, UNIFORM, 0.01 to 3.0
* Shape, UNIFORM, 0.05 to 1.0
* Angle, UNIFORM, Sq = 0 to 45°, Rec = 0 to 90°, Tri = 0 to 30°
* Rectangular grid long/short ratios, LOGNORMAL, location = 1.3, scale = 0.5

* Semimajor	Shape	Angle	GridSize	Type	Orient.	TargetID
76.3	0.36	2.5	100.0	1	0	# 1
269.0	0.61	1.8	100.0	1	0	# 2
8.0	0.93	23.6	100.0	1	0	# 3
231.1	0.65	14.1	100.0	1	0	# 4
270.6	0.11	23.9	100.0	1	0	# 5
283.4	0.65	32.5	100.0	1	0	# 6
96.1	0.40	25.5	100.0	1	0	# 7
14.5	0.58	6.3	100.0	1	0	# 8
288.8	0.27	11.2	100.0	1	0	# 9
50.3	0.20	0.7	100.0	1	0	# 10
145.5	0.98	3.3	100.0	1	0	# 11
178.5	0.13	22.0	100.0	1	0	# 12
282.8	0.11	14.9	100.0	1	0	# 13
60.9	0.76	6.5	100.0	1	0	# 14
75.2	0.90	22.7	100.0	1	0	# 15
179.3	0.83	24.7	100.0	1	0	# 16
132.7	0.07	36.2	100.0	1	0	# 17
207.8	0.58	6.7	100.0	1	0	# 18
153.9	0.85	2.0	100.0	1	0	# 19
145.3	0.57	21.5	100.0	1	0	# 20
87.6	0.10	18.4	100.0	1	0	# 21
73.3	0.93	31.1	100.0	1	0	# 22
215.6	0.49	42.4	100.0	1	0	# 23
30.5	0.92	4.3	100.0	1	0	# 24
122.1	0.12	16.7	100.0	1	0	# 25
205.4	0.51	23.1	100.0	1	0	# 26
181.9	0.27	16.6	100.0	1	0	# 27
123.3	0.62	39.7	100.0	1	0	# 28
165.3	0.37	7.3	100.0	1	0	# 29
148.3	0.89	7.1	100.0	1	0	# 30
88.8	0.42	24.7	100.0	1	0	# 31
100.1	0.82	25.7	100.0	1	0	# 32
222.1	0.88	30.9	100.0	1	0	# 33
207.7	0.27	1.1	100.0	1	0	# 34
57.1	0.08	17.2	100.0	1	0	# 35

Table A.3. (continued)

239.1	0.94	9.3	100.0	1	0	# 36
152.5	0.83	39.6	100.0	1	0	# 37
26.7	0.41	17.7	100.0	1	0	# 38
196.4	0.57	11.3	100.0	1	0	# 39
187.5	0.46	0.3	100.0	1	0	# 40
221.8	0.75	7.6	100.0	1	0	# 41
55.5	0.97	20.9	100.0	1	0	# 42
284.6	0.64	11.7	100.0	1	0	# 43
94.7	0.31	39.1	100.0	1	0	# 44
93.4	0.70	35.4	100.0	1	0	# 45
187.4	0.70	1.8	100.0	1	0	# 46
109.8	0.18	27.4	100.0	1	0	# 47
102.1	0.46	39.1	100.0	1	0	# 48
69.5	0.38	17.3	100.0	1	0	# 49
147.8	0.38	41.5	100.0	1	0	# 50
293.0	0.56	15.9	100.0	3	0	# 51
5.3191						
214.8	0.97	0.0	100.0	3	0	# 52
2.2443						
243.8	0.32	58.0	100.0	3	0	# 53
7.5820						
124.2	0.42	39.6	100.0	3	0	# 54
7.4843						
69.4	0.78	52.9	100.0	3	0	# 55
4.5560						
46.3	0.57	4.6	100.0	3	0	# 56
5.6421						
66.4	0.74	87.5	100.0	3	0	# 57
2.6114						
160.3	0.29	33.6	100.0	3	0	# 58
2.2016						
213.8	0.23	77.6	100.0	3	0	# 59
2.5606						
157.3	0.11	64.0	100.0	3	0	# 60
4.2246						
55.6	0.49	36.6	100.0	3	0	# 61
8.8497						
206.2	0.49	87.9	100.0	3	0	# 62
2.7193						
87.0	0.97	16.4	100.0	3	0	# 63
4.0294						
28.3	0.74	84.5	100.0	3	0	# 64
6.0057						

Table A.3. (continued)

283.1	0.09	24.1	100.0	3	0	# 65
2.4193						
173.6	0.45	61.1	100.0	3	0	# 66
5.7623						
3.7	0.75	47.2	100.0	3	0	# 67
1.2288						
57.3	0.15	22.9	100.0	3	0	# 68
4.1348						
44.5	0.94	24.2	100.0	3	0	# 69
6.1335						
225.0	0.21	83.8	100.0	3	0	# 70
5.6483						
140.9	0.06	75.2	100.0	3	0	# 71
3.7445						
202.9	0.41	38.7	100.0	3	0	# 72
3.6616						
130.7	0.58	24.1	100.0	3	0	# 73
1.7802						
61.3	0.80	52.2	100.0	3	0	# 74
6.4426						
113.2	0.56	30.3	100.0	3	0	# 75
1.3975						
278.4	0.46	16.3	100.0	3	0	# 76
1.7616						
153.4	0.35	28.9	100.0	3	0	# 77
1.4301						
268.4	0.73	85.9	100.0	3	0	# 78
1.7451						
253.0	0.54	17.8	100.0	3	0	# 79
8.8951						
127.4	0.43	60.8	100.0	3	0	# 80
3.3863						
290.0	0.35	2.0	100.0	3	0	# 81
5.3417						
277.1	0.40	24.7	100.0	3	0	# 82
4.4130						
86.5	0.30	25.5	100.0	3	0	# 83
1.5501						
156.1	0.76	21.7	100.0	3	0	# 84
7.0607						
164.6	0.05	52.2	100.0	3	0	# 85
11.5232						
238.0	0.05	42.6	100.0	3	0	# 86

Table A.3. (continued)

1.5970						
147.7	0.19	16.8	100.0	3	0	# 87
5.1976						
152.1	0.53	6.7	100.0	3	0	# 88
3.3424						
29.2	0.46	56.3	100.0	3	0	# 89
3.3647						
166.5	0.89	2.0	100.0	3	0	# 90
2.8041						
219.3	0.43	26.1	100.0	3	0	# 91
4.3854						
167.8	0.66	29.9	100.0	3	0	# 92
5.6132						
261.8	0.46	17.5	100.0	3	0	# 93
2.5743						
58.5	0.06	77.3	100.0	3	0	# 94
2.9013						
141.3	0.62	4.2	100.0	3	0	# 95
4.2476						
195.5	0.40	37.5	100.0	3	0	# 96
2.8911						
146.8	0.32	78.3	100.0	3	0	# 97
3.3060						
32.7	0.37	0.3	100.0	3	0	# 98
3.5939						
179.7	0.59	19.7	100.0	3	0	# 99
4.2808						
13.8	0.32	74.2	100.0	3	0	#100
1.9189						
81.6	0.08	17.3	100.0	2	0	#101
177.8	0.33	3.7	100.0	2	0	#102
91.9	0.12	10.8	100.0	2	0	#103
17.3	0.67	25.3	100.0	2	0	#104
198.4	0.92	28.4	100.0	2	0	#105
293.3	0.40	24.9	100.0	2	0	#106
224.0	0.48	14.6	100.0	2	0	#107
195.6	0.11	6.8	100.0	2	0	#108
147.5	0.45	26.9	100.0	2	0	#109
163.8	0.40	14.7	100.0	2	0	#110
32.6	0.58	13.9	100.0	2	0	#111
259.7	0.89	22.8	100.0	2	0	#112
227.1	0.18	23.7	100.0	2	0	#113
188.4	0.46	29.0	100.0	2	0	#114

Table A.3. (continued)

107.6	0.27	28.8	100.0	2	0	#115
7.9	0.73	2.1	100.0	2	0	#116
88.9	0.50	14.6	100.0	2	0	#117
55.7	0.61	6.0	100.0	2	0	#118
242.2	0.88	11.0	100.0	2	0	#119
188.2	0.65	7.2	100.0	2	0	#120
13.8	0.72	23.7	100.0	2	0	#121
111.1	0.47	24.6	100.0	2	0	#122
70.9	0.56	0.3	100.0	2	0	#123
286.9	0.15	9.3	100.0	2	0	#124
275.6	0.65	0.2	100.0	2	0	#125
59.5	0.75	14.0	100.0	2	0	#126
165.1	0.14	10.3	100.0	2	0	#127
76.0	0.84	8.9	100.0	2	0	#128
3.9	0.09	4.9	100.0	2	0	#129
187.5	0.36	26.2	100.0	2	0	#130
138.5	0.76	8.4	100.0	2	0	#131
161.9	0.18	18.8	100.0	2	0	#132
97.6	0.67	23.6	100.0	2	0	#133
274.1	0.08	10.4	100.0	2	0	#134
30.1	0.16	12.7	100.0	2	0	#135
219.6	0.89	9.2	100.0	2	0	#136
236.6	0.83	22.9	100.0	2	0	#137
181.6	0.26	21.3	100.0	2	0	#138
25.7	0.13	12.5	100.0	2	0	#139
20.9	0.69	20.2	100.0	2	0	#140
224.4	0.68	6.5	100.0	2	0	#141
283.6	0.46	10.0	100.0	2	0	#142
297.8	0.71	3.4	100.0	2	0	#143
148.3	0.27	2.3	100.0	2	0	#144
194.2	0.90	27.0	100.0	2	0	#145
40.7	0.92	8.9	100.0	2	0	#146
211.6	0.21	23.0	100.0	2	0	#147
28.0	0.77	10.1	100.0	2	0	#148
44.0	0.05	0.9	100.0	2	0	#149
58.1	0.39	10.3	100.0	2	0	#150

Table A.4. Output file listing of 150 random cases

Output from ORNL ELIPGRID-MC Monte Carlo Test Program Version: 11/14/94
File Name.: C:\JRD\TESTD150.OUT
Created on: 11/14/94
Input file: TESTD150.SIF using SIF format.
Title line: TestD150.SIF input file, 11/14/94, 50 each Sq Rec Tri grid cases for

Target	Grid Type	Semi-major Axis in Relative Units	Gridspace in Orig Units	Shape	Angle	ELIPGRID Prob(0)	MCarlo Prob(0)	EG - MC Diff	MCarlo NumTrials	MC 99% %-C.I.	EG in 99% C.I.
# 1	Square	0.7630	100.00	0.36	2.5	0.4904	0.4902	0.0002	85,000	0.0050	Y
# 2	Square	2.6900	100.00	0.61	1.8	0.0000	0.0000	0.0000	10,000	0.0000	Y
# 3	Square	0.0800	100.00	0.93	23.6	0.9813	0.9813	0.0000	10,000	0.0034	Y
# 4	Square	2.3110	100.00	0.65	14.1	0.0000	0.0000	0.0000	10,000	0.0000	Y
# 5	Square	2.7060	100.00	0.11	23.9	0.0000	0.0000	0.0000	10,000	0.0000	Y
# 6	Square	2.8340	100.00	0.65	32.5	0.0000	0.0000	0.0000	10,000	0.0000	Y
# 7	Square	0.9610	100.00	0.40	25.5	0.0535	0.0560	-0.0025	12,000	0.0048	Y
# 8	Square	0.1450	100.00	0.58	6.3	0.9617	0.9624	-0.0007	10,000	0.0041	Y
# 9	Square	2.8880	100.00	0.27	11.2	0.0000	0.0000	0.0000	10,000	0.0000	Y
# 10	Square	0.5030	100.00	0.20	0.7	0.8411	0.8448	-0.0037	52,000	0.0049	Y
# 11	Square	1.4550	100.00	0.98	3.3	0.0000	0.0000	0.0000	10,000	0.0000	Y
# 12	Square	1.7850	100.00	0.13	22.0	0.0005	0.0006	-0.0001	10,000	0.0007	Y
# 13	Square	2.8280	100.00	0.11	14.9	0.0000	0.0000	0.0000	10,000	0.0000	Y
# 14	Square	0.6090	100.00	0.76	6.5	0.1903	0.1920	-0.0017	45,000	0.0050	Y
# 15	Square	0.7520	100.00	0.90	22.7	0.0000	0.0000	0.0000	10,000	0.0000	Y
# 16	Square	1.7930	100.00	0.83	24.7	0.0000	0.0000	0.0000	10,000	0.0000	Y
# 17	Square	1.3270	100.00	0.07	36.2	0.6128	0.6119	0.0009	68,000	0.0050	Y
# 18	Square	2.0780	100.00	0.58	6.7	0.0000	0.0000	0.0000	10,000	0.0000	Y
# 19	Square	1.5390	100.00	0.85	2.0	0.0000	0.0000	0.0000	10,000	0.0000	Y
# 20	Square	1.4530	100.00	0.57	21.5	0.0000	0.0000	0.0000	10,000	0.0000	Y
# 21	Square	0.8760	100.00	0.10	18.4	0.7589	0.7613	-0.0024	39,000	0.0050	Y
# 22	Square	0.7330	100.00	0.93	31.1	0.0000	0.0000	0.0000	10,000	0.0000	Y
# 23	Square	2.1560	100.00	0.49	42.4	0.0000	0.0000	0.0000	10,000	0.0000	Y
# 24	Square	0.3050	100.00	0.92	4.3	0.7311	0.7327	-0.0015	78,000	0.0050	Y
# 25	Square	1.2210	100.00	0.12	16.7	0.4380	0.4384	-0.0004	90,000	0.0050	Y
# 26	Square	2.0540	100.00	0.51	23.1	0.0000	0.0000	0.0000	10,000	0.0000	Y
# 27	Square	1.8190	100.00	0.27	16.6	0.0000	0.0000	0.0000	10,000	0.0000	Y
# 28	Square	1.2330	100.00	0.62	39.7	0.0000	0.0000	0.0000	10,000	0.0000	Y
# 29	Square	1.6530	100.00	0.37	7.3	0.0000	0.0000	0.0000	10,000	0.0000	Y
# 30	Square	1.4830	100.00	0.89	7.1	0.0000	0.0000	0.0000	10,000	0.0000	Y
# 31	Square	0.8880	100.00	0.42	24.7	0.1026	0.1052	-0.0026	15,000	0.0050	Y
# 32	Square	1.0010	100.00	0.82	25.7	0.0000	0.0000	0.0000	10,000	0.0000	Y
# 33	Square	2.2210	100.00	0.88	30.9	0.0000	0.0000	0.0000	10,000	0.0000	Y
# 34	Square	2.0770	100.00	0.27	1.1	0.0000	0.0000	0.0000	10,000	0.0000	Y
# 35	Square	0.5710	100.00	0.08	17.2	0.9181	0.9188	-0.0007	20,000	0.0050	Y
# 36	Square	2.3910	100.00	0.94	9.3	0.0000	0.0000	0.0000	10,000	0.0000	Y
# 37	Square	1.5250	100.00	0.83	39.6	0.0000	0.0000	0.0000	10,000	0.0000	Y
# 38	Square	0.2670	100.00	0.41	17.7	0.9082	0.9089	-0.0007	24,000	0.0049	Y
# 39	Square	1.9640	100.00	0.57	11.3	0.0000	0.0000	0.0000	10,000	0.0000	Y
# 40	Square	1.8750	100.00	0.46	0.3	0.0000	0.0000	0.0000	10,000	0.0000	Y
# 41	Square	2.2180	100.00	0.75	7.6	0.0000	0.0000	0.0000	10,000	0.0000	Y
# 42	Square	0.5550	100.00	0.97	20.9	0.1169	0.1196	-0.0027	35,000	0.0049	Y
# 43	Square	2.8460	100.00	0.64	11.7	0.0000	0.0000	0.0000	10,000	0.0000	Y
# 44	Square	0.9470	100.00	0.31	39.1	0.2291	0.2312	-0.0021	59,000	0.0050	Y
# 45	Square	0.9340	100.00	0.70	35.4	0.0000	0.0000	0.0000	10,000	0.0000	Y
# 46	Square	1.8740	100.00	0.70	1.8	0.0000	0.0000	0.0000	10,000	0.0000	Y

Table A.4. (continued)

# 47	Square		1.0980	100.00	0.18	27.4	0.3182	0.3208	-0.0025	70,000	0.0050	Y
# 48	Square		1.0210	100.00	0.46	39.1	0.0000	0.0000	0.0000	10,000	0.0000	Y
# 49	Square		0.6950	100.00	0.38	17.3	0.4488	0.4497	-0.0009	80,000	0.0050	Y
# 50	Square		1.4780	100.00	0.38	41.5	0.0000	0.0000	0.0000	10,000	0.0000	Y
# 51	Rectangular,	5.3/1	2.9300	100.00	0.56	15.9	0.0000	0.0000	0.0000	10,000	0.0000	Y
# 52	Rectangular,	2.2/1	2.1480	100.00	0.97	0.0	0.0000	0.0000	0.0000	10,000	0.0000	Y
# 53	Rectangular,	7.6/1	2.4380	100.00	0.32	58.0	0.6266	0.6257	0.0009	77,000	0.0050	Y
# 54	Rectangular,	7.5/1	1.2420	100.00	0.42	39.6	0.7602	0.7579	0.0023	41,000	0.0049	Y
# 55	Rectangular,	4.6/1	0.6940	100.00	0.78	52.9	0.7678	0.7648	0.0031	45,000	0.0049	Y
# 56	Rectangular,	5.6/1	0.4630	100.00	0.57	4.6	0.9320	0.9322	-0.0002	23,000	0.0048	Y
# 57	Rectangular,	2.6/1	0.6640	100.00	0.74	87.5	0.6629	0.6611	0.0019	72,000	0.0049	Y
# 58	Rectangular,	2.2/1	1.6030	100.00	0.29	33.6	0.0402	0.0423	-0.0020	12,000	0.0047	Y
# 59	Rectangular,	2.6/1	2.1380	100.00	0.23	77.6	0.4898	0.4903	-0.0005	82,000	0.0050	Y
# 60	Rectangular,	4.2/1	1.5730	100.00	0.11	64.0	0.7976	0.7956	0.0020	45,000	0.0049	Y
# 61	Rectangular,	8.8/1	0.5560	100.00	0.49	36.6	0.9462	0.9479	-0.0017	20,000	0.0049	Y
# 62	Rectangular,	2.7/1	2.0620	100.00	0.49	87.9	0.2627	0.2650	-0.0023	65,000	0.0050	Y
# 63	Rectangular,	4.0/1	0.8700	100.00	0.97	16.4	0.5958	0.5939	0.0018	80,000	0.0050	Y
# 64	Rectangular,	6.0/1	0.2830	100.00	0.74	84.5	0.9690	0.9701	-0.0011	15,000	0.0048	Y
# 65	Rectangular,	2.4/1	2.8310	100.00	0.09	24.1	0.4451	0.4459	-0.0008	82,000	0.0050	Y
# 66	Rectangular,	5.8/1	1.7360	100.00	0.45	61.1	0.6345	0.6331	0.0014	78,000	0.0050	Y
# 67	Rectangular,	1.2/1	0.0370	100.00	0.75	47.2	0.9974	0.9970	0.0004	10,000	0.0007	Y
# 68	Rectangular,	4.1/1	0.5730	100.00	0.15	22.9	0.9626	0.9640	-0.0014	11,000	0.0049	Y
# 69	Rectangular,	6.1/1	0.4450	100.00	0.94	24.2	0.9047	0.9059	-0.0012	24,000	0.0050	Y
# 70	Rectangular,	5.6/1	2.2500	100.00	0.21	83.8	0.8147	0.8141	0.0006	34,000	0.0050	Y
# 71	Rectangular,	3.7/1	1.4090	100.00	0.06	75.2	0.9001	0.8996	0.0005	20,000	0.0049	Y
# 72	Rectangular,	3.7/1	2.0290	100.00	0.41	38.7	0.1281	0.1327	-0.0046	40,000	0.0049	Y
# 73	Rectangular,	1.8/1	1.3070	100.00	0.58	24.1	0.0000	0.0000	0.0000	10,000	0.0000	Y
# 74	Rectangular,	6.4/1	0.6130	100.00	0.80	52.2	0.8591	0.8610	-0.0019	26,000	0.0050	Y
# 75	Rectangular,	1.4/1	1.1320	100.00	0.56	30.3	0.0000	0.0000	0.0000	10,000	0.0000	Y
# 76	Rectangular,	1.8/1	2.7840	100.00	0.46	16.3	0.0000	0.0000	0.0000	10,000	0.0000	Y
# 77	Rectangular,	1.4/1	1.5340	100.00	0.35	28.9	0.0000	0.0000	0.0000	10,000	0.0000	Y
# 78	Rectangular,	1.7/1	2.6840	100.00	0.73	85.9	0.0000	0.0000	0.0000	10,000	0.0000	Y
# 79	Rectangular,	8.9/1	2.5300	100.00	0.54	17.8	0.4620	0.4633	-0.0013	82,000	0.0050	Y
# 80	Rectangular,	3.4/1	1.2740	100.00	0.43	60.8	0.5625	0.5623	0.0002	79,000	0.0050	Y
# 81	Rectangular,	5.3/1	2.9000	100.00	0.35	2.0	0.0026	0.0024	0.0002	10,000	0.0015	Y
# 82	Rectangular,	4.4/1	2.7710	100.00	0.40	24.7	0.0000	0.0000	0.0000	10,000	0.0000	Y
# 83	Rectangular,	1.6/1	0.8650	100.00	0.30	25.5	0.5451	0.5461	-0.0011	85,000	0.0050	Y
# 84	Rectangular,	7.1/1	1.5610	100.00	0.76	21.7	0.5831	0.5822	0.0009	79,000	0.0050	Y
# 85	Rectangular,	11.5/1	1.6460	100.00	0.05	52.2	0.9631	0.9662	-0.0031	16,000	0.0049	Y
# 86	Rectangular,	1.6/1	2.3800	100.00	0.05	42.6	0.4429	0.4529	-0.0101	54,000	0.0049	N
# 87	Rectangular,	5.2/1	1.4770	100.00	0.19	16.8	0.7495	0.7508	-0.0014	62,000	0.0050	Y
# 88	Rectangular,	3.3/1	1.5210	100.00	0.53	6.7	0.1557	0.1588	-0.0031	49,000	0.0049	Y
# 89	Rectangular,	3.4/1	0.2920	100.00	0.46	56.3	0.9634	0.9631	0.0003	10,000	0.0047	Y
# 90	Rectangular,	2.8/1	1.6650	100.00	0.89	2.0	0.0000	0.0000	0.0000	10,000	0.0000	Y
# 91	Rectangular,	4.4/1	2.1930	100.00	0.43	26.1	0.1198	0.1238	-0.0041	46,000	0.0050	Y
# 92	Rectangular,	5.6/1	1.6780	100.00	0.66	29.9	0.4623	0.4632	-0.0009	79,000	0.0050	Y
# 93	Rectangular,	2.6/1	2.6180	100.00	0.46	17.5	0.0000	0.0000	0.0000	10,000	0.0000	Y
# 94	Rectangular,	2.9/1	0.5850	100.00	0.06	77.3	0.9778	0.9788	-0.0010	10,000	0.0047	Y
# 95	Rectangular,	4.2/1	1.4130	100.00	0.62	4.2	0.3737	0.3767	-0.0029	80,000	0.0049	Y
# 96	Rectangular,	2.9/1	1.9550	100.00	0.40	37.5	0.0000	0.0000	0.0000	10,000	0.0000	Y
# 97	Rectangular,	3.3/1	1.4680	100.00	0.32	78.3	0.6775	0.6747	0.0028	62,000	0.0049	Y
# 98	Rectangular,	3.6/1	0.3270	100.00	0.37	0.3	0.9654	0.9651	0.0003	10,000	0.0048	Y
# 99	Rectangular,	4.3/1	1.7970	100.00	0.59	19.7	0.2208	0.2239	-0.0032	71,000	0.0050	Y
#100	Rectangular,	1.9/1	0.1380	100.00	0.32	74.2	0.9900	0.9895	0.0005	10,000	0.0032	Y
#101	Triangular		0.8160	100.00	0.08	17.3	0.8068	0.8049	0.0019	49,000	0.0050	Y
#102	Triangular		1.7780	100.00	0.33	3.7	0.0000	0.0000	0.0000	10,000	0.0000	Y

Table A.4. (continued)

#103	Triangular	0.9190	100.00	0.12	10.8	0.6324	0.6338	-0.0015	80,000	0.0050	Y
#104	Triangular	0.1730	100.00	0.67	25.3	0.9273	0.9282	-0.0009	22,000	0.0048	Y
#105	Triangular	1.9840	100.00	0.92	28.4	0.0000	0.0000	0.0000	10,000	0.0000	Y
#106	Triangular	2.9330	100.00	0.40	24.9	0.0000	0.0000	0.0000	10,000	0.0000	Y
#107	Triangular	2.2400	100.00	0.48	14.6	0.0000	0.0000	0.0000	10,000	0.0000	Y
#108	Triangular	1.9560	100.00	0.11	6.8	0.2897	0.2920	-0.0023	67,000	0.0050	Y
#109	Triangular	1.4750	100.00	0.45	26.9	0.0000	0.0000	0.0000	10,000	0.0000	Y
#110	Triangular	1.6380	100.00	0.40	14.7	0.0000	0.0000	0.0000	10,000	0.0000	Y
#111	Triangular	0.3260	100.00	0.58	13.9	0.7764	0.7767	-0.0003	59,000	0.0050	Y
#112	Triangular	2.5970	100.00	0.89	22.8	0.0000	0.0000	0.0000	10,000	0.0000	Y
#113	Triangular	2.2710	100.00	0.18	23.7	0.0000	0.0000	0.0000	10,000	0.0000	Y
#114	Triangular	1.8840	100.00	0.46	29.0	0.0000	0.0000	0.0000	10,000	0.0000	Y
#115	Triangular	1.0760	100.00	0.27	28.8	0.0113	0.0125	-0.0012	10,000	0.0041	Y
#116	Triangular	0.0790	100.00	0.73	2.1	0.9835	0.9838	-0.0003	10,000	0.0035	Y
#117	Triangular	0.8890	100.00	0.50	14.6	0.0141	0.0157	-0.0016	10,000	0.0038	Y
#118	Triangular	0.5570	100.00	0.61	6.0	0.3369	0.3405	-0.0036	75,000	0.0050	Y
#119	Triangular	2.4220	100.00	0.88	11.0	0.0000	0.0000	0.0000	10,000	0.0000	Y
#120	Triangular	1.8820	100.00	0.65	7.2	0.0000	0.0000	0.0000	10,000	0.0000	Y
#121	Triangular	0.1380	100.00	0.72	23.7	0.9503	0.9513	-0.0010	11,000	0.0049	Y
#122	Triangular	1.1110	100.00	0.47	24.6	0.0000	0.0000	0.0000	10,000	0.0000	Y
#123	Triangular	0.7090	100.00	0.56	0.3	0.1661	0.1704	-0.0043	34,000	0.0050	Y
#124	Triangular	2.8690	100.00	0.15	9.3	0.0000	0.0000	0.0000	10,000	0.0000	Y
#125	Triangular	2.7560	100.00	0.65	0.2	0.0000	0.0000	0.0000	10,000	0.0000	Y
#126	Triangular	0.5950	100.00	0.75	14.0	0.0969	0.1001	-0.0032	22,000	0.0048	Y
#127	Triangular	1.6510	100.00	0.14	10.3	0.1750	0.1778	-0.0029	42,000	0.0049	Y
#128	Triangular	0.7600	100.00	0.84	8.9	0.0000	0.0000	0.0000	10,000	0.0000	Y
#129	Triangular	0.0390	100.00	0.09	4.9	0.9995	0.9993	0.0002	10,000	0.0007	Y
#130	Triangular	1.8750	100.00	0.36	26.2	0.0000	0.0000	0.0000	10,000	0.0000	Y
#131	Triangular	1.3850	100.00	0.76	8.4	0.0000	0.0000	0.0000	10,000	0.0000	Y
#132	Triangular	1.6190	100.00	0.18	18.8	0.0000	0.0000	0.0000	10,000	0.0000	Y
#133	Triangular	0.9760	100.00	0.67	23.6	0.0000	0.0000	0.0000	10,000	0.0000	Y
#134	Triangular	2.7410	100.00	0.08	10.4	0.0000	0.0000	0.0000	10,000	0.0000	Y
#135	Triangular	0.3010	100.00	0.16	12.7	0.9474	0.9518	-0.0043	12,000	0.0049	Y
#136	Triangular	2.1960	100.00	0.89	9.2	0.0000	0.0000	0.0000	10,000	0.0000	Y
#137	Triangular	2.3660	100.00	0.83	22.9	0.0000	0.0000	0.0000	10,000	0.0000	Y
#138	Triangular	1.8160	100.00	0.26	21.3	0.0000	0.0000	0.0000	10,000	0.0000	Y
#139	Triangular	0.2570	100.00	0.13	12.5	0.9689	0.9705	-0.0016	12,000	0.0049	Y
#140	Triangular	0.2090	100.00	0.69	20.2	0.8907	0.8901	0.0006	30,000	0.0049	Y
#141	Triangular	2.2440	100.00	0.68	6.5	0.0000	0.0000	0.0000	10,000	0.0000	Y
#142	Triangular	2.8360	100.00	0.46	10.0	0.0000	0.0000	0.0000	10,000	0.0000	Y
#143	Triangular	2.9780	100.00	0.71	3.4	0.0000	0.0000	0.0000	10,000	0.0000	Y
#144	Triangular	1.4830	100.00	0.27	2.3	0.0843	0.0834	0.0008	16,000	0.0049	Y
#145	Triangular	1.9420	100.00	0.90	27.0	0.0000	0.0000	0.0000	10,000	0.0000	Y
#146	Triangular	0.4070	100.00	0.92	8.9	0.4472	0.4498	-0.0026	91,000	0.0050	Y
#147	Triangular	2.1160	100.00	0.21	23.0	0.0000	0.0000	0.0000	10,000	0.0000	Y
#148	Triangular	0.2800	100.00	0.77	10.1	0.7810	0.7805	0.0005	67,000	0.0049	Y
#149	Triangular	0.4400	100.00	0.05	0.9	0.9649	0.9669	-0.0020	12,000	0.0047	Y
#150	Triangular	0.5810	100.00	0.39	10.3	0.5320	0.5326	-0.0006	82,000	0.0050	Y

END OF RUN (OR ERROR IN SHAPE OR L/G RATIO > 3)-

Table A.5 Input file listing of 80 triangular grid cases

```
TestTL80.SIF, a triangular grid test file for Monte Carlo testing of ELIPGRID.
* ELIPGRID exhibited a discontinuity for triangular grids in the region where
* L/G = 0.5 to 0.6 and the shape ratio is >= 0.85 and < 1.0.  This file tests
* the old linear regression coefficients used in ELIPGRID-PC for this region.
* Note that all angles are 15° since the angle is not significant in determining
* the probability in this region as demonstrated by multiple regression.
* Semimajor    Shape     Angle    GridSize Type Orient. TargetID
     49.0       0.84      15.0      100.0   2    0       #T01
     49.0       0.86      15.0      100.0   2    0       #T02
     49.0       0.88      15.0      100.0   2    0       #T03
     49.0       0.90      15.0      100.0   2    0       #T04
     49.0       0.92      15.0      100.0   2    0       #T05
     49.0       0.94      15.0      100.0   2    0       #T06
     49.0       0.96      15.0      100.0   2    0       #T07
     49.0       0.98      15.0      100.0   2    0       #T08
     49.0       0.99      15.0      100.0   2    0       #T09
     49.0       1.00      15.0      100.0   2    0       #T10
     51.0       0.84      15.0      100.0   2    0       #T11
     51.0       0.86      15.0      100.0   2    0       #T12
     51.0       0.88      15.0      100.0   2    0       #T13
     51.0       0.90      15.0      100.0   2    0       #T14
     51.0       0.92      15.0      100.0   2    0       #T15
     51.0       0.94      15.0      100.0   2    0       #T16
     51.0       0.96      15.0      100.0   2    0       #T17
     51.0       0.98      15.0      100.0   2    0       #T18
     51.0       0.99      15.0      100.0   2    0       #T19
     51.0       1.00      15.0      100.0   2    0       #T20
     53.0       0.84      15.0      100.0   2    0       #T21
     53.0       0.86      15.0      100.0   2    0       #T22
     53.0       0.88      15.0      100.0   2    0       #T23
     53.0       0.90      15.0      100.0   2    0       #T24
     53.0       0.92      15.0      100.0   2    0       #T25
     53.0       0.94      15.0      100.0   2    0       #T26
     53.0       0.96      15.0      100.0   2    0       #T27
     53.0       0.98      15.0      100.0   2    0       #T28
     53.0       0.99      15.0      100.0   2    0       #T29
     53.0       1.00      15.0      100.0   2    0       #T30
     55.0       0.84      15.0      100.0   2    0       #T31
     55.0       0.86      15.0      100.0   2    0       #T32
     55.0       0.88      15.0      100.0   2    0       #T33
     55.0       0.90      15.0      100.0   2    0       #T34
     55.0       0.92      15.0      100.0   2    0       #T35
     55.0       0.94      15.0      100.0   2    0       #T36
```

Table A.5. (continued)

```
     55.0      0.96      15.0      100.0   2   0      #T37
     55.0      0.98      15.0      100.0   2   0      #T38
     55.0      0.99      15.0      100.0   2   0      #T39
     55.0      1.00      15.0      100.0   2   0      #T40
     57.0      0.84      15.0      100.0   2   0      #T41
     57.0      0.86      15.0      100.0   2   0      #T42
     57.0      0.88      15.0      100.0   2   0      #T43
     57.0      0.90      15.0      100.0   2   0      #T44
     57.0      0.92      15.0      100.0   2   0      #T45
     57.0      0.94      15.0      100.0   2   0      #T46
     57.0      0.96      15.0      100.0   2   0      #T47
     57.0      0.98      15.0      100.0   2   0      #T48
     57.0      0.99      15.0      100.0   2   0      #T49
     57.0      1.00      15.0      100.0   2   0      #T50
     59.0      0.84      15.0      100.0   2   0      #T51
     59.0      0.86      15.0      100.0   2   0      #T52
     59.0      0.88      15.0      100.0   2   0      #T53
     59.0      0.90      15.0      100.0   2   0      #T54
     59.0      0.92      15.0      100.0   2   0      #T55
     59.0      0.94      15.0      100.0   2   0      #T56
     59.0      0.96      15.0      100.0   2   0      #T57
     59.0      0.98      15.0      100.0   2   0      #T58
     59.0      0.99      15.0      100.0   2   0      #T59
     59.0      1.00      15.0      100.0   2   0      #T60
     60.0      0.84      15.0      100.0   2   0      #T61
     60.0      0.86      15.0      100.0   2   0      #T62
     60.0      0.88      15.0      100.0   2   0      #T63
     60.0      0.90      15.0      100.0   2   0      #T64
     60.0      0.92      15.0      100.0   2   0      #T65
     60.0      0.94      15.0      100.0   2   0      #T66
     60.0      0.96      15.0      100.0   2   0      #T67
     60.0      0.98      15.0      100.0   2   0      #T68
     60.0      0.99      15.0      100.0   2   0      #T69
     60.0      1.00      15.0      100.0   2   0      #T70
* Next cases are near #T13 where the largest diff. with ELIPGRID was found.
     50.0      0.87      15.0      100.0   2   0      #T71
     50.0      0.88      15.0      100.0   2   0      #T72
     50.5      0.88      15.0      100.0   2   0      #T73
     50.0      0.89      15.0      100.0   2   0      #T74
     51.0      0.87      15.0      100.0   2   0      #T75
     51.5      0.88      15.0      100.0   2   0      #T76
     51.0      0.89      15.0      100.0   2   0      #T77
     52.0      0.87      15.0      100.0   2   0      #T78
```

Table A.5. (continued)

52.0	0.88	15.0	100.0	2	0	#T79
52.0	0.89	15.0	100.0	2	0	#T80

Table A.6. Output file listing of 80 triangular grid cases tested with old regression

Output from OS9L ELIPGRID-MC Monte Carlo Test Program Version: 11/14/94
File Name.: C:\CLIPPER2\EDITOR\EGMC\VALIDTST\TESTTO80.OUT
Created on: 11/14/94
Input file: TESTTO80.SIF using SIF format.
Title line: TestTL80.SIF, a triangular grid test file for Monte Carlo testing of ELIPGRID.

Target	Grid Type	Semi-major Axis in Relative Units	Gridspace in Orig Units	Shape	Angle	ELIPGRID Prob(0)	MCarlo Prob(0)	EG - MC Diff	MCarlo NumTrials	MC 99% ½-C.I.	EG in 99% C.I.
#T01	Triangular	0.4900	100.00	0.84	15.0	0.2684	0.2699	-0.0016	68,000	0.0049	Y
#T02	Triangular	0.4900	100.00	0.86	15.0	0.2510	0.2525	-0.0015	63,000	0.0050	Y
#T03	Triangular	0.4900	100.00	0.88	15.0	0.2335	0.2353	-0.0018	64,000	0.0049	Y
#T04	Triangular	0.4900	100.00	0.90	15.0	0.2161	0.2171	-0.0009	57,000	0.0050	Y
#T05	Triangular	0.4900	100.00	0.92	15.0	0.1987	0.1996	-0.0009	58,000	0.0050	Y
#T06	Triangular	0.4900	100.00	0.94	15.0	0.1813	0.1816	-0.0003	55,000	0.0050	Y
#T07	Triangular	0.4900	100.00	0.96	15.0	0.1639	0.1670	-0.0032	41,000	0.0049	Y
#T08	Triangular	0.4900	100.00	0.98	15.0	0.1464	0.1496	-0.0031	36,000	0.0050	Y
#T09	Triangular	0.4900	100.00	0.99	15.0	0.1377	0.1419	-0.0042	32,000	0.0049	Y
#T10	Triangular	0.4900	100.00	1.00	15.0	0.1290	0.1331	-0.0040	32,000	0.0049	Y
#T11	Triangular	0.5100	100.00	0.84	15.0	0.2079	0.2099	-0.0021	44,000	0.0050	Y
#T12	Triangular	0.5100	100.00	0.86	15.0	0.1814	0.1922	-0.0108	35,000	0.0049	N
#T13	Triangular	0.5100	100.00	0.88	15.0	0.1537	0.1747	-0.0210	34,000	0.0049	N
#T14	Triangular	0.5100	100.00	0.90	15.0	0.1537	0.1566	-0.0029	33,000	0.0050	Y
#T15	Triangular	0.5100	100.00	0.92	15.0	0.1266	0.1393	-0.0128	29,000	0.0049	N
#T16	Triangular	0.5100	100.00	0.94	15.0	0.1090	0.1198	-0.0108	28,000	0.0049	N
#T17	Triangular	0.5100	100.00	0.96	15.0	0.0918	0.1015	-0.0097	32,000	0.0049	N
#T18	Triangular	0.5100	100.00	0.98	15.0	0.0751	0.0834	-0.0084	28,000	0.0049	N
#T19	Triangular	0.5100	100.00	0.99	15.0	0.0751	0.0745	0.0006	27,000	0.0049	Y
#T20	Triangular	0.5100	100.00	1.00	15.0	0.0658	0.0679	-0.0022	27,000	0.0050	Y
#T21	Triangular	0.5300	100.00	0.84	15.0	0.1533	0.1578	-0.0045	29,000	0.0050	Y
#T22	Triangular	0.5300	100.00	0.86	15.0	0.1262	0.1377	-0.0116	31,000	0.0049	N
#T23	Triangular	0.5300	100.00	0.88	15.0	0.1003	0.1192	-0.0190	29,000	0.0050	N
#T24	Triangular	0.5300	100.00	0.90	15.0	0.1003	0.0994	0.0009	25,000	0.0050	Y
#T25	Triangular	0.5300	100.00	0.92	15.0	0.0760	0.0828	-0.0068	20,000	0.0049	N
#T26	Triangular	0.5300	100.00	0.94	15.0	0.0610	0.0644	-0.0034	21,000	0.0049	Y
#T27	Triangular	0.5300	100.00	0.96	15.0	0.0465	0.0510	-0.0044	20,000	0.0050	Y
#T28	Triangular	0.5300	100.00	0.98	15.0	0.0327	0.0400	-0.0073	21,000	0.0049	N
#T29	Triangular	0.5300	100.00	0.99	15.0	0.0327	0.0353	-0.0026	21,000	0.0048	Y
#T30	Triangular	0.5300	100.00	1.00	15.0	0.0300	0.0316	-0.0016	19,000	0.0048	Y
#T31	Triangular	0.5500	100.00	0.84	15.0	0.1022	0.1069	-0.0047	27,000	0.0049	Y
#T32	Triangular	0.5500	100.00	0.86	15.0	0.0777	0.0873	-0.0097	26,000	0.0049	N
#T33	Triangular	0.5500	100.00	0.88	15.0	0.0558	0.0710	-0.0153	20,000	0.0050	N
#T34	Triangular	0.5500	100.00	0.90	15.0	0.0558	0.0524	0.0033	21,000	0.0048	Y
#T35	Triangular	0.5500	100.00	0.92	15.0	0.0370	0.0393	-0.0023	19,000	0.0048	Y
#T36	Triangular	0.5500	100.00	0.94	15.0	0.0260	0.0293	-0.0032	15,000	0.0047	Y
#T37	Triangular	0.5500	100.00	0.96	15.0	0.0159	0.0206	-0.0046	14,000	0.0049	Y
#T38	Triangular	0.5500	100.00	0.98	15.0	0.0065	0.0127	-0.0062	10,000	0.0049	N
#T39	Triangular	0.5500	100.00	0.99	15.0	0.0065	0.0106	-0.0041	10,000	0.0044	Y
#T40	Triangular	0.5500	100.00	1.00	15.0	0.0095	0.0086	0.0009	10,000	0.0038	Y
#T41	Triangular	0.5700	100.00	0.84	15.0	0.0609	0.0657	-0.0048	16,000	0.0048	N
#T42	Triangular	0.5700	100.00	0.86	15.0	0.0387	0.0465	-0.0078	15,000	0.0050	N
#T43	Triangular	0.5700	100.00	0.88	15.0	0.0228	0.0319	-0.0092	15,000	0.0050	N
#T44	Triangular	0.5700	100.00	0.90	15.0	0.0228	0.0225	0.0003	14,000	0.0048	Y
#T45	Triangular	0.5700	100.00	0.92	15.0	0.0110	0.0139	-0.0029	10,000	0.0044	Y
#T46	Triangular	0.5700	100.00	0.94	15.0	0.0050	0.0093	-0.0043	10,000	0.0045	Y

Table A.6. (continued)

#T47	Triangular	0.5700	100.00	0.96	15.0	0.0000	0.0045	-0.0045	10,000	0.0026	N
#T48	Triangular	0.5700	100.00	0.98	15.0	0.0000	0.0024	-0.0024	10,000	0.0011	N
#T49	Triangular	0.5700	100.00	0.99	15.0	0.0000	0.0018	-0.0018	10,000	0.0012	N
#T50	Triangular	0.5700	100.00	1.00	15.0	0.0007	0.0010	-0.0003	10,000	0.0008	Y
#T51	Triangular	0.5900	100.00	0.84	15.0	0.0277	0.0279	-0.0002	14,000	0.0049	Y
#T52	Triangular	0.5900	100.00	0.86	15.0	0.0123	0.0176	-0.0053	11,000	0.0047	N
#T53	Triangular	0.5900	100.00	0.88	15.0	0.0035	0.0097	-0.0062	10,000	0.0041	N
#T54	Triangular	0.5900	100.00	0.90	15.0	0.0035	0.0046	-0.0011	10,000	0.0039	Y
#T55	Triangular	0.5900	100.00	0.92	15.0	0.0000	0.0020	-0.0020	10,000	0.0016	N
#T56	Triangular	0.5900	100.00	0.94	15.0	0.0000	0.0004	-0.0004	10,000	0.0007	Y
#T57	Triangular	0.5900	100.00	0.96	15.0	0.0000	0.0000	0.0000	10,000	0.0000	Y
#T58	Triangular	0.5900	100.00	0.98	15.0	0.0000	0.0000	0.0000	10,000	0.0000	Y
#T59	Triangular	0.5900	100.00	0.99	15.0	0.0000	0.0000	0.0000	10,000	0.0000	Y
#T60	Triangular	0.5900	100.00	1.00	15.0	0.0000	0.0000	0.0000	10,000	0.0000	Y
#T61	Triangular	0.6000	100.00	0.84	15.0	0.0162	0.0171	-0.0009	10,000	0.0049	Y
#T62	Triangular	0.6000	100.00	0.86	15.0	0.0089	0.0091	-0.0002	10,000	0.0032	Y
#T63	Triangular	0.6000	100.00	0.88	15.0	0.0042	0.0036	0.0006	10,000	0.0028	Y
#T64	Triangular	0.6000	100.00	0.90	15.0	0.0014	0.0015	-0.0001	10,000	0.0014	Y
#T65	Triangular	0.6000	100.00	0.92	15.0	0.0001	0.0002	-0.0001	10,000	0.0004	Y
#T66	Triangular	0.6000	100.00	0.94	15.0	0.0000	0.0000	0.0000	10,000	0.0000	Y
#T67	Triangular	0.6000	100.00	0.96	15.0	0.0000	0.0000	0.0000	10,000	0.0000	Y
#T68	Triangular	0.6000	100.00	0.98	15.0	0.0000	0.0000	0.0000	10,000	0.0000	Y
#T69	Triangular	0.6000	100.00	0.99	15.0	0.0000	0.0000	0.0000	10,000	0.0000	Y
#T70	Triangular	0.6000	100.00	1.00	15.0	0.0000	0.0000	0.0000	10,000	0.0000	Y
#T71	Triangular	0.5000	100.00	0.87	15.0	0.2110	0.2140	-0.0030	49,000	0.0050	Y
#T72	Triangular	0.5000	100.00	0.88	15.0	0.2019	0.2044	-0.0025	49,000	0.0050	Y
#T73	Triangular	0.5050	100.00	0.88	15.0	0.1682	0.1890	-0.0208	36,000	0.0050	N
#T74	Triangular	0.5000	100.00	0.89	15.0	0.1929	0.1958	-0.0029	46,000	0.0050	Y
#T75	Triangular	0.5100	100.00	0.87	15.0	0.1814	0.1834	-0.0020	35,000	0.0049	Y
#T76	Triangular	0.5150	100.00	0.88	15.0	0.1397	0.1602	-0.0205	34,000	0.0049	N
#T77	Triangular	0.5100	100.00	0.89	15.0	0.1537	0.1652	-0.0115	35,000	0.0049	N
#T78	Triangular	0.5200	100.00	0.87	15.0	0.1531	0.1551	-0.0020	32,000	0.0050	Y
#T79	Triangular	0.5200	100.00	0.88	15.0	0.1260	0.1465	-0.0205	31,000	0.0049	N
#T80	Triangular	0.5200	100.00	0.89	15.0	0.1260	0.1371	-0.0111	29,000	0.0049	N

END OF RUN (OR ERROR IN SHAPE OR L/G RATIO > 3)-

Table A.7. Output file listing of 80 triangular grid cases tested with new regression

Output from ORNL ELIPGRID-MC Monte Carlo Test Program Version: 11/14/94
File Name.: C:\CLIPPER2\EDITOR\EGMC\VALIDTST\TESTT80.OUT
Created on: 11/14/94
Input file: TESTT80.SIF using SIF format.
Title line: TestT80.SIF, a triangular grid test file for Monte Carlo testing of ELIPGRID.

Target	Grid Type	Semi-major Axis in Relative Units	Gridspace in Orig Units	Shape	Angle	ELIPGRID Prob(0)	MCarlo Prob(0)	EG - MC Diff	MCarlo NumTrials	MC 99% ½-C.I.	EG in 99% C.I.
#T01	Triangular	0.4900	100.00	0.84	15.0	0.2684	0.2699	-0.0016	68,000	0.0049	Y
#T02	Triangular	0.4900	100.00	0.86	15.0	0.2510	0.2525	-0.0015	63,000	0.0050	Y
#T03	Triangular	0.4900	100.00	0.88	15.0	0.2335	0.2353	-0.0018	64,000	0.0049	Y
#T04	Triangular	0.4900	100.00	0.90	15.0	0.2161	0.2171	-0.0009	57,000	0.0050	Y
#T05	Triangular	0.4900	100.00	0.92	15.0	0.1987	0.1996	-0.0009	58,000	0.0050	Y
#T06	Triangular	0.4900	100.00	0.94	15.0	0.1813	0.1816	-0.0003	55,000	0.0050	Y
#T07	Triangular	0.4900	100.00	0.96	15.0	0.1639	0.1670	-0.0032	41,000	0.0049	Y
#T08	Triangular	0.4900	100.00	0.98	15.0	0.1464	0.1496	-0.0031	36,000	0.0050	Y
#T09	Triangular	0.4900	100.00	0.99	15.0	0.1377	0.1419	-0.0042	32,000	0.0049	Y
#T10	Triangular	0.4900	100.00	1.00	15.0	0.1290	0.1331	-0.0040	32,000	0.0049	Y
#T11	Triangular	0.5100	100.00	0.84	15.0	0.2079	0.2099	-0.0021	44,000	0.0050	Y
#T12	Triangular	0.5100	100.00	0.86	15.0	0.1937	0.1922	0.0015	35,000	0.0049	Y
#T13	Triangular	0.5100	100.00	0.88	15.0	0.1722	0.1747	-0.0025	34,000	0.0049	Y
#T14	Triangular	0.5100	100.00	0.90	15.0	0.1522	0.1566	-0.0045	33,000	0.0050	Y
#T15	Triangular	0.5100	100.00	0.92	15.0	0.1337	0.1393	-0.0056	29,000	0.0049	N
#T16	Triangular	0.5100	100.00	0.94	15.0	0.1168	0.1198	-0.0030	28,000	0.0049	Y
#T17	Triangular	0.5100	100.00	0.96	15.0	0.1015	0.1015	0.0000	32,000	0.0049	Y
#T18	Triangular	0.5100	100.00	0.98	15.0	0.0877	0.0834	0.0043	28,000	0.0049	Y
#T19	Triangular	0.5100	100.00	0.99	15.0	0.0814	0.0745	0.0069	27,000	0.0049	N
#T20	Triangular	0.5100	100.00	1.00	15.0	0.0658	0.0679	-0.0022	27,000	0.0050	Y
#T21	Triangular	0.5300	100.00	0.84	15.0	0.1533	0.1578	-0.0045	29,000	0.0050	Y
#T22	Triangular	0.5300	100.00	0.86	15.0	0.1318	0.1377	-0.0059	31,000	0.0049	N
#T23	Triangular	0.5300	100.00	0.88	15.0	0.1136	0.1192	-0.0056	29,000	0.0050	N
#T24	Triangular	0.5300	100.00	0.90	15.0	0.0970	0.0994	-0.0024	25,000	0.0050	Y
#T25	Triangular	0.5300	100.00	0.92	15.0	0.0818	0.0828	-0.0010	20,000	0.0049	Y
#T26	Triangular	0.5300	100.00	0.94	15.0	0.0683	0.0644	0.0039	21,000	0.0049	Y
#T27	Triangular	0.5300	100.00	0.96	15.0	0.0563	0.0510	0.0053	20,000	0.0050	N
#T28	Triangular	0.5300	100.00	0.98	15.0	0.0458	0.0400	0.0058	21,000	0.0049	N
#T29	Triangular	0.5300	100.00	0.99	15.0	0.0412	0.0353	0.0059	21,000	0.0048	N
#T30	Triangular	0.5300	100.00	1.00	15.0	0.0300	0.0316	-0.0016	19,000	0.0048	Y
#T31	Triangular	0.5500	100.00	0.84	15.0	0.1022	0.1069	-0.0047	27,000	0.0049	Y
#T32	Triangular	0.5500	100.00	0.86	15.0	0.0824	0.0873	-0.0049	26,000	0.0049	N
#T33	Triangular	0.5500	100.00	0.88	15.0	0.0675	0.0710	-0.0035	20,000	0.0050	Y
#T34	Triangular	0.5500	100.00	0.90	15.0	0.0542	0.0524	0.0018	21,000	0.0048	Y
#T35	Triangular	0.5500	100.00	0.92	15.0	0.0425	0.0393	0.0032	19,000	0.0048	Y
#T36	Triangular	0.5500	100.00	0.94	15.0	0.0322	0.0293	0.0030	15,000	0.0047	Y
#T37	Triangular	0.5500	100.00	0.96	15.0	0.0236	0.0206	0.0030	14,000	0.0049	Y
#T38	Triangular	0.5500	100.00	0.98	15.0	0.0165	0.0127	0.0038	10,000	0.0049	Y
#T39	Triangular	0.5500	100.00	0.99	15.0	0.0135	0.0106	0.0029	10,000	0.0044	Y
#T40	Triangular	0.5500	100.00	1.00	15.0	0.0095	0.0086	0.0009	10,000	0.0038	Y
#T41	Triangular	0.5700	100.00	0.84	15.0	0.0609	0.0657	-0.0048	16,000	0.0048	N
#T42	Triangular	0.5700	100.00	0.86	15.0	0.0455	0.0465	-0.0010	15,000	0.0050	Y
#T43	Triangular	0.5700	100.00	0.88	15.0	0.0340	0.0319	0.0020	15,000	0.0050	Y
#T44	Triangular	0.5700	100.00	0.90	15.0	0.0240	0.0225	0.0015	14,000	0.0048	Y
#T45	Triangular	0.5700	100.00	0.92	15.0	0.0156	0.0139	0.0017	10,000	0.0044	Y
#T46	Triangular	0.5700	100.00	0.94	15.0	0.0087	0.0093	-0.0006	10,000	0.0045	Y

Table A.7. (continued)

#T47	Triangular	0.5700	100.00	0.96	15.0	0.0034	0.0045	-0.0011	10,000	0.0026	Y
#T48	Triangular	0.5700	100.00	0.98	15.0	0.0000	0.0024	-0.0024	10,000	0.0011	N
#T49	Triangular	0.5700	100.00	0.99	15.0	0.0000	0.0018	-0.0018	10,000	0.0012	N
#T50	Triangular	0.5700	100.00	1.00	15.0	0.0007	0.0010	-0.0003	10,000	0.0008	Y
#T51	Triangular	0.5900	100.00	0.84	15.0	0.0277	0.0279	-0.0002	14,000	0.0049	Y
#T52	Triangular	0.5900	100.00	0.86	15.0	0.0211	0.0176	0.0034	11,000	0.0047	Y
#T53	Triangular	0.5900	100.00	0.88	15.0	0.0129	0.0097	0.0032	10,000	0.0041	Y
#T54	Triangular	0.5900	100.00	0.90	15.0	0.0062	0.0046	0.0016	10,000	0.0039	Y
#T55	Triangular	0.5900	100.00	0.92	15.0	0.0012	0.0020	-0.0008	10,000	0.0016	Y
#T56	Triangular	0.5900	100.00	0.94	15.0	0.0000	0.0004	-0.0004	10,000	0.0007	Y
#T57	Triangular	0.5900	100.00	0.96	15.0	0.0000	0.0000	0.0000	10,000	0.0000	Y
#T58	Triangular	0.5900	100.00	0.98	15.0	0.0000	0.0000	0.0000	10,000	0.0000	Y
#T59	Triangular	0.5900	100.00	0.99	15.0	0.0000	0.0000	0.0000	10,000	0.0000	Y
#T60	Triangular	0.5900	100.00	1.00	15.0	0.0000	0.0000	0.0000	10,000	0.0000	Y
#T61	Triangular	0.6000	100.00	0.84	15.0	0.0162	0.0171	-0.0009	10,000	0.0049	Y
#T62	Triangular	0.6000	100.00	0.86	15.0	0.0089	0.0091	-0.0002	10,000	0.0032	Y
#T63	Triangular	0.6000	100.00	0.88	15.0	0.0042	0.0036	0.0006	10,000	0.0028	Y
#T64	Triangular	0.6000	100.00	0.90	15.0	0.0014	0.0015	-0.0001	10,000	0.0014	Y
#T65	Triangular	0.6000	100.00	0.92	15.0	0.0001	0.0002	-0.0001	10,000	0.0004	Y
#T66	Triangular	0.6000	100.00	0.94	15.0	0.0000	0.0000	0.0000	10,000	0.0000	Y
#T67	Triangular	0.6000	100.00	0.96	15.0	0.0000	0.0000	0.0000	10,000	0.0000	Y
#T68	Triangular	0.6000	100.00	0.98	15.0	0.0000	0.0000	0.0000	10,000	0.0000	Y
#T69	Triangular	0.6000	100.00	0.99	15.0	0.0000	0.0000	0.0000	10,000	0.0000	Y
#T70	Triangular	0.6000	100.00	1.00	15.0	0.0000	0.0000	0.0000	10,000	0.0000	Y
#T71	Triangular	0.5000	100.00	0.87	15.0	0.2110	0.2140	-0.0030	49,000	0.0050	Y
#T72	Triangular	0.5000	100.00	0.88	15.0	0.2019	0.2044	-0.0025	49,000	0.0050	Y
#T73	Triangular	0.5050	100.00	0.88	15.0	0.1888	0.1890	-0.0002	36,000	0.0050	Y
#T74	Triangular	0.5000	100.00	0.89	15.0	0.1929	0.1958	-0.0029	46,000	0.0050	Y
#T75	Triangular	0.5100	100.00	0.87	15.0	0.1828	0.1834	-0.0007	35,000	0.0049	Y
#T76	Triangular	0.5150	100.00	0.88	15.0	0.1564	0.1602	-0.0038	34,000	0.0049	Y
#T77	Triangular	0.5100	100.00	0.89	15.0	0.1620	0.1652	-0.0032	35,000	0.0049	Y
#T78	Triangular	0.5200	100.00	0.87	15.0	0.1511	0.1551	-0.0040	32,000	0.0050	Y
#T79	Triangular	0.5200	100.00	0.88	15.0	0.1413	0.1465	-0.0052	31,000	0.0049	N
#T80	Triangular	0.5200	100.00	0.89	15.0	0.1320	0.1371	-0.0051	29,000	0.0049	N

END OF RUN (OR ERROR IN SHAPE OR L/G RATIO > 3)-

APPENDIX B

MODIFIED REGRESSION FUNCTION SOURCE CODE

APPENDIX B

MODIFIED REGRESSION FUNCTION SOURCE CODE

```
**********************************
Function Prob0_Regr(nLtoG, nShape)
* Determine prob. of missing, P(0), by using a 4th-order polynomial regression.
* New, 11/30/94, uses 4th-order multiple regression based on Monte Carlo values
* calculated from triangular grid discontinuity region.  Regression done with
* Minitab Release 10.2.  This regression was done on the deviations of L/G and
* Shape from their respective means.
* Input:    nLtoG    Semi-major axis to grid size ratio.
*           nShape   Semi-minor axis to semi-major axis ratio.
* Output:   Prob0    Prob. of missing target.
* Errors:   if nLtoG or nShape out of applicable range, returns 9.
local nRtnVal := 0
local nB0, nB1, nB2          // Regression coefficients
local nB11, nB22, nB12, nB112, nB122, nB1111, nB1122, nB111, nB1112

* The 4th-order multiple regression used these means to obtain the deviations
* from the mean that became the predictor variables.
* nLtoGMean  := 0.5447103              // L/G mean from 500 values used in reg.
* nShapeMean := 0.9159746              // Shape mean from 500 values
local nLtoGDev    := nLtoG  - 0.5447103 // L/G   deviation from its mean
local nShapeDev   := nShape - 0.9159746 // Shape deviation from its mean
local nLtoGDevSq  := nLtoGDev^2        // Deviations from mean squared
local nShapeDevSq := nShapeDev^2

* Check L/G ratio for correct range.
if nLtoG > 0.50 .and. nLtoG < 0.60
   * Use 4th-order multiple regression coeffs.
   * See Minitab worksheet C:\Clipper2\Editor\EGPC\Validtst\NewC500A.MTW
   * All coeffs. copied over from Minitab worksheet
   nB0    :=     0.05258302
   nB1    :=    -1.92219150
   nB2    :=    -0.70375878
   nB11   :=    18.34282112
   nB22   :=     2.59228277
   nB12   :=    10.74025822
   nB112  :=    86.09394836
   nB122  :=    26.11154175
   nB1111 :=  -464.66976929
   nB1122 :=  -725.48101807
   nB111  :=     9.29963398
```

```
   nB1112 := -1304.08276367
else
   * Error: nLtoG ratio out of range.
   nRtnVal := 9
endif
if nRtnVal != 9
   * Calculate 4th-order multi-regr. polynomial.
   nRtnVal := nB0 + nB1 *nLtoGDev + nB2 * nShapeDev + ;
      nB11   * nLtoGDevSq              + nB22   * nShapeDevSq + ;
      nB12   * nLtoGDev   * nShapeDev  + nB111  * nLtoGDev^3  + ;
      nB112  * nLtoGDevSq * nShapeDev  + nB122  * nLtoGDev   * nShapeDevSq+;
      nB1111 * nLtoGDev^4              + nB1122 * nLtoGDevSq * nShapeDevSq+;
      nB1112 * nLtoGDev^3 * nShapeDev
   * Round any neg. values up to 0.0.
   nRtnVal := iif(nRtnVal<0.0, 0.0, nRtnVal)
endif
return (nRtnVal)
*** End of Func: Prob0_Regr()
```

APPENDIX C

ELIPGRID-MC SOURCE CODE

APPENDIX C
ELIPGRID-MC SOURCE CODE

The first page of this appendix is an index of all user-defined functions in the program. The source code file name is listed for each function. The next two pages are example make and link files. Next are listed two ORNL-developed CA-Clipper® language header files. These have a .CH extension and are listed in alphabetic order. Following these are all the CA-Clipper® source code files. These files have a .PRG extension and are listed in alphabetic order after the main file named EGMCMAIN.PRG.

Program execution begins with function "Main". Execution can be traced from function to function from that point.

Index of all user-defined functions in the ELIPGRID-MC program.

All filenames have a .PRG extension.

Function Name	File Name
AlertBox()	EGMCFILE
BoxCenter()	EGMCMAIN
ChangeDrive()	EGMCMAIN
ChangeDrOrSub()	EGMCMAIN
ChangeSubdir()	EGMCMAIN
ChooseGrid()	EGMCMAIN
ChooseInput()	EGMCMAIN
DOS_Prompt()	EGMCMAIN
ElipGrid()	EGMCFORT
Err_MsgBox()	EGMCMAIN
ExtrctPath()	EGMCFILE
GetFileBox()	EGMCFILE
GetFilOutFile()	EGMCFILE
InitTScores()	EGMCMCAL
InputFromFile()	EGMCMAIN
Main()	EGMCMAIN
MC_ProbHit()	EGMCMCAL
MenuBox()	EGMCMAIN
MenuCenter()	EGMCMAIN
NotReadyYet()	EGMCMAIN
NumTrim()	EGMCMAIN
ParamHelp()	EGMCMAIN
Prob0_Regr()	EGMCFORT
Rect()	EGMCFORT
SayCenter()	EGMCMAIN
SeedUnifRand()	EGMCMCAL
SIF_FileInput()	EGMCFILE
Subdir()	EGMCFILE
TestUnifRand()	EGMCMCAL
UnifRand()	EGMCMCAL

```
// File....: EGMC.rmk
// Purpose.: Make file for EGMC program, ELIPGRID-PC Monte Carlo test program.
// Compiler: Clipper 5.2
// Author..: Jim Davidson
// Started.: 10/27/94
// Last Mod: 11/01/94
// Clipper compiler switches:
// /A = Automatic declaration of publics/privates as memvars.
// /B = Include debugging info., delete this switch for final exe.
// /I = #Include file search path.
// /N = No automatic main proc., must be used for file-wide var declarations.
// /O = Object file drive and or path.
// /Q = Quiet, suppress line number display.
// /T = Path for temp file creation
// /V = Treat all ambiguous var references as dynamic vars, not as fields.
// /W = Warn of ambiguous var references.

"e:\EGMCMain.OBJ": "C:\CLIPPER2\EDITOR\EGMC\EGMCMain.PRG"
   e:\Clipper C:\CLIPPER2\EDITOR\EGMC\EGMCMain /A/N/Q/V/W /Oe:\ /Te:\ /Ie:\

"e:\EGMCFile.OBJ": "C:\CLIPPER2\EDITOR\EGMC\EGMCFile.PRG"
   e:\Clipper C:\CLIPPER2\EDITOR\EGMC\EGMCFile /A/N/Q/V/W /Oe:\ /Te:\ /Ie:\

"e:\EGMCFort.OBJ": "C:\CLIPPER2\EDITOR\EGMC\EGMCFort.PRG"
   e:\Clipper C:\CLIPPER2\EDITOR\EGMC\EGMCFort /A/N/Q/V/W /Oe:\ /Te:\ /Ie:\

"e:\EGMCMCAl.OBJ": "C:\CLIPPER2\EDITOR\EGMC\EGMCMCAl.PRG"
   e:\Clipper C:\CLIPPER2\EDITOR\EGMC\EGMCMCAl /A/N/Q/V/W /Oe:\ /Te:\ /Ie:\

"e:\EGMC.EXE": "e:\EGMCMain.OBJ" "e:\EGMCFort.OBJ"  \
"e:\EGMCFile.OBJ" "e:\EGMCMCAl.OBJ"
   e:\blinker @C:\CLIPPER2\EDITOR\EGMC\EGMC.LNK
```

```
# File....: EGMC.Lnk
# Purpose.: Blinker 3.0 response file for ELIPGRID-PC Monte Carlo test program.
# Compiler: Clipper 5.2
# Author..: Jim Davidson
# Started.: 10/27/94
# Last Mod: 12/01/94
blinker incremental off
blinker message noblink
output e:\EGMC
file e:\EGMCMain
# Start of dynamic overlay area for Clipper code.
beginarea
   file e:\EGMCFile
   file e:\EGMCFort
   file e:\EGMCMCAl
endarea
lib e:\clipper
beginarea
   lib e:\extend
endarea
lib e:\terminal
lib e:\dbfntx
lib e:\ct                              # Clipper Tools library
#lib e:\cld                            # Clipper debugger library
```

```
//=====================================================================
// File....: Colors.Ch
// Purpose.: Provides color definitions for Clipper 5.x programs
// By......: Jim Davidson
// Started.: 07/24/91
// Last Mod: 12/28/94
//=====================================================================

/*** Below is example use in a program ***
public C_Normal   := C_WHT_BLU                // Normal screen colors
public C_HighLght := C_CYN_BLU                // Color Data highlight
public C_Help     := C_WHT_MAG                // Help screens
public C_Error    := C_WHT_RED                // Error screens
// later...
setcolor(m->C_Normal)
****/

#define C_BLK_WHT     "n/w,w+/n,,,w/n"        // Black on white

#define C_W_BLK       "w/n"                   // White on black
#define C_WHT_BLK     "w+/n,n/w"              // Bright white on black
#define C_W_BLU       "w/b,w+/n,,,gr+/n"      // White on blue
#define C_WHT_BLU     "w+/b,w+/n,,,gr+/n"     // Bright white on blue
#define C_WHT_BLKB    "w+*/n,n/w"             // Bright blinking white on black
#define C_WHT_RED     "w+/r,w+/b,,,gr+/n"     // Bright white on red
#define C_WHT_MAG     "w+/rb,gr+/n,,,bg+/n"   // Bright white on magenta
#define C_W_MAG       "w/rb"                  // White on magenta

#define C_CYN_BLK     "bg+/n"                 // Bright cyan on black
#define C_CYN_BLU     "bg+/b,gr+/n,,,bg+/n"   // Bright cyan on blue
#define C_CYN_BRN     "bg+/gr"                // Bright cyan on brown
#define C_CYN_MAG     "bg+/rb,gr+/n,,,bg+/n"  // Bright cyan on magenta

#define C_YEL_BLK     "gr+/n,gr+/n"           // Bright yellow on black
#define C_YEL_BLNK    "gr+*/n"                // Bright blinking yellow on black
#define C_YEL_BLU     "gr+/b,w+/n,,,bg+/n"    // Bright yellow on blue
#define C_YEL_MAG     "gr+/rb"                // Bright yellow on magenta

#define C_RED_BLK     "r+/n,w+/n"             // Bright red on black
#define C_GRN_BLK     "g+/n,w+/n,,,bg+/n"     // Bright green on black

* Colors for Flipper grf_colors() *
* D refers to #define origin
* Normal colors
#define DBLACK    0
#define DBLUE     1
#define DGREEN    2
#define DCYAN     3
#define DRED      4
#define DMAGENTA  5
#define DYELLOW   6
#define DWHITE    7
* Light, bright colors
#define DLGRAY    8
#define DLBLUE    9
#define DLGREEN   10
#define DLCYAN    11
#define DLRED     12
#define DLMAGENTA 13
#define DLYELLOW  14
#define DLWHITE   15
```

Code File: EGMCMain.Prg

```
//=============================================================================
// File.....: EGMCMain.Prg
// Program..: EGMC.Exe
// Purpose..: Main program file for ELIPGRID-MC Monte Carlo test program.
// Author...: Jim Davidson
// Started..: 10/27/94 from EGPC code, 09/06/94 version.
// Last Mod.: 12/19/94
//
// Files....: EGMCMain.Prg       This file, main module
//            EGMCFile.Prg       File input code
//            EGMCFort.Prg       Code translated from ELIPGRID FORTRAN
//            EGMCMCAl.Prg       Monte Carlo related functions
//
//            Functions are arranged in most files in alphabetical order,
//            although main() is the first function in this file.
//
// Notes....: Compiler = CA-Clipper 5.2
//            Linker   = BLINKER 3.0
//            Uses CA-Clipper Tools 3.0 library
//
//=============================================================================
// Version Info
#define VER_DATE "12/19/94"

// Include files.
// Clipper supplied include files.
#include "Directry.Ch"                  // File info definitions
#include "Inkey.Ch"                     // Key definitions
#include "Set.Ch"                       // set() definitions
#include "Setcurs.Ch"                   // setcursor() related
#include "Box.Ch"                       // Box drawing constants

// ORNL developed include file.
#include "Colors.Ch"                    // Color definitions
// Below for rand num test
#define FC_NORMAL  0                    // Create normal read/write file
#define CR_LF chr(13) + chr(10)         // End of line

// User-defined command
#xcommand DEFAULT <TheParam> TO <DefaultVal> => ;
   IF (<TheParam> == NIL); <TheParam>:=<DefaultVal>; ENDIF

*==============================| Main Module |==============================
Function Main()
* Main module of program.

* Initialize local variables.
local nCh             := 1              // Main menu choice
local nLCol           := 0              // Left col chosen by MenuCenter()
local nTRow           := 8              // Main Menu top screen row
local cDOSScreen      := savescreen(0,0,24,79)
local nDOSRow         := row()
local lDone           := .F.            // Main Menu loop flag
local cDOSCmdLine     := ""             // DOS command line params
local nHandle         := 0              // File handle for rand num test

* Program wide publics.
public lOLR           := .F.            // Old linear regression flag
* The ElipGrid correction levels below provide ability to test old algorithms.
* Correction level 3 is only level used in EGMC.
public nElpGrdCor     := 3              // ElipGrid correction level
public C_Normal       := C_WHT_BLU      // Normal screen colors
public C_HighLght     := C_CYN_BLU      // Current subdir color
public C_Help         := C_WHT_MAG      // Help screens
public C_Error        := C_WHT_RED      // Error screens
```

Code File: EGMCMain.Prg

```
public C_Menu1        := C_WHT_MAG      // Menu screen color 1
public C_Menu2        := C_YEL_MAG      // Menu screen color 2

* Get DOS command line parameters.
cDOSCmdLine           := upper(dosparam())
if "H" $ cDOSCmdLine
   * Help param. passed.
   ParamHelp(VER_DATE)
   quit
elseif "OLR" $ cDOSCmdLine
   * Use "Old Linear Regression" (OLR) for tri. grid discontinuity.
   * Note that default is 2nd order multiple regression.
   m->lOLR  := .T.
endif

set escape on
set scoreboard off
set bell off
set confirm on
set wrap on

* Test random number generator.
setcolor(C_Normal)
* Test random number gen. if check file does not exist.
if ! file("RandTest.OK")
   cls
   ?? repli("=",80)
   ?? " Testing random number generator on this computer's floating point"
   ? " arithmetic."
   ? repli("=",80)
   @row()-2,12                              // Move cursor after "arithmetic."
   if TestUnifRand() == .T.
      if (nHandle := fcreate("RandTest.OK", FC_NORMAL)) == -1
         * Can't create dummy check file.  Will have to run test each time.
      else
         fwrite(nHandle, "File..: RandTest.OK" + CR_LF)
         fwrite(nHandle, "Result: Rand gen OK"+ CR_LF)
         fwrite(nHandle, "Date..: " + dtoc(date()))
         fclose(nHandle)
      endif
   else
      * Rand number gen test failed
      ? "Random number generator test failed on this machine."
      ? "Press a key to quit."
      quit
   endif
endif

do while ! lDone
   setcolor(m->C_Normal)                    // Reset since looping back
   cls
   dispbox(00,00,04,79, B_DOUBLE_SINGLE)
   setcolor(m->C_Help)
   SayCenter(1, " ORNL ELIPGRID-MC Monte Carlo Test Program ")
   setcolor(m->C_Normal)
   SayCenter(2,"Compares ELIPGRID-PC algorithm with Monte Carlo results.")
   SayCenter(3,"Version " + VER_DATE)
   @05, 2 say "Current subdirectory: "
   @row(),col() say  diskname() + ":" + dirname() color (m->C_HighLght)
   setcolor(m->C_Help)
   SayCenter(23, " Esc key to Exit ")
   setcolor(m->C_Normal)
   dispbox(06,00,22,79, B_DOUBLE_SINGLE)
   dispbox(22,00,24,79, B_DOUBLE_SINGLE)
   @22,00 say "├"
   @22,79 say "┤"
```

Code File: EGMCMain.Prg

```
   SayCenter(nTRow-1,"Main Menu")

   nCh := MenuCenter(nTRow,  {"P  Probability of Hitting Hot Spot"    ,;
                              "N  New Drive or Subdirectory"         ,;
                              "D  DOS Prompt"                        ,;
                              "Q  Quit program..."}, nCh,1, @nLCol)
   do case
      case nCh == 1
         * P  Probability of Hitting Hot Spot
         ChooseInput(nTRow+nCh+1,nLCol+2)
      case nCh == 2
         * N  New Drive or Subdirectory
         ChangeDrOrSub(nTRow+nCh+1,nLCol+2)
      case nCh == 3
         * D DOS Prompt
         DOS_Prompt()
      otherwise
         * Q  Quit program... (or Esc key)
         lDone := .T.
         loop
   endcase
enddo

set color to
restscreen(0,0,24,79,cDOSScreen)
devpos(nDOSRow-1,0)                    // -1 makes DOS prompt come in just
* return to DOS                        //  below last prompt.
return (0)                             // Return 0 to DOS ErrorLabel
*** End of Func: Main()
*=========================| End of Main Module |================================

*--- Begin Other Functions ---*
*********************************************
Function BoxCenter(nTRow, nRows, nWidth, nType)
* Displays box centered on nRow.
* nType of 1 = double line top, single side, 2 = double all.
* Returns left column.
local nCol := (80-nWidth)/2
default nType to 1
if nType == 1
   MenuBox(nTRow,nCol,nTRow+nRows,nCol+nWidth-1, B_DOUBLE_SINGLE)
else
   MenuBox(nTRow,nCol,nTRow+nRows,nCol+nWidth-1,B_DOUBLE)
endif
return (nCol)
*** End of Func: BoxCenter()

**********************
Function ChangeDrive()
* Change current drive.
local cCurrDrive
local lOrgConfrm
local lDone       := .F.
local GetList     := {}
lOrgConfrm        := set(_SET_CONFIRM,.F.)

do while ! lDone
   cls
   cCurrDrive    := diskname()
   MenuBox(02,01,7,67)
   @03,02 say " Change current drive to?"
   @05,03 say "Enter new drive letter" get cCurrDrive pict "!"
   @05,col() say ":"
   read

   if ! diskchange(cCurrDrive)
```

Code File: EGMCMain.Prg

```
      * Invalid drive.
      Err_MsgBox(10,"E","Error: Invalid drive.", ;
                "Drive: " + cCurrDrive)
      loop
   else
      lDone := .T.
      loop
   endif
enddo
set(_SET_CONFIRM,lOrgConfrm)
return (NIL)
*** End of Func: ChangeDrive()

*******************************
Function ChangeDrOrSub(nTR,nLC)
* Menu for changing drive or subir.
* Input: nTR is top row.
*        nLC is left col.
*
static nLastCh    := 1                 // Remembers last choice
local nBR         := nTR +  4
local nRC         := nLC + 32
* Change drive.
ChangeDrive()
if lastkey() != K_ESC
   * Change subdirectory.
   ChangeSubdir(10)
endif
return (NIL)
*** End of Func: ChangeDrOrSub()

**************************
Function ChangeSubdir(nTR)
* Changes current subdir.
local cCurrSubdir := ""
local cCurrDrive  := ""
local cDOSCmnd    := ""
local lDone       := .F.
local GetList     := {}

cCurrDrive        := diskname()
cCurrSubdir       := dirname()

do while ! lDone
   cCurrSubdir    := padr(cCurrSubdir, 64)
   MenuBox(nTR,1,nTR+6,66)
   @nTR+1,03 say "Change current subdirectory.  Must be on drive " + ;
      diskname() + ":..."
   @nTR+3,03 say "Change to " + cCurrDrive + ":"
   @nTR+3,col() get cCurrSubdir pict "@S50!"
   @nTR+5,03 say "Current path: " + diskname() + ":" + dirname()
   keyboard chr(K_END)
   read
   cCurrSubdir := alltrim(cCurrSubdir)

   if lastkey() == K_ESC
      * Esc key abort.
      lDone := .T.
      loop
   elseif ":" $ cCurrSubdir
      * Error, drive name entered.
      Err_MsgBox(nTR+6,"E","Error: Drive name entered.", ;
                "Note.: This option only changes subdirectories" + ;
                " on current drive.", ;
                "       Use change drive option for new drive.")
      loop
```

Code File: EGMCMain.Prg

```
   elseif ! subdir(cCurrSubdir)
      * Error, invalid subdirectory.
      Err_MsgBox(nTR+6,"E","Error: Invalid subdirectory.", ;
                 "Path.: " + cCurrDrive + ":" + cCurrSubdir )
      loop
   else
      * Do the DOS CD command.
      if len(cCurrSubdir) > 3 .and. right(cCurrSubdir,1) == "\"
         * If not root subdir and we have trailing "\", remove it.
         * Will mess up DOS CD command.
         cCurrSubdir := left(cCurrSubdir,len(cCurrSubdir)-1)
      endif
      * If no characters in subdir name, default to root of current drive.
      if empty(cCurrSubdir)
         cCurrSubdir := "\"
      endif

      * Form the command and do the work.
      cDOSCmnd := "CD " + cCurrSubdir
      run (cDOSCmnd)

      lDone := .T.
      loop
   endif
enddo
return (NIL)
*** End of Func: ChangeSubdir()

************************************
Function ChooseGrid(nTR,nLC, cColor)
* Choose grid type desired.
* Input:    Top row, left col., menu color.
* Returns:  Grid type as "S", "R", or "T", for square, rect., or triangle.
*           NIL returned if Esc pressed.
static nLastGrid  := 1                  // Remembers last grid type chosen
local nBR         := nTR +  5
local nRC         := nLC + 30
local cGridType   := NIL
local cOrgColor   := ""
default cColor to m->C_Normal
cOrgColor         := setcolor(cColor)

MenuBox(nTR,nLC,nBR,nRC)
@nTR+1,nLC+2 say    "Choose Grid Type "
@nTR+2,nLC+2 prompt " Square "
@nTR+3,nLC+2 prompt " Rectangular "
@nTR+4,nLC+2 prompt " Triangular "
menu to nLastGrid
if nLastGrid == 1
   * Square grid.
   cGridType := "S"
elseif nLastGrid == 2
   * Rect. grid.
   cGridType := "R"
elseif nLastGrid == 3
   * Triangular grid.
   cGridType := "T"
endif
setcolor(cOrgColor)
return (cGridType)
*** End of Func: ChooseGrid()

*****************************
Function ChooseInput(nTR,nLC)
* Choose input from Screen/File.
* Input: nTR is top row.
```

Code File: EGMCMain.Prg

```
*       nLC is left col.
*
static nLastInput := 1                  // Remembers last input type chosen
local nBR         := nTR +  4
local nRC         := nLC + 30
local cOrgColor   := setcolor(m->C_Menu1)

MenuBox(nTR,nLC,nBR,nRC)
@nTR+1,nLC+2 say    "Enter Data From?"
@nTR+2,nLC+2 prompt " F  File Input "
@nTR+3,nLC+2 prompt " S  Screen Input N/A"
menu to nLastInput

setcolor(m->C_Normal)
if nLastInput == 1
   * File input.
   InputFromFile(nTR+2,nLC+1)
elseif nLastInput == 2
   * Screen input.
   // N/A GetProbHit(ChooseGrid(nTR+2,nLC+1), VER_DATE)
endif
setcolor(cOrgColor)
return (NIL)
*** End of Func: ChooseInput()

*********************
Function DOS_Prompt()
* Shell to DOS.
* Returns: NIL
local nMajErr  := 0                     // Major error code
local nMinErr  := 0                     // Minor error code
local lSuccess := .F.
set color to
cls
setcolor(m->C_Help)
scroll(0,0,4,79)
@0,0 to 4,79
@1,2 say "Type  EXIT  at DOS prompt to return to ELIPGRID-MC program."
@3,2 say "The DOS  MEM  command will give largest executable program size."
* Blinker 3.0 command, swpruncmd("",0,"",""),
*  leaves much more memory free than Clipper run command.
* Default swpruncmd() parameters: run command.com, free as much mem as possible,
*  leave current path the default, swap to current path.
lSuccess := swpruncmd("",0,"","")
if ! lSuccess
   scroll(0,0,2,79)
   ? "DOS access failed."
   nMajErr := swperrmaj()
   nMinErr := swperrmin()
   ? "Blinker major, minor error codes: ", NumTrim(nMajErr)+",",NumTrim(nMinErr)
   ? "Press a key to continue..."
   inkey(0)
endif
return(NIL)
*** End of Func: DOS_Prompt()

*************************************************
Function Err_MsgBox(nTR, cType, cLin1, cLin2, cLin3)
* Generic error or msg box.  Defaults to error box.
* Displays up to 3 lines + Press key msg and waits for keypress.
* Returns: NIL
local cTmpScn       := ""
local lDispMsg      := .T.
local nMaxLineWdth  := 0
local nWidth        := 0
local cOrgClr       := ""
```

Code File: EGMCMain.Prg

```
local nLC            := 0
local nBR            := 0
local nRC            := 0
local nLines         := 0
local nCurRow        := 0
local nCurCol        := 0
default cType to "E"                    // Default to error box

* Set box color.
if upper(cType) == "E"
   cOrgClr := setcolor(m->C_Error)
else
   cOrgClr := setcolor(m->C_Help)
endif

* Get current cursor pos.
nCurRow := row()
nCurCol := col()

if (valtype(cLin3) == "C")
   * 3 lines to display
   nBR := nTR + 4 + 3                // 4 lines for misc. + 3 msg lines
   nMaxLineWdth := max( max(len(cLin1), len(cLin2)), len(cLin3) )
   nLines := 3
elseif (valtype(cLin2) == "C")
   * 2 lines to display
   nBR := nTR + 4 + 2                // 4 lines for misc. + 2 msg lines
   nMaxLineWdth := max(len(cLin1), len(cLin2))
   nLines := 2
elseif (valtype(cLin1) == "C")
   * 1 line to display
   nBR := nTR + 4 + 1
   nMaxLineWdth := len(cLin1)
   nLines := 1
else
   * Incorrect params. passed
   lDispMsg := .F.
endif

* Display message.
if (lDispMsg)
   nMaxLineWdth := max( nMaxLineWdth, len("Press a key to continue...") )
   nWidth := 4 + nMaxLineWdth          // 2 lines/blanks + largest line
   nLC := (79 - nWidth)/2              // center
   nRC := nLC + nWidth - 1
   cTmpScn := savescreen(nTR, nLC, nBR+1, nRC+1)
   MenuBox(nTR,nLC,nBR,nRC)
   if (nLines >= 1)
      @nTR+2, nLC + 2 say cLin1
   endif
   if (nLines >= 2)
      @nTR+3, nLC + 2 say cLin2
   endif
   if (nLines == 3)
      @nTR+4, nLC + 2 say cLin3
   endif
   @nBR-1, nLC + 2 say "Press a key to continue..."
   tone(440,1)
   inkey(0)
   restscreen(nTR, nLC, nBR+1, nRC+1, cTmpScn)
else
   @0,0
   @0,0 say " Err_MsgBox() error: Check parameters.   Press a key to return... "
   inkey(0)
endif (lDispMsg )
setcolor( cOrgClr )
```

Code File: EGMCMain.Prg

```
@nCurRow, nCurCol say ""
return (NIL)
*** End of Func: Err_MsgBox()

*******************************
Function InputFromFile(nTR,nLC)
* Get input data from ELIPGRID type file or SIF type file.
* Input: nTR = Top row for box.
*        nLC = Left col for box.
static nLastType  := 1                  // Remembers last file type chosen
static nLastFileE := 1                  // Remembers last ELIPGRID type file ch
static nLastFileS := 1                  // Remembers last SIF type file ch
local nBR         := nTR +  4
local nRC         := nLC + 30

MenuBox(nTR,nLC,nBR,nRC)
@nTR+1,nLC+2 say    "Choose Input File Format"
@nTR+2,nLC+2 prompt "SIF      Type Format"
@nTR+3,nLC+2 prompt "ELIPGRID Type Format N/A"
menu to nLastType
if nLastType == 1
   * SIF Format.
   SIF_FileInput(nTR+2,nLC+1,@nLastFileS,VER_DATE) // Pass nLastFileS by refer.
elseif nLastType == 2
   * ELIPGRID Format.
// N/A FileInput(nTR+2,nLC+1,@nLastFileE,VER_DATE)     // Pass nLastFileE by refer.
endif
return (NIL)
*** End of Func: InputFromFile()

************************************************
Function MenuBox(nTR,nLC,nBR,nRC, cSides, lShadow)
* Draw box sides for a menu.
* cSides defaults to double top, single sides.
* cSides could be defined constants from from Box.Ch.
* lShadow defaults to .T.
local cOrgColor := setcolor()
default cSides to B_DOUBLE_SINGLE
default lShadow to .T.
if lShadow
   set color to
   scroll(nTR+1,nLC+1,nBR+1,nRC+1)
   setcolor(cOrgColor)
   scroll(nTR,nLC,nBR,nRC)
endif
dispbox(nTR,nLC,nBR,nRC, cSides)
return (NIL)
*** End of Func: MenuBox()

************************************************************
Function MenuCenter(nRow, aPrmpts, nChoice, nType,nLeftCol)
* Displays centered menu of prompts.
* Returns menu choice.
* Returns left col of menu when nLeftCol is passed in by reference.
local nLong   := 0
local nPLen   := 0
local nPrmpts := len(aPrmpts)
local nLCol   := 0
local i
default nChoice to 1                    // 1st choice to highlight
default nType to 1                      // 1=Double top, single side,2=all double

* Find longest prompt, set nLong.
for i = 1 to len(aPrmpts)
   nPLen := len(aPrmpts[i])
   nLong := if(nPLen > nLong, nPLen, nLong)
```

Code File: EGMCMain.Prg

```
next i
nLCol    := BoxCenter(nRow,nPrmpts+1,nLong+4,nType)
nLeftCol := nLCol
for i = 1 to nPrmpts
   @nRow+i,nLCol+1 prompt " " + padr(aPrmpts[i],nLong) + " "
next i
menu to nChoice
return (nChoice)
*** End of Func: MenuCenter()

*************************
Function NotReadyYet(cMsg)
* Not ready yet msg.
save screen
cls
@0,0 to 5,79
@ 2, 2 say cMsg + " option is not ready yet."
@ 4, 2 say "Press a key to continue..."
inkey(0)
return (NIL)
*** End of Func: NotReadyYet()

**********************
Function NumTrim(nNum)
* Returns nNum in str form trimmed.
local cNumStr := alltrim(str(nNum))
return (cNumStr)
*** End of Func: NumTrim()

****************************
Function ParamHelp(cVerDate)
* Parameter help screen.
set color to W+/N
cls
?? repli("=",80)
?? "ORNL ELIPGRID-MC Program, Version: " + cVerDate
? " This program calculates the probability of missing a hot spot by both"
? " Monte Carlo and corrected ELIPGRID-PC techniques.  Note that two "
? " different sets of linear regression coefficients are available for "
? " triangular grids in the discontinuity region."
?
? "Usage: EGMC [H | OLR]"
?
? "       EGMC          = Uses new 4th-order multiple regression coefficients for"
? "                       triangular grids in region of discontinuity, i.e.,"
? "                       L/G > 0.5 and < 0.6 and shape >= 0.85 and < 1.0."
? "       EGMC H[elp]   = Help on command line parameters, this screen."
? "       EGMC OLR      = Uses old linear regression coefficients for triangular"
? "                       grid discontinuity region.  These are based on a series"
? "                       of 4th-order simple regressions of probability vs L/G"
? "                       for various shapes.  Usually, these are less accurate."
?
?
? " Example: EGMC OLR"
? "          Use old linear regression coefficients."
? repli("=",80)
return (NIL)
*** End of Func: ParamHelp()
*****************************
Function SayCenter(nRow, cMsg)
* Displays cMsg on centered nRow.
local nCol := (80-len(cMsg))/2
@nRow,nCol say cMsg
return (NIL)
*** End of Func: SayCenter()
*** End of File: EGMCMain.Prg
```

Code File: EGMCFile.Prg

```
//=============================================================================
// File:          EGMCFile.Prg
// For:           EGMC.Exe, for ELIPGRID-MC Monte Carlo test program.
// Purpose:       Provides file input code.
// Author:        Jim Davidson
// Prog Started:  10/27/94 from ELIPGRID-PC code, 09/06/94 version.
// Last Mod:      11/13/94
// Note:          Functions are arranged in alphabetical order.
//=============================================================================

// Include files
#include "Inkey.Ch"                    // key definitions
#include "Directry.ch"                 // File info definitions

// Max line width for SIF input files.
#define nSIF_LINE_WDTH 90

// User-defined command
#xcommand DEFAULT <TheParam> TO <DefaultVal> => ;
   IF (<TheParam> == NIL); <TheParam>:=<DefaultVal>; ENDIF

*****************************************************
Function AlertBox(nTR, acOptions, nLin1, nLin2, nLin3)
* Substitute for alert() function.  Alert() does not obey color settings.
* AlertBox obeys current color setting.  Alert() is hard to read on LCD screens.
* Lines 2 and 3 are optional.
* Returns: Esc = 0, else number of array element of acOptions chosen.
local cTmpScn        := ""
local lDispMsg       := .T.
local nMaxLineWdth   := 0
local nPrmptWdth     := 0
local nWidth         := 0
local cOrgClr        := ""
local nLC            := 0
local nBR            := 0
local nRC            := 0
local nLines         := 0
local nCurRow        := 0
local nCurCol        := 0
local nNumOps        := len(acOptions)
local nCurOp         := 1
local nOpCol         := 0
local nRtnVal        := 0

* Set box color.
cOrgClr := setcolor(m->C_Error)

* Get current cursor pos.
nCurRow := row()
nCurCol := col()

if (valtype(nLin3) == "C")
   * 3 lines to display
   nBR := nTR + 4 + 3                  // 4 lines for misc. + 3 msg lines
   nMaxLineWdth := max( max(len(nLin1), len(nLin2)), len(nLin3) )
   nLines := 3
elseif (valtype(nLin2) == "C")
   * 2 lines to display
   nBR := nTR + 4 + 2                  // 4 lines for misc. + 2 msg lines
   nMaxLineWdth := max(len(nLin1), len(nLin2))
   nLines := 2
elseif (valtype(nLin1) == "C")
   * 1 line to display
   nBR := nTR + 4 + 1
   nMaxLineWdth := len(nLin1)
```

Code File: EGMCFile.Prg

```
   nLines := 1
else
   * Incorrect params. passed
   lDispMsg := .F.
endif

* Display message.
if (lDispMsg)
   * Get total width of the prompts plus inner spacing.
   for nCurOp = 1 to nNumOps
       nPrmptWdth := nPrmptWdth + len(acOptions[nCurOp])
   next nCurOp
   nPrmptWdth := nPrmptWdth + 3 * (nNumOps-1)

   * Determine overall width of box.
   nMaxLineWdth := max(nMaxLineWdth, nPrmptWdth)
   nWidth := 4 + nMaxLineWdth
   nLC := (79 - nWidth)/2              // center
   nRC := nLC + nWidth - 1
   cTmpScn := savescreen(nTR, nLC, nBR+1, nRC+1)
   MenuBox(nTR,nLC,nBR,nRC)
   if (nLines >= 1)
      @nTR+2, nLC + 2 say nLin1
   endif
   if (nLines >= 2)
      @nTR+3, nLC + 2 say nLin2
   endif
   if (nLines == 3)
      @nTR+4, nLC + 2 say nLin3
   endif

   * Display and get desired menu option.
   for nCurOp = 1 to nNumOps
      if nCurOp == 1
         nOpCol := nLC + 2
      else
         nOpCol := nOpCol + len(acOptions[nCurOp-1]) + 3
      endif
      @nBR-1, nOpCol prompt acOptions[nCurOp]
   next nCurOp
   tone(440,0.3)
   menu to nRtnVal

   restscreen(nTR, nLC, nBR+1, nRC+1, cTmpScn)
else
   @0,0
   @0,0 say " AlertBox() error: Check parameters.   Press a key to return... "
   inkey(0)
endif (lDispMsg )
setcolor( cOrgClr )
@nCurRow, nCurCol say ""
return (nRtnVal)
*** End of Func: AlertBox()

*********************************
Function ExtrctPath(cPathFileN)
* Extract path from cPathFileN.
* Example: ExtrctPath("D:\file.ext") ==> "D:\"
* Based on Environ.prg fuction FilePath() supplied by Clipper.
local nBkSlshPos := 0
local cPath := ""
nBkSlshPos := rat("\", cPathFileN)
if nBkSlshPos == 0
   cPath := ""
else
   cPath := substr(cPathFileN, 1, nBkSlshPos)
```

Code File: EGMCFile.Prg

```
endif
return (cPath)
*** End of Func: ExtrctPath()

*****************************************************************************
Function GetFileBox(nTR, nLC, nBR, nRC, cDirSpec, lDispBox, cColor, nInitFile)
* Pop-up file selector, all params. are optional
* Parameter defaults:
*    nTR top row   ==> to  0
*    nLC left col  ==> to  0
*    nBR bot row   ==> to maxrow
*    nRC right col ==> to nLC + 38
*    cDirSpec      ==> "*.*"
*    lDispBox      ==> .T.
*    ColorVAr      ==> "W+/n,n/W"
*    nInitFile     ==> 1
* Returns:
*    if Enter key ==> File name
*    if Esc   key ==> NIL
*    if error     ==> NIL
**
local cOrgClr     := ""
local cFileName   := NIL
local cTmpScn     := ""
local i                                 // Scratch
local aDrctry     := {}                 // Array of dir info
local acFileNames := {}                 // Array of file names
local nFileChoice := 0

* If any param. not passed, below assigns defaults as needed.
default nTR to 0
default nLC to 0
default nBR to maxrow()
default nRC to nLC + 38
default cDirSpec to "*.*"
default lDispBox to .T.
default cColor to (m->C_Help)
default nInitFile to 1

cTmpScn := savescreen(nTR,nLC,nBR+1,nRC+1)   // +1 for shadow lines

if (!SubDir(cDirSpec))
   Err_MsgBox(15,"E","No .SIF files found in current subdir.")
   return (NIL)
endif

cOrgClr := setcolor(cColor)
scroll(nTR,nLC,nBR,nRC)
if (lDispBox)
   MenuBox(nTR,nLC,nBR,nRC)
endif
@nTR,nLC+2 say " Choose Input File... "

aDrctry := directory(cDirSpec)
* Sort array according to file name.
asort(aDrctry,,, {|FrstName,NextName| FrstName[F_NAME] < NextName[F_NAME]})

* Fill an array with file info to display.
acFileNames := {}
for i = 1 to len(aDrctry)
   aadd(acFileNames, ;
      padl(aDrctry[i,F_NAME],13) + ;
      padl(numtrim(aDrctry[i,F_SIZE]),8) + ;
      padl(dtoc(aDrctry[i,F_DATE]), 9) + ;
      padl(substr(aDrctry[i,F_TIME],1,5),6) )
next i
```

Code File: EGMCFile.Prg

```
* Display files and get choice.
nFileChoice := achoice(nTR+1,nLC+1,nBR-1,nRC-1, acFileNames,,,nInitFile)
if (nFileChoice != 0)
   * Is 0 if Esc key exit
   cFileName    := aDrctry[nFileChoice,F_NAME]
   nInitFile    := nFileChoice
endif
setcolor(cOrgClr)
restscreen(nTR,nLC,nBR+1,nRC+1,cTmpScn)
return (cFileName)
*** End of Func: GetFileBox()

*****************************************
Function GetFilOutFile(cInFile, lProceed)
* Returns file output file name entered by user.
* Updates flag lProceed.
* lProceed parameter should be passed in by reference.
static cOutFile   := ""             // Screen output file
local nChoice     := 1
local GetList     := {}
local cCurrPath   := ""
local lDone       := .F.
local lOrgReadIns := readinsert(.T.)   // Insert mode for read = on.

* Make default outfile name.
cCurrPath    := diskname() + ":" + dirname()
* Make output file name, cInFile plus .OUT.
if at(".",cInFile) == 0
  cOutFile := cInFile + ".OUT"
else
  cOutFile := substr(cInFile,1,at(".",cInFile)) + "OUT"
endif
* Add trailing \ to path, if needed.
cOutFile :=  cCurrPath + iif(right(cCurrPath,1)=="\","","\") + cOutFile

do while ! lDone
   cls
   MenuBox(2,1,8,67)
   cOutFile := padr(cOutFile,64)
   @03,03 say "Enter output file name:"
   @04,03 get cOutFile pict "@!"
   @05,03 say "Current path: " + cCurrPath
   keyboard chr(K_END)
   read
   readinsert(lOrgReadIns)
   cOutFile := alltrim(cOutFile)

   if lastkey() == K_ESC
      lProceed := .F.
   else
      lProceed := .T.
      if file(cOutFile)
         * Decide whether to overwrite output file.
         nChoice := AlertBox(8,{"YES, Overwrite It", "Enter New Name"}, ;
            "Warning: Above output file exists!","", ;
            "          Overwrite it?          ")
         if nChoice == 1
            * Overwrite output file.
            set alternate to (cOutFile)
         elseif nChoice == 2
            * Enter new name.
            loop
         else
            * Esc key.  Don't open output file.
            lProceed := .F.
         endif
```

Code File: EGMCFile.Prg

```
      else
         * cOutFile does not exist, try to open it.
         * First test for valid subdir and valid file name.
         if ! Subdir(ExtrctPath(cOutFile)) .or. ;
            ! filevalid(token(cOutFile,":\"))
            * Invalid path or file name.
            Err_MsgBox(10,"E","Error: Invalid path or file name.", ;
                    "File.: " + cOutfile)
            loop
         else
            * Valid path, open file for output.
            set alternate to (cOutFile)
         endif
      endif
   endif
   lDone := .T.
enddo
return (cOutFile)
*** End of Func: GetFilOutFile()

*****************************************************
Function SIF_FileInput(nTR, nLC, nInitFile, cVerDate)
* Gets data from SIF format input file.
* Writes output to cInFile.Out.
* Input:    nTR is row for file selection box,
*           nLC is left col.
*           nInitFile is initial file to highlight.
*           cVerDate is program version date.
* Returns:  NIL
// For anResults array used by MC_ProbHit() function.
#define nHIT_PROB    1                // mean hit probability
#define nSTD_DEV     2                // std. dev. of the mean
#define nBOUND       3                // 99% confidence half-interval
#define nTOT_TRIALS  4                // total trials needed

local lReadInsert    := readinsert(.T.)
local cInFile        := ""            // Input file name
local cOutFile       := ""
local cDataLine      := ""
local nSemiMajor     := 0
local nShape         := 0             // Ellipse semi-major/semi-minor axis
local nAngle         := 0
local nGSize         := 0             // Grid size, for Rec. grids, short side
local nGTyp          := 0             // Grid type, 1=Sq., 2=Tri., 3=Rec.
local nOrientn       := 0             // Specific angle or "random",
local cTrgtID        := ""            //   if nOrientn > 0 use "random" angles
local nInputLine     := 0             // Current file input line
local nRecRatio      := 0             // Rec. grid long side/short side ratio
local nLines         := 0             // Lines in input file
local nProbNoHit     := 0
local nProbSum       := 0.0           // Used for "random" angle case
local nCrntAngle     := 0             // Used for "random" angle case
local nLrgstAngle    := 0             // Used for "random" angle case
local cFileText      := ""
local lProceed       := .T.
local GetList        := {}            // Stops compiler warnings
* Monte Carlo related vars.
local nDsrdBound     := 0.005         // Desired 99% confidence half-interval
local nMaxRuns       := 100           // Maximum number of simulation runs
local nNumRuns       := 0             // Number of runs actually needed to
                                      // achieve error bound
local nNumTrials     := 1000//10000       // Number of trials per simulation run
local lConver        := .F.           // Flag for Monte Carlo reaching bound
local anResults[4]                    // Monte Carlo results array

cInFile     := GetFileBox(nTR,nLC,,,"*.SIF",,,@nInitFile)
```

Code File: EGMCFile.Prg

```
cFileText   := memoread(cInFile)
nLines      := mlcount(cFileText)
if lastkey() == K_ESC .or. empty(cInFile)
   * Just return, if Esc key pressed.
elseif nLines < 3
   * Error, invalid file.
   Err_MsgBox(10,"E","Error: Less than 3 lines in file.", ;
              "File.: " + cInFile, ;
              "Need (1) Title line, (2) Data line, (3) EOF line.")
else
   * Input file looks OK, create output file name.
   cOutFile := GetFilOutFile(cInFile, @lProceed)

   if lProceed
      * Do the work!
      cls
      set alternate to (cOutFile)
      set alternate on
      ?? "Output from ORNL ELIPGRID-MC Monte Carlo Test Program Version: " + cVerDate
      ? "File Name.: " + cOutFile
      ? "Created on: " + dtoc(date())
      ? "Input file: " + cInFile + " using SIF format."
      ? "Title line: " + memoline(cFileText,,1)
      ?
      ? "Target  Grid Type        Semi-major Axis    Gridspace    Shape   Angle   ELIPGRID MCarlo
EG - MC  MCarlo    MC 99%  EG in"
      ? "                        in Relative Units  in Orig Units                Prob(0)  Prob(0)
Diff   NumTrials  ½-C.I.  99% C.I."

      * Get data lines
      nInputLine  := 2                 // Skip title line
      do while nInputLine <= nLines
         cDataLine   := alltrim(memoline(cFileText,nSIF_LINE_WDTH,nInputLine))
         if left(cDataLine,1) == "*"
            * Comment line.
            nInputLine++
            loop
         endif
         * Parse the data values.
         nSemiMajor  := val(substr(cDataLine, 1,at(" ",cDataLine)))
         cDataLine   := ltrim(substr(cDataLine,at(" ",cDataLine)))
         nShape      := val(substr(cDataLine, 1,at(" ",cDataLine)))
         cDataLine   := ltrim(substr(cDataLine,at(" ",cDataLine)))
         nAngle      := val(substr(cDataLine, 1,at(" ",cDataLine)))
         cDataLine   := ltrim(substr(cDataLine,at(" ",cDataLine)))
         nGSize      := val(substr(cDataLine, 1,at(" ",cDataLine)))
         cDataLine   := ltrim(substr(cDataLine,at(" ",cDataLine)))
         nGTyp       := val(substr(cDataLine, 1,at(" ",cDataLine)))
         cDataLine   := ltrim(substr(cDataLine,at(" ",cDataLine)))
         nOrientn    := val(substr(cDataLine, 1,at(" ",cDataLine)))
         cDataLine   := ltrim(substr(cDataLine,at(" ",cDataLine)))
         cTrgtID     := cDataLine

         if nShape > 1.0 .or. nShape < 0.05 .or. nSemiMajor/nGSize > 3.0
            * EOF or error in shape or L/G ratio > 3.
            exit                     // Exit do while loop
         endif

         if nGTyp == 3
            * If rect. grid, get long/short ratio from next line.
            nInputLine++
            cDataLine := memoline(cFileText,nSIF_LINE_WDTH,nInputLine)
            cDataLine := alltrim(cDataLine)  // Had problems with getting valid
            nRecRatio := val(cDataLine)      // nRecRatio.  Used multiple command lines.

            if nRecRatio == 1.0
```

Code File: EGMCFile.Prg

```
      * Trap for a rect. grid with a long/short side ratio of 1.0.
      * Use a sq. grid since problems can develop using rect. grid in
      * certain cases.  Problem found in tech. review by John Wilson.
      nGTyp := 1
   endif
elseif nGTyp == 1
   * Sq. grid
   nRecRatio := 1.0
endif

*---------------| Calculate probability of no hit, P(0) |------*
if nOrientn <= 0.0
   * Calcualte for a single angle.
   nProbNoHit := ElipGrid(nSemiMajor,nShape,nAngle,nGSize,nGTyp, ;
      nRecRatio)
   * Calculate for a single angle using Monte Carlo algorithm.
   lConver := MC_ProbHit(nSemiMajor,nShape,nAngle,nGSize,nGTyp,nRecRatio, ;
      nMaxRuns, nNumTrials, nDsrdBound, anResults)
   if lastkey() == K_ESC
      * Esc key abort
      exit
   endif
else
   * Calculate for average of multiple angles,
   * i.e., "random" choice in Singer's 1972 ELIPGRID.
   if nGTyp == 1
      nLrgstAngle := 45
   elseif nGTyp == 2
      * For triangular grid (hexagon).
      nLrgstAngle := 30
   elseif nGTyp == 3
      * For rectangular grid.
      nLrgstAngle := 90
   endif
   * Sum up multiple angles results.
   nProbSum := 0.0
   for nCrntAngle = 0 to nLrgstAngle
      nProbNoHit := ElipGrid(nSemiMajor,nShape,nCrntAngle,nGSize, ;
         nGTyp, nRecRatio)
      nProbSum := nProbSum + nProbNoHit
   next nCrntAngle

   * Calculate average.
   nProbNoHit  := nProbSum/(nLrgstAngle+1)
endif
*-----------------------------------------------------------------*

* Print a line of data.
? padr(cTrgtID,8)
if nGTyp == 1
   ?? "Square       " + space(8)
elseif nGTyp == 3
   ?? "Rectangular, " + trans(nRecRatio,"99.9") + "/1  "
elseif nGTyp == 2
   ?? "Triangular   " + space(8)
endif

* Print data fields.
?? trans(nSemiMajor/nGSize,"9999.9999")+ space(6) + ;
   trans(nGSize,"9999.99")              + space(7) + ;
   trans(nShape,"9.99")                 + space(3) + ;
   iif(nOrientn > 0,"Random",trans(nAngle,"99.9"+"  ")) + ;
      space(2) + ;
   trans(nProbNoHit,"9.9999") + space(3) + ; // Monte Carlo related
   trans(1.0-anResults[nHIT_PROB],"9.9999") + space(2) + ;
   trans(nProbNoHit-(1.0-anResults[1]),"99.9999") + ;
```

Code File: EGMCFile.Prg

```
            trans(anResults[nTOT_TRIALS], "9,999,999") + space(3) + ;
            trans(anResults[nBOUND], "9.9999") + space(4) + ;
            iif(abs(nProbNoHit-(1.0-anResults[1])) <=anResults[nBOUND], "Y","N")
         * Increment line index.
         nInputLine++
      enddo
      ?
      ? "END OF RUN  (OR ERROR IN SHAPE OR L/G RATIO > 3)"
      set alternate to
      set alternate off
      setcolor(m->C_Help)
      scroll(0,0,4,79)
      @0,0 to 4,79 double
      @1,2 say "Output written to file: " + cOutFile
      @2,2 say "Current subdirectory..: " + diskname() + ":" + dirname()
      @3,2 say "Press a key to continue..."
      inkey(0)
   endif
endif
readinsert(lReadInsert)
return (NIL)
*** End of Func: SIF_FileInput()

*****************************
Function Subdir(cTestSubdir)
* Returns .T. if cTestSubdir exits, .F. otherwise.
* The directory() command will return an empty array
*  if cTestSubdir does not exist.
local lRtnVal  := .F.
local aDirctry := {}

cTestSubdir := alltrim(cTestSubdir)
* directory() returns an empty array, {}, if invalid cTestSubdir.
aDirctry := directory(cTestSubdir, "D") // D to include all subdirs
if len(aDirctry) > 0
   lRtnVal := .T.
endif
return (lRtnVal)
*** End of Func: Subdir()

*** End of File: EGMCFile.Prg
```

Code File: EGMCFort.Prg

```
//=============================================================================
// File:          EGMCFort.Prg
// For:           EGMC.Exe, for ELIPGRID-MC Monte Carlo test program.
// Purpose:       Provides ELIPGRID FORTRAN code in Clipper form.
//                Note correction to ELIPGRID in RECT subroutine.
// Author:        Jim Davidson
// Prog Started:  10/27/94 from 09/06/94 version of ELIPGRID-PC.
// Last Mod:      11/30/94
// Note:          Functions are arranged in alphabetical order.
//=============================================================================

************************************************
Function ElipGrid(A, Shape, Angle, GdSpac, Net, Q)
* This function is taken from Singer's 1972 ELIPGRID program.
* It retains the original algorithm, but is modified to remove
* all goto type statements.  Many line numbers have been left in the
* comments as references back to the original code.
*
* Shape      : Shapes > 1.0  are trapped before reaching this function.
* assumptions: Shapes < 0.05 are trapped before reaching this function.
*              Trapping all Shapes < 0.05 is somewhat more restrictive than
*              ELIPGRID, but should have little practical consequences.
*              I would like to see a verification of the math before accepting
*              arbitrarily small Shapes, JRD, 04/15/94.
* L/G
* assumptions: Code assumes all L/G ratios > 3.0 are trapped.  Very large L/G
*              ratios, e.g. 6 or 7, have caused problems.  No known practical
*              need requires them.  Singer's largest L/G ratio in his 100
*              cases was 2.83. Gilbert's largest L/G ratio in nomographs is 1.0.
*
* Note that the MET parameter described below, is not used in this function.
* Random angle case is taken care of by calling code.
*
* Below is original code documentation.
*
*                     PROGRAM ELIPGRID
*
*    PROGRAM TO DETERMINE THE PROBABLITY OF LOCATING AN ELLIPTIC OR
*    CIRCULAR TARGET WITH A SQUARE, HEXAGONAL OR RECTANGULAR GRID
*
*
*       DESCRIPTION OF PARAMETERS
*
*     TARGET= ANY IDENTIFICATION OF TARGET   (READ IN "A" FORMAT)
*          A= LENGTH OF SEMIMAJOR AXIS OF TARGET
*      SHAPE= SHAPE OF TARGET - SEMIMINOR AXIS DIVIDED BY THE SEMIMAJOR
*      ANGLE= POSITIVE ANGLE BETWEEN LONG AXIS OF TARGET AND GRID
*             DIRECTION - FOR A SQUARE GRID ANGLE CAN BE ANY ABGLE FROM
*             0 TO 45 DEGRESS, FOR A HEXAGONAL GRID ANGLE CAN BE ANY
*             ANGLE FROM 0 TO 30 DEGREES INCLUSIVE, FOR A RECTANGULAR
*             GRID ANGLE CAN BE ANY ANGLE FROM 0 TO 90 DEGREES
*             INCLUSIVE AND IS MEASURED FROM THE X AXIS OF THE GRID
*     GDSPAC= DISTANCE BETWEEN POINTS ON THE GRID (IN THE SAME UNITS AS
*             "A") - FOR A RECTANGULAR GRID GDSPAC IS THE DISTANCE
*             BETWEEN POINTS ALONG THE Y AXIS OF THE GRID
*        NET= GRID TYPE - SQUARE GRID=1,HEXAGONAL GRID=2, RECTANGULAR
*             GRID=3
*        MET= SPECIFIC OR RANDOM ORIENTATION - IF MET>0 - RANDOM
*          Q= SHAPE OF RECTANGULAR GRID - LONG(X) AXIS DIVIDED BY THE
*             SHORT(Y) AXIS
*

* These locals are integers in ELIPGRID.
local I        := 0
local IBLANK   := 0
local IWARN    := 0
```

Code File: EGMCFort.Prg

```
local IZONK     := 0
local M         := 0

* These locals are reals in ELIPGRID.
local ALPHA     := 0
local ANP       := 0
local AQUAR     := 0
local AREA1     := 0
local AREA2     := 0
local AREA3     := 0
local AREA4     := 0
local AREA5     := 0
local AREA6     := 0
local AREA7     := 0
local AREA8     := 0
local AREA9     := 0
local AREA10    := 0
local ASQ       := 0
local AVPR0     := 0
local AVPR1     := 0
local AVPR2     := 0
local B         := 0
local BALLS     := 0
local BSQU      := 0
local C         := 0
local CAROL     := 0
local CIM       := 0
local CNM       := 0
local D         := 0
local DJ0       := 0
local DJ1       := 0
local DM0       := 0
local DM1       := 0
local EQU       := 0
local FIN       := 0
local FORN      := 0
local GAME      := 0
local GRO       := 0
local HAI       := 0
local HALFC     := 0                    // C/2
local HALFD     := 0                    // D/2
local HALFJ0    := 0                    // J0/2
local HALFJ1    := 0                    // etc.
local HALFM0    := 0
local HALFM1    := 0
local HORN      := 0
local PET       := 0
local PI        := 0                    // Constant pi, 3.141592 in ELIPGRID
local POT       := 0
local PROB0     := 0                    // Prob. of no hits
local PROB1     := 0                    // Prob. of 1 hit
local PROB2     := 0
local R0        := 0
local RDW       := 0
local REVA      := 0                    // For rect. grid, transformed A
local REVANG    := 0                    // For rect. grid, transformed angle
local REVK      := 0                    // For rect. grid, transformed SHAPE
local SER       := 0
local SLING     := 0
local SNGLE     := 0
local SUM0      := 0
local SUM1      := 0
local SUM2      := 0
local T         := 0
local TIN       := 0
local TIZ       := 0
```

Code File: EGMCFort.Prg

```
local WINE      := 0
local XHAPE     := 0
local XI        := 0
local XM        := 0
local Y1        := 0
local YI        := 0
local YAM       := 0
local YM        := 0
local ZAP       := 0

/*** Note 08/11/94, JRD ***/
*--- Force full correction level all the time. ---*
m->nElpGrdCor := 3

* Below are assignments made in ELIPGRID.
PI     := 3.141592                      // Follows original value.
TIZ    := 0.50000
RDW    := SQRT(3.0)*0.5

IZONK := IBLANK
A     := A/GDSPAC
SLING := A
XHAPE := SHAPE
SNGLE := ANGLE
SUM1  := 0.0
SUM2  := 0.0
SUM0  := 0.0

*
*    AREAS 1 TO 10 ARE RELATIVE AREAS OF OVERLAP IN THE TRANSFORMED NET
*   35
AREA1     := 0.0
AREA2     := 0.0
AREA3     := 0.0
AREA4     := 0.0
AREA5     := 0.0
AREA6     := 0.0
AREA7     := 0.0
AREA8     := 0.0
AREA9     := 0.0
AREA10    := 0.0

*
*    PROB0 IS THE PROBABILITY OF MISSING THE TARGET
*    PROB1 IS THE PROBABILITY OF LOCATING THE TARGET ONCE
*    PROB2 IS THE PROBABILITY OF LOCATING THE TARGET TWO OR MORE TIMES
*
PROB0 := 0.0
PROB1 := 0.0
PROB2 := 0.0
*
*    DETERMINES THE GRID TYPE
*
* GO TO (65,40,45),NET
if NET == 2
   ***New Code, 04/04/94, JRD***
   * Handle problem with tri. grid discontinuity near L/G = 0.577.
   * A is the L/G ratio.
   * If ElipGrid correction level is >= 2, consider 4th order linear regression.
   * 11/11/94 Prob0_Regr() now has new 2nd order multi-regr. option developed
   * from regressing Monte Carlo results on Shape and A.  Command line param.
   * OLR will force "old linear regression" option instead of new reg. default.
   if m->nElpGrdCor >= 2
      if (A > 0.50 .and. A < 0.60) .and. (Shape >= 0.85 .and. Shape < 1.0)
         * Use 4th order linear regression results, not ELIPGRID algorithm.
         return(Prob0_Regr(A,Shape))
```

Code File: EGMCFort.Prg

```
      endif
   endif
   ***End New Code, 04/04/94***
   *
   *   HEXAGONAL NET
   *  40
   FIN   := RDW
   * IROT  := 30 not needed in this function.
   ZAP   := 6.0
   BALLS := 0.57735
   * GO TO 75
elseif NET == 3
   *
   *   RECTANGULAR NET
   *
   *   45 IF (MROT)  50,50,60
   *
   *    READ SHAPE OF RECTANGULAR GRID
   *
   *   50 READ (IREAD,55) Q
   *   55 FORMAT (F10.5)
   *   60 CALL RECT(SLING,XHAPE,ANGLE,Q,REVK,REVA,REVANG)                    GRID1035
   *   Argument SLING is never used by subroutine RECT().
   RECT(XHAPE,ANGLE,Q,@REVK,@REVA,@REVANG)
   SHAPE := REVK
   A     := REVA*SLING
   ANGLE := REVANG
   * IROT  := 90 not needed in this function.
   * GO TO 70
   * 70
   FIN   := 1.000
   ZAP   := 4.0
   BALLS := 0.707107
elseif NET == 1
   *
   *
   *     SQUARE NET
   *
   *   65 IROT=45
   * IROT := 45 not needed in this function.
   * 70
   FIN   := 1.000
   ZAP   := 4.0
   BALLS := 0.707107
endif

* SHAPE restiction below handled by trapping ALL SHAPES < 0.05 in calling
* code.  This is more restrictive than SHAPES < 0.05 and A (L/G) > 2 test below.
* ELIPGRID-MC also traps all L/G ratios > 3.0.
* 75 IF (SHAPE-0.05)  80,95,95
* 80 IF (A-2.0)  95,95,85
* 85 WRITE (IPRIN,90) TARGET
* 90 FORMAT (1H ,6HTARGET,A4,45H IS TOO NEEDLE-LIKE AND LONG FOR THIS P
*  1ROGRAM)
*   GO TO 20
***

* 95 IF (SHAPE-1.0)  140,115,100    Note: 100 terminates.
* SHAPEs > 1.0 or < 0.05 are not allowed to come in to ElipGrid().

if SHAPE == 1.0
   *
   *    CIRCLE
   ASQ := A**2                          // 115
   * IF (A-TIZ)  120,120,125
   if A - TIZ <= 0.0
```

Code File: EGMCFort.Prg

```
      PROB2 := 0.0
      PROB1 := PI*ASQ/FIN
      PROB0 := 1.0-PROB1
      ***New Code, 04/07/94, JRD***
      * Handle cases where a hit prob. is > 1.0, thus making P(0) negative.
      * See relevant ELIPGRID code just above code line 435.
      PROB0 := iif(PROB0 < 0.0, 0.0, PROB0)
      ***End New Code, 04/07/94***
                                    // Top, left STOP in Fig. 7 flowchart
      * 1st return                  // (Singer and Wickman 1969)
      return(PROB0)                 // In JRD notes as STOP 2
   else
      * IF (A-BALLS)  130,135,135   // 125
      if A-BALLS < 0
         CIM   := ACOS(TIZ/A)       // 130
         PROB2 := ZAP*(ASQ*CIM-TIZ*SQRT(ASQ-0.25))/FIN
         PROB1 := PI*ASQ/FIN-2.0*PROB2
         PROB0 := 1.0-PROB1-PROB2
         ***New Code, 04/07/94, JRD***
         * Handle cases where a hit prob. is > 1.0, thus making P(0) negative.
         * See relevant ELIPGRID code just above code line 435.
         PROB0 := iif(PROB0 < 0.0, 0.0, PROB0)
         ***End New Code, 04/07/94***

         * 2nd return
         return(PROB0)
      else
         *
         *    IF THE RADIUS OF THE CIRCLE IS GREATER THAN 0.7071 THE PROBABILITY
         *    OF MISSING IS ZERO AND  PROB1 AND PROB2 ARE SET EQUAL TO  9. AS
         *    FLAGS
         *
         PROB1 := 9.0               // 135
         PROB2 := 9.0
         PROB0 := 0.0
         * 3rd return
         return(PROB0)
      endif
   endif
elseif SHAPE < 1.0
   *
   *    ELLIPSE
   *
   B := A*SHAPE                     // 140
   *
   *    B IS THE RADIUS OF THE CIRCLE IN THE TRANSFORMED NET
   *
   * IF (A-TIZ)  145,145,150
   if A-TIZ <= 0.0
      PROB1 := PI*A*B/FIN           // 145
      PROB2 := 0.0
      PROB0 := 1.0-PROB1
      ***New Code, 04/07/94, JRD***
      * Handle cases where a hit prob. is > 1.0, thus making P(0) negative.
      * See relevant ELIPGRID code just above code line 435.
      PROB0 := iif(PROB0 < 0.0, 0.0, PROB0)
      ***End New Code, 04/07/94***

      * 4th return
      return(PROB0)                 // Top, right STOP in Fig. 7 flowchart
   endif                            // (Singer and Wickman 1969)
                                    // In JRD notes as STOP 1
   ***New Code, 04/06/94, JRD***
   * Handle 0.0 angle being incremented to 0.1.
   if m->nElpGrdCor < 3
      * IF(ANGLE-0.1)  155,155,160      // 150
```

Code File: EGMCFort.Prg

```
   if ANGLE-0.1 <= 0.0
      *
      *    ALPHA IS THE ANGLE IN RADIANS
      *
      ANGLE := ANGLE+0.1                // 155
   endif
else
   * Level 3 correction below does not increment 0.0 to 0.1,
   * but does make sure the angle is positive.
   ANGLE := abs(ANGLE)
endif
***End New Code, 04/06/94***

ALPHA := ANGLE/57.295779             // 160
CNM   := 1.0-SHAPE**2
*
*    C,D,DJ1,DJ0,DM1,DM0 ARE DISTANCES BETWEEN CIRCLES IN THE
*    TRANSFORMED NET
*
C := SQRT(1.0-CNM*COS(ALPHA)**2)

* GO TO (170,165,170),NET             // 164
if NET == 2
   * 165 below.
   Y1 := 3.0+SHAPE**2-2.0*CNM*SIN(ALPHA)**2-CNM*4.0*FIN*SIN(ALPHA)*COS(ALPHA)
   D  := SQRT(Y1)*0.5000
elseif NET == 1 .or. NET == 3
   D  := SQRT(1.0-CNM*SIN(ALPHA)**2)// 170
endif

BSQU  := B**2                        // 175
FORN  := C*C
HORN  := D*D
WINE  := FIN*SHAPE
HALFC := C*0.50
HALFD := D*0.50

* IF (B-HALFC)  185,185,180          // 179
if B-HALFC > 0.0
   EOU   := ACOS(HALFC/B)            // 180
   AREA1 := 2.0*(BSQU*EOU-HALFC*SQRT(BSQU-HALFC**2))
elseif B-HALFC <= 0.0
   AREA1 := 0.0                      // 185
endif

* IF (B-HALFD)  200,200,195          // 190
if B-HALFD > 0.0
   HAI   := ACOS(HALFD/B)            // 195
   AREA2 := 2.0*(BSQU*HAI-HALFD*SQRT(BSQU-HALFD**2))
else
   AREA2 := 0.0                      // 200
endif

* IF (A-BALLS)  210,210,215          // 205
if A-BALLs <= 0.0
   PROB2 := (AREA1+AREA2)/WINE       // 210
   PROB1 := PI*BSQU/WINE-2.0*PROB2
   PROB0 := 1.0-PROB1-PROB2
   * 5th return
   ***New Code, 04/07/94, JRD***
   * Handle cases where a hit prob. is > 1.0, thus making P(0) negative.
   * See relevant ELIPGRID code just above code line 435.
   PROB0 := iif(PROB0 < 0.0, 0.0, PROB0)
   ***End New Code, 04/07/94***
                                     // Center, left STOP in Fig. 7 flowchart
   return(PROB0)                     // (Singer and Wickman 1969)
```

Code File: EGMCFort.Prg

```
                                    // In JRD notes as STOP 3
else
   * IF (ANGLE)  220,220,225        // 215
   if ANGLE <= 0.0
      C      := C+0.05              // 220
   endif
   CAROL     := C*D                 // 225

   T := ASIN(WINE/CAROL)

   * IF (ANGLE)  235,230,235
   if ANGLE == 0.0
      DJ1 := SQRT(FORN+HORN)        // 230
      DJ0 := 5.0
      *
      *    R0 IS THE RADIUS NECESSARY FOR THE TARGET TO BE HIT WITH CERTAINTY
      *
      R0 := DJ1/2.0
   else
      *I=1.0+(D*COS(T)/C)           // 235
      I := int(1.0+(D*COS(T)/C))    // 235 modified with int()
      * IF (I-1)  240,240,245
      if I-1 <= 0.0
         DJ1    := SQRT((FORN+HORN)-2.0*CAROL*COS(T))
         DJ0    := 5.0
         R0     := DJ1/(2.0*SIN(T))
      else
         XI     := I                // 245
         YI     := I-1
         DJ1    := SQRT(XI**2*FORN+HORN-2.0*XI*CAROL*COS(T))
         DJ0    := SQRT(YI**2*FORN+HORN-2.0*YI*CAROL*COS(T))
         R0     := DJ1*DJ0/(2.0*D*SIN(T))
      endif
   endif
endif

* 250 IF (B-R0)  260,255,255
if B-R0 >= 0.0
   PROB1 := 9.0                     // 255
   PROB2 := 9.0
   PROB0 := 0.0
   ***New Code, 04/07/94, JRD***
   * Handle cases where a hit prob. is > 1.0, thus making P(0) negative.
   * See relevant ELIPGRID code just above code line 435.
   PROB0 := iif(PROB0 < 0.0, 0.0, PROB0)
   ***End New Code, 04/07/94***

   * 6th return
   return(PROB0)                    // Bot., right STOP in Fig. 7 flowchart
endif                               // (Singer and Wickman 1969)
                                    // In JRD notes as STOP 4

HALFJ1 := DJ1*0.50                  // 260
HALFJ0 := DJ0*0.50

* Below is CIRCLE A on flowchart Fig. 7.  (Singer and Wickman 1969).
* IF (B-HALFJ1)  270,270,265
if B-HALFJ1 > 0.0
   GR0    := ACOS(HALFJ1/B)         // 265
   AREA3 := 2.0*(BSQU*GR0-HALFJ1*SQRT(BSQU-HALFJ1**2))
else
   AREA3 := 0.0                     // 270
endif

* 275 IF (B-HALFJ0)  285,285,280
if B-HALFJ0 > 0.0
```

Code File: EGMCFort.Prg

```
      PET   := ACOS(HALFJ0/B)          // 280
      AREA4 := 2.0*(BSQU*PET-HALFJ0*SQRT(BSQU-HALFJ0**2))
   else
      AREA4 := 0.0
   endif

   * 290 M=1.0+(2.0*D*COS(T)/C)
   M  := int(1.0+(2.0*D*COS(T)/C))     // 290 modified with int().
   YM := M-1
   XM := M

   * IF (M-1)  295,295,300
   if M-1 <= 0.0
      DM1 := SQRT(FORN+HORN*4.0-4.0*CAROL*COS(T))        // 295
      DM0 := 5.0
   else
      DM1 := SQRT(XM**2*FORN+4.0*HORN-4.0*CAROL*COS(T))  // 300
      DM0 := SQRT(YM**2*FORN+4.0*HORN-4.0*CAROL*COS(T))
   endif

   HALFM1 := DM1*0.50                  // 305
   HALFM0 := DM0*0.50
/***
      IF (HALFM1-DJ1)  310,325,310
  310 IF (HALFM1-DJ0)  315,325,315
  315 IF (HALFM0-DJ1)  320,325,320
  320 IF (HALFM0-DJ0)  330,325,330
  replaced with below:
***/
   if HALFM1 == DJ1 .or. ;
      HALFM1 == DJ0 .or. ;
      HALFM0 == DJ1 .or. ;
      HALFM0 == DJ0
      AREA5 := 0.0                     // 325
      AREA6 := 0.0
   else
      * 330 IF (B-HALFM1)  340,340,335
      if B-HALFM1 > 0.0
         YAM   := ACOS(HALFM1/B)       // 335
         AREA5 := 2.0*(BSQU*YAM-HALFM1*SQRT(BSQU-HALFM1**2))
      else
         AREA5 := 0.0                  // 340
      endif

      * 345 IF (B-HALFM0)  355,355,350
      if B-HALFM0 > 0.0
         GAME  := ACOS(HALFM0/B)       // 350
         AREA6 := 2.0*(BSQU*GAME-HALFM0*SQRT(BSQU-HALFM0**2))
      else
         AREA6 := 0.0                  // 355
      endif
   endif

   * 360 IF (B-DJ1)  370,370,365
   if B-DJ1 > 0.0
      SER := ACOS(DJ1/B)               // 365
      AREA7 := 2.0*(BSQU*SER-DJ1*SQRT(BSQU-DJ1**2))
   else
      AREA7 := 0.0                     // 370
   endif

   * 375 IF (B-DJ0)  385,385,380
   if B-DJ0 > 0.0
      AQUAR := ACOS(DJ0/B)             // 380
      AREA8 := 2.0*(BSQU*AQUAR-DJ0*SQRT(BSQU-DJ0**2))
   else
```

Code File: EGMCFort.Prg

```
      AREA8 := 0.0                        // 385
   endif

   * 390 IF (B-C)  400,400,395
   if B-C > 0.0
      POT   := ACOS(C/B)                  // 395
      AREA9 := 2.0*(BSQU*POT-C*SQRT(BSQU-FORN))
   else
      AREA9 := 0.0                        // 400
   endif

   * 405 IF (B-D)  415,415,410
   if B-D > 0.0
      TIN      := ACOS(D/B)               // 410
      AREA10   := 2.0*(BSQU*TIN-D*SQRT(BSQU-HORN))
   else
      AREA10   := 0.0                     // 415
   endif

   PROB2 := (AREA1+AREA2+AREA3+AREA4+AREA5+AREA6-AREA7-AREA8-AREA9-AREA10)/WINE
   PROB1 := PI*BSQU/WINE-2.0*PROB2-(AREA7+AREA8+AREA9+AREA10)/WINE
   PROB0 := 1.0-PROB1-PROB2
   ***New Code, 04/07/94, JRD***
   * Handle cases where a hit prob. is > 1.0, thus making P(0) negative.
   * See relevant ELIPGRID code just above code line 435.
   PROB0 := iif(PROB0 < 0.0, 0.0, PROB0)
   ***End New Code, 04/07/94***

   * 7th return
   return(PROB0)                          // Bot., right STOP in Fig. 7 flowchart
endif                                     // 2nd page (Singer and Wickman 1969)
                                          // In JRD notes as STOP 5

* Error return, should never get here.
* 8th return
return(-1)
*** End of Func: ELIPGRID()

**********************************
Function Prob0_Regr(nLtoG, nShape)
* Determine prob. of missing, P(0), by using a 4th-order polynomial regression.
* New, 11/30/94, uses 4th-order multiple regression based on Monte Carlo values
* calculated from triangular grid discontinuity region.  Regression done with
* Minitab Release 10.2.  This regression was done on the deviations of L/G and
* Shape from their respective means.
* Below describes old regression which may be forced
* with command line parameter OLR.
* The regression coefficients were determined using SigmaPlot 5.01 and data
* sets with the values near the discontinuity removed.
* Input:    nLtoG    Semi-major axis to grid size ratio.
*           nShape   Semi-minor axis to semi-major axis ratio.
* Output:   Prob0    Prob. of missing target.
* Errors:   if nLtoG or nShape out of applicable range, returns 9.
local nRtnVal := 0
local nB0, nB1, nB2, nB3, nB4          // Regression coefficients
local nB11, nB22, nB12, nB112, nB122, nB1111, nB1122, nB111, nB1112

* The 4th-order multiple regression used these means to obtain the deviations
* from the mean that became the predictor variables.
* nLtoGMean  := 0.5447103               // L/G mean from 500 values used in reg.
* nShapeMean := 0.9159746               // Shape mean from 500 values
local nLtoGDev    := nLtoG  - 0.5447103 // L/G   deviation from its mean
local nShapeDev   := nShape - 0.9159746 // Shape deviation from its mean
local nLtoGDevSq  := nLtoGDev^2         // Deviations from mean squared
local nShapeDevSq := nShapeDev^2
```

Code File: EGMCFort.Prg

```
* Check L/G ratio for correct range.
if nLtoG > 0.50 .and. nLtoG < 0.60
   * Check if command line parmameter, "OLR", passed.
   if m->lOLR == .F.
      * Use 4th-order multiple regression coeffs., i.e. default.
      * See Minitab worksheet C:\Clipper2\Editor\EGPC\Validtst\NewC500A.MTW
      * All coeffs. copied over from Minitab worksheet
      nB0    :=     0.05258302
      nB1    :=    -1.92219150
      nB2    :=    -0.70375878
      nB11   :=    18.34282112
      nB22   :=     2.59228277
      nB12   :=    10.74025822
      nB112  :=    86.09394836
      nB122  :=    26.11154175
      nB1111 :=  -464.66976929
      nB1122 :=  -725.48101807
      nB111  :=     9.29963398
      nB1112 := -1304.08276367
   else
      * OLR command line parameter forces "Old linear regr."
      do case
         case nShape >= 0.85 .and. nShape < 0.86
            * Will use regr. coeffs. calculated with nShape == 0.85.
            * Any shape < 0.85 did not appear to need regression.
            nB0 :=      0.8736
            nB1 :=     -5.8080
            nB2 :=     39.1737
            nB3 :=    -95.8914
            nB4 :=     71.2386
         case nShape >= 0.86 .and. nShape < 0.88
            * Will use regr. coeffs. calculated with nShape == 0.87.
            nB0 :=     -1.5907
            nB1 :=     13.4531
            nB2 :=    -16.2801
            nB3 :=    -26.7985
            nB4 :=     39.9151
         case nShape >= 0.88 .and. nShape < 0.92
            * Will use regr. coeffs. calculated with nShape == 0.90.
            nB0 :=     -8.7963
            nB1 :=     71.1939
            nB2 :=   -187.7169
            nB3 :=    195.8430
            nB4 :=    -66.7013
         case nShape >= 0.92 .and. nShape < 0.94
            * Will use regr. coeffs. calculated with nShape == 0.93.
            nB0 :=    -19.3100
            nB1 :=    156.3713
            nB2 :=   -443.8610
            nB3 :=    533.8017
            nB4 :=   -231.6841
         case nShape >= 0.94 .and. nShape < 0.96
            * Will use regr. coeffs. calculated with nShape == 0.95.
            nB0 :=    -27.4195
            nB1 :=    222.4814
            nB2 :=   -644.0422
            nB3 :=    800.0227
            nB4 :=   -362.8194
         case nShape >= 0.96 .and. nShape < 0.98
            * Will use regr. coeffs. calculated with nShape == 0.97.
            nB0 :=    -35.7606
            nB1 :=    290.7372
            nB2 :=   -851.5507
            nB3 :=   1077.1734
            nB4 :=   -499.9610
         case nShape >= 0.98 .and. nShape < 1.00
```

Code File: EGMCFort.Prg

```
            * Will use regr. coeffs. calculated with nShape == 0.99.
            nB0 :=    -44.0006
            nB1 :=    358.3580
            nB2 :=  -1057.7388
            nB3 :=   1353.4032
            nB4 :=   -637.0739
         otherwise
            * Error: nShape out of range.
            nRtnVal := 9
      endcase
   endif
else
   * Error: nLtoG ratio out of range.
   nRtnVal := 9
endif
if nRtnVal != 9
   if m->lOLR == .F.
      * Calculate 4th-order multi-regr. polynomial.
      nRtnVal := nB0 + nB1 *nLtoGDev + nB2 * nShapeDev + ;
         nB11   * nLtoGDevSq             + nB22   * nShapeDevSq + ;
         nB12   * nLtoGDev   * nShapeDev   + nB111  * nLtoGDev^3  + ;
         nB112  * nLtoGDevSq * nShapeDev   + nB122  * nLtoGDev   * nShapeDevSq+;
         nB1111 * nLtoGDev^4               + nB1122 * nLtoGDevSq * nShapeDevSq+;
         nB1112 * nLtoGDev^3 * nShapeDev
   else
      * Calculate 4th order polynomial.
      nRtnVal := nB0 + nB1 * nLtoG + nB2 * nLtoG^2 + nB3 * nLtoG^3 + nB4 * nLtoG^4
   endif
   * Round any neg. values up to 0.0.
   nRtnVal := iif(nRtnVal<0.0, 0.0, nRtnVal)
endif
return (nRtnVal)
*** End of Func: Prob0_Regr()

*********************************************
Function RECT(SHAPE,ANGLE,Q,REVK,REVA,REVANG)
* This function is taken from Singer's 1972 ELIPGRID program.
* It retains the original algorithm, but is modified to remove
* all goto type statements.  Many line numbers have been left in the
* comments as references back to the original code.
* Note the comment below regarding apparent error in the 1972 ELIPGRID code.
*                                                                    RECT  15
*                                                                    RECT  25
*     THIS SUBROUTINE REDUCES THE RECTANGULAR POINT NET TO A SQUARE  RECT  35
*     POINT NET WITH AN AFFINE TRANSFORMATION                        RECT  45
*                                                                    RECT  55
local AQ
local SQK
local TIS
local ALPHA
local COAL
local SIAL
local T

AQ    := Q*Q
SQK   := SHAPE**2
TIS   := AQ*SQK

***New Code, 04/06/94, JRD***
* Handle 0.0 angle being incremented to 0.1.
if m->nElpGrdCor < 3
   * IF (ANGLE-0.1)  5,5,10                                    RECT  95
   if ANGLE-0.1 <= 0.0
      ANGLE := ANGLE+0.1                  // 5
   endif
else
```

Code File: EGMCFort.Prg

```
   * Level 3 correction below does not increment 0.0 to 0.1,
   * but does make sure the angle is positive.
   ANGLE := abs(ANGLE)
   * Added ANGLE = 90° trap to level 3 correction, 04/17/94.
   ANGLE := iif(ANGLE==90.0, 89.999, ANGLE)
endif
***End New Code, 04/06/94***

ALPHA := ANGLE/57.295779                // 10
COAL  := COS(ALPHA)**2
SIAL  := SIN(ALPHA)**2
T     := SQRT(((1.0-TIS)*COAL-(AQ-SQK)*SIAL)**2+4.0*AQ*(1.0-SQK)**2*SIAL*COAL)
REVK  := ((1.0+TIS)*COAL+(AQ+SQK)*SIAL-T)/(2.0*Q*SHAPE)
* Below appears to be an error in the original code.
* See (Singer and Wickman 1969, p. 16) for the original math formula.
* REVANG=(ATAN(2.0*Q*(1.0-SQK)*TAN(ALPHA)/((AQ-SQK)*TAN(ALPHA)**2*TIRECT 175
* S-1.0))/2.0)*57.295779                                          RECT 185
if m->nElpGrdCor == 0
   * Use original formula.
   REVANG   := (ATAN(2.0*Q*(1.0-SQK)*TAN(ALPHA)/((AQ-SQK)*TAN(ALPHA)**2 * ;
      TIS-1.0))/2.0)*57.295779
else
   * Next line is corrected formula.
   REVANG   := (ATAN(2.0*Q*(1.0-SQK)*TAN(ALPHA)/(1.0-TIS-(AQ-SQK)* ;
      TAN(ALPHA)**2))/2.0)*57.295779
endif
REVA     := SQRT(SHAPE/(Q*REVK))
REVANG   := ABS(REVANG)                 // RECT 205
* The following optional code matches (Singer and Wickman 1969, 16)
* and can be used in place of line RECT 205 above.  However, no differences
* in output values were seen when testing Singer's 30 rect. grid examples after
* this code was substituted for line RECT 205 (in ELIPGRD2.FOR).
* if (tan(2.0 * REVANG) >= 0.0) then
*    REVANG = abs(REVANG)
* else
*    REVANG = 90.0 - abs(REVANG)
* endif
RETURN (NIL)
*** End of Func: RECT()

*** End of File: EGMCFort.Prg
```

Code File: EGMCMCAl.Prg

```
//=============================================================================
// File:          EGMCMCAl.Prg
// For:           EGMC.Exe, for ELIPGRID-MC Monte Carlo test program.
// Purpose:       Provides Monte Carlo algorithm.
// Author:        Jim Davidson
// Prog Started:  10/27/94 from test code.
// Last Mod:      11/13/94
//=============================================================================

// For anResults array used by MC_ProbHit() function.
#define nHIT_PROB    1                  // mean hit probability
#define nSTD_DEV     2                  // std. dev. of the mean
#define nBOUND       3                  // 99% confidence half-interval
#define nTOT_TRIALS  4                  // total trials needed

// Include files
#include "Inkey.Ch"                     // key definitions
#include "Colors.Ch"                    // Color definitions

// User-defined commands
#xcommand DEFAULT <TheParam> TO <DefaultVal> => ;
   IF (<TheParam> == NIL); <TheParam>:=<DefaultVal>; ENDIF

static nSeed := 1                       // Initialize seed for UnifRand().

*******************
Function UnifRand()
* UnifRand() is a Clipper implementation of a "minimal standard" random number
* generator.  It is based on article by Park and Miller, "Random Number
* Generators: Good Ones are Hard to Find", Comm. of the ACM, Vol. 31, No. 10.
* UnifRand() produces uniform random numbers in the open interval (0,1).
* The actual range is: 1/nMODULUS = 4.656612875 x E-10 to
*                      (nMODULUS-1)/nMODULUS = 0.9999999995.
* Park and Miller give the period as equal to nMODULUS - 1, i.e., 2,147,483,646.
#define nMULTIPLIER 16807
#define nMODULUS 2147483647
nSeed := (nMULTIPLIER * nSeed) % nMODULUS
return (nSeed/nMODULUS)
*** End of Func: UnifRand()

*******************************
Function SeedUnifRand(nNewSeed)
* Put new seed into filewide static variable, nSeed.
#define nMOD_MINUS_1 2147483646
if nNewSeed == NIL
   nSeed := 1
elseif nNewSeed < 1 .or. nNewSeed > nMOD_MINUS_1  // Range is 1 to 2^31 - 2
   nSeed := 1
else
   nSeed := nNewSeed
endif
return (NIL)
*** End of Func: SeedUnifRand()

************************************
Function TestUnifRand(lDisplayResult)
* Tests UnifRand() random number generator.
* To display results to screen, call as TestUnifRand(.T.) or TestUnifRand().
* To merely return result of test as .T. or .F., call as  TestUnifRand(.F.).
local nCount      := 0
local nDummy      := 0
local nElapTime   := 0
local nStart      := 0

* Display results if no parameter passed.
default lDisplayResult to .T.
```

Code File: EGMCMCAl.Prg

```
* Start the timer and produce 10,000 random numbers.
nStart := seconds()
SeedUnifRand(1)
for nCount = 1 to 10000
   nDummy := UnifRand()                  // nDummy makes time more realistic.
next nCount
nElapTime := seconds() - nStart

if lDisplayResult
   ?
   ?
   ? "*******************************************************"
   ? "**** Test of UnifRand() function.                  ****"
   ? "**** Based on Comm. of the ACM, Oct. 88, p. 1195. ****"
   ? "*******************************************************"
   ? " Random numbers/sec    =", 10000/(nElapTime)
   ? " Final seed should be  = 1043618065.00"
   ? " Final seed calculated =",nSeed
   @ row()+1,1 say "Test of UnifRand() is" + ;
      iif(nSeed == 1043618065, " OK.", " NOT OK.") color (m->C_HELP)
   ?
   ? " Press a key to continue..."
   inkey(0)
endif
return (nSeed == 1043618065)
*** End of Func: TestUnifRand()

*****************************************************************************
Function MC_ProbHit(nSemiMajor, nShape, nAngle, nGSize, nGTyp, nRecRatio, ;
   nMaxRuns, nNumTrials, nDsrdBound, anResults)
/*
Purpose..:  Provides Monte Carlo simulation of ELLIPGRID type
            hot spot probabilities.
Input
Variables:  nSemiMajor  Length of semi-major axis of hot spot
            nShape      Elliptical hot spot minor/major axis ratio
            nAngle      Orientation angle of hot spot to grid
            nGSize      Grid size (for rect. grid, short side)
            nRecRatio   Long/short side ratio for rectangular grids
                        nRecRatio = 1 for square grids
            nGTyp       Grid type, 1 = square, 2 = triangle, 3 = rectangle
            nMaxRuns    Maximum number of simulation runs
            nNumRuns    Number of runs actually needed to achieve error bound
            nNumTrials  Number of trials per simulation run
            nDsrdBound  Desired 99% confidence half-interval

Output
Variable.:  anResults[1] = anResults[nHIT_PROB]    is mean hit probability
            anResults[2] = anResults[nSTD_DEV]     is std. dev. of the mean
            anResults[3] = anResults[nBOUND]       is 99% confidence half-interval
            anResults[4] = anResults[nTOT_TRIALS]  is total trials used

Return
Value....:  lConverged = .T. or .F. for whether algorithm converged

Algorithm:
            * Loop through as many simulation runs as needed, up to nMaxRuns
            for nRun = 1 to nMaxRuns
               * Do each trial for a simulation run
               for nTrial = 1 to nNumTrials
                  * Get coordinates of random center
                  * Get node minimums and maximums
                  * Check nodes for a hit and count hits
               next nTrial
               ** Check if we have reached bound criteria **
            next nRun
```

Code File: EGMCMCAl.Prg

```
Output...:  nMeanProb, nStdDev, nCurBound, nTotalTrials
Note.....:  99% C.I. = nMeanProb ± nCurBound
*/
* Define grid types used with nGTyp
#define nSQR_GRID 1
#define nTRI_GRID 2
#define nREC_GRID 3

#define nSIN60 0.866025404              // sin(60°)

* Initialize and define key local variables
// Length of semi-major axis scaled by grid size
local nLtoG       := nSemiMajor/nGSize
local nNumRuns    := 0  // Number of runs actually needed to achieve error bound
local nNumHits    := 0  // Number of times ellipse covers a sample node
local nProbHitSum := 0  // Sum of hit probabilities for all runs
local nMeanProb   := 0  // Average hit probability = nProbHitSum/nRun
local nSumOfSqrs  := 0  // Numerator of standard deviation of hit probabilities
                        // nSumOfSqrs = Sum of (anProbHit[i] - nMeanProb)^2
local nStdDev     := 0  // Standard deviation of hit probabilites
                        // nStdDev = sqrt(nSumOfSqrs/(nRun - 1)
local nCurBound   := 0  // Current 99% confidence half-interval
local lConverged  := .F. // Flag for convergence
local anProbHit   := {} // Array of hit probabilities, one for each run
                        // anProbHit[i] = nNumHits/nNumTrials for ith run

* Misc. local variables
local nASq        := 0  // See below in code for variable definitions
local nBSq        := 0
local nAngleRad   := 0
local nA          := 0
local nB          := 0
local nCos        := 0
local nSin        := 0
local nMinXCnst   := 0
local nMaxXCnst   := 0
local nMinYCnst   := 0
local nMaxYCnst   := 0
local nMinX       := 0
local nMaxX       := 0
local nMinY       := 0
local nMinYRow    := 0
local nMaxY       := 0
local nMaxYRow    := 0
local nRandX      := 0
local nRandY      := 0
local nRun        := 0
local nTrial      := 0
local nTotTrials  := 0
local nYNode      := 0
local nXNode      := 0
local nXNodeT     := 0
local nXNodeTSq   := 0
local nYNodeT     := 0
local nYNodeTSq   := 0
local nDiff       := 0
local nDF         := 0
local nRowsChecked := 0
local i           := 0

* Array of Student's t scores for 99% conf. interval
local anTScores[57]
InitTScores(@anTScores)                 // Pass anTScores by reference

* Seed rand. number gen.
SeedUnifRand(1)
```

Code File: EGMCMCAl.Prg

```
* Square of the scaled semi-major axis
nASq        := nLtoG * nLtoG
* Square of the scaled semi-minor axis
nBSq        := (nLtoG * nShape) * (nLtoG * nShape)

* Constants needed to speed up finding min/max X,Y nodes
nAngleRad   := dtor(nAngle)           // Angle in radians
nA          := nLtoG                  // Scaled semi-major axis
nB          := nA * nShape            // Scaled semi-minor axis
nCos        := cos(nAngleRad)
nSin        := sin(nAngleRad)
nMinXCnst   := -nA * nCos - nB * nSin  // See Sept. 94 Log p. 67.
nMaxXCnst   :=  nA * nCos + nB * nSin
nMinYCnst   := -nA * nSin - nB * nCos
nMaxYCnst   :=  nA * nSin + nB * nCos

* Loop through as many simulation runs as needed        Random
for nRun = 1 to nMaxRuns                 //             center
   // nNumHits is for each run           //             . (h,k)
   nNumHits := 0                         //   ----------+----------
                                         //             |
   * Do each trial for a simulation run
   for nTrial = 1 to nNumTrials
      * Check for Esc key abort
      if inkey() == K_ESC
         anResults[nTOT_TRIALS] := -1
         return (.F.)
      endif

      nTotTrials++
      * Get coordinates of random center
      if nGTyp != nTRI_GRID
         * For rectangles, squares (for squares, nRecRatio == 1)
         nRandX := UnifRand() * nRecRatio  // h in math lingo
         nRandY := UnifRand()              // k in math lingo
      else
         * For triangular grids
         * nRandX will vary from ≈ 0.0 to ≈ 1.0
         * nRandY will vary from ≈ 0.0 to ≈ 0.866 (peak of tri. cell)
         nRandX := UnifRand()
         nRandY := nSIN60 * UnifRand()
      endif

      * Get node minimums and maximums
      * See Sept. 94 Log p. 67.
      if nGTyp == nSQR_GRID
         * For square grids
         nMinX := ceiling(nMinXCnst + nRandX)
         nMaxX := floor(nMaxXCnst + nRandX)
         nMinY := ceiling(nMinYCnst + nRandY)
         nMaxY := floor(nMaxYCnst + nRandY)
      elseif nGTyp == nTRI_GRID
         * For tri grids, see Oct. 94 Log p. 98.
         nMinYRow := int((nMinYCnst + nRandY)/nSIN60)
         nMinY    := nMinYRow * nSIN60
         nMaxYRow := int((nMaxYCnst + nRandY)/nSIN60)
         nMaxY    := nMaxYRow * nSIN60
         nMinX    := int((nMinXCnst + nRandX)/0.5) * 0.5
         * If min Y row is even and min X is not an integer, add 0.5.
         if nMinYRow % 2 == 0 .and. int(nMinX) != nMinX
            nMinX += 0.5
         endif
         nMaxX := int((nMaxXCnst + nRandX)/0.5) * 0.5
      elseif nGTyp == nREC_GRID
         * For rect grids
         nMinX := ceiling((nMinXCnst + nRandX) / nRecRatio) * nRecRatio
```

Code File: EGMCMCAl.Prg

```
   nMaxX := floor(  (nMaxXCnst + nRandX) / nRecRatio ) * nRecRatio
   nMinY := ceiling(nMinYCnst + nRandY)
   nMaxY := floor(nMaxYCnst + nRandY)
else
   * Grid type error
   @0,0
   @0,0 say " Error in MC_ProbHit(): Invalid grid type.  " + ;
      "Results not valid.  Press a key..." color(m->C_Error)
   inkey(0)
endif

* Check nodes for a hit
* First see if a hit is possible
if nMaxX < nMinX .or. nMaxY < nMinY
   * If here, hot spot is between grid lines, can not be hit
   * Just loop back to next trial
   loop
endif

* Now sequence through all possible nodes, looking for a hit
* Break out of sequence when a hit occurs
if nGTyp != nTRI_GRID
   *--- For square and rectangular grids ---*
   begin sequence
      for nXNode = nMinX to nMaxX step nRecRatio
         for nYNode = nMinY to nMaxY
            * Transform nodes x-coord. to X',Y' translated and rotated axis
            * See "Math Handbook" by Spiegel p. 36
            nXNodeT := ((nXNode - nRandX) * nCos)+((nYNode - nRandY) *nSin)
            nXNodeTSq := nXNodeT * nXNodeT
            * Transform nodes y-coord. to X',Y' translated and rotated axis
            nYNodeT := ((nYNode - nRandY) * nCos)-((nXNode - nRandX) *nSin)
            nYNodeTSq := nYNodeT * nYNodeT

            *=== Check if current node is inside hot spot ===*
            if nXNodeTSq/nASq + nYNodeTSq/nBSq <= 1
               nNumHits++
               * Break out of node checking sequence and do
               * another trial
               break(NIL)
            endif
         next nYNode
      next nXNode
   end sequence // for square and rectangular grids
else
   *--- For triangular grids ---*
   * Now sequence through all possible nodes, looking for a hit
   * Break out of sequence when a hit occurs
   nRowsChecked := 0
   begin sequence
      for nYNode = nMinY to nMaxY step nSIN60
         * As we move up the grid from row to row, the X coordinates of
         * the nodes alternate by 0.5.  On first pass, when nRowsChecked
         * is 0, nMinX is not altered.  Thereafter, it alternates by 0.5.
         * See 11/2/94 JRD Log, p. 98.
         if nRowsChecked > 0
            if nRowsChecked % 2 == 0
               nMinX += 0.5
            else
               nMinX -= 0.5
            endif
         endif
         nRowsChecked++

         for nXNode = nMinX to nMaxX
            * Transform nodes x-coord. to X',Y' translated and rotated axis
```

Code File: EGMCMCAl.Prg

```
                * See "Math Handbook" by Spiegel p. 36
                nXNodeT := ((nXNode - nRandX) * nCos)+((nYNode - nRandY) *nSin)
                nXNodeTSq := nXNodeT * nXNodeT
                * Transform nodes y-coord. to X',Y' translated and rotated axis
                nYNodeT := ((nYNode - nRandY) * nCos)-((nXNode - nRandX) *nSin)
                nYNodeTSq := nYNodeT * nYNodeT

                *=== Check if current node is inside hot spot ===*
                if nXNodeTSq/nASq + nYNodeTSq/nBSq <= 1
                   nNumHits++
                   * Break out of node checking sequence and do
                   * another trial
                   break(NIL)
                endif
             next nXNode
          next nYNode
       end sequence // for triangular grids
    endif

 next nTrial

 aadd(anProbHit, nNumHits/nNumTrials)   // The number of trials is fixed
 nProbHitSum += anProbHit[nRun]         // for each run.

 ** Check if we have reached bound criteria **
 if nRun >= 10  // Do at least 10 runs
    * Get current mean prob. of a hit
    nMeanProb := nProbHitSum/nRun
    * Get current sum of squared differences
    nSumOfSqrs := 0                 // Initialize sum of squares
    for i = 1 to nRun
       nDiff      := (anProbHit[i] - nMeanProb)
       nSumOfSqrs += nDiff * nDiff
    next i
    * Get current standard dev. of hit probabilities
    nStdDev := sqrt(nSumOfSqrs/(nRun-1))

    * Get degrees of freedom for t score array
    nDF := nRun - 1
    if nDF < 50
       * No change needed, nDF will index t score array
    elseif nDF >= 50 .and. nDF < 100
       * Get t scores for df = 50 to 99
       * 50 to 59 uses t score for 50
       * 60 to 69 uses t score for 60, etc.
       nDF := 50 + int( (nDF - 50)/10 )
    elseif nDF >= 200
       * anTScores[57] is for df = infinity
       nDF := 57
    elseif nDF >= 120
       * anTScores[56] is for df = 120
       nDF := 56
    elseif nDF >= 100
       * anTScores[55] is for df = 100
       nDF := 55
    endif
    nCurBound := anTScores[nDF] * nStdDev/sqrt(nRun)

    ** Key check for bound convergence **
   if nCurBound <= nDsrdBound .and. nRun >= 10
       * Done, no more runs needed
       lConverged := .T.
       exit
    endif
 endif
next nRun
```

Code File: EGMCMCAl.Prg

```
* Set output values
anResults[nHIT_PROB]    := nMeanProb
anResults[nSTD_DEV]     := nStdDev
anResults[nBOUND]       := nCurBound
anResults[nTOT_TRIALS]  := nTotTrials
return(lConverged)
*** End of Func: MC_ProbHit()

*****************************
Function InitTScores(anArray)
/* Initialize Student's t scores for 99% conf. interval.
 * Values are two-tailed values for alpha = 1% with 0.5% in each tail.
 * Based on Table A in "Applied Linear Regression" by S. Weisberg, 1985.
 * Deg. of freedom = 1 to 50, and selected values to 120.
 * Row  1 =   1 deg. of freedom
 * Row  2 =   2 deg. of freedom, etc.
 * ...
 * Row 50 =  50 deg. of freedom
 * Row 51 =  60 deg. of freedom, etc.
 * Row 56 = 120 deg. of freedon
 * Row 57 = infinite deg. of freedom
 */

anArray :=   ;                          // Deg. freedom
     {63.66, ;                          //  1
       9.92, ;
       5.84, ;
       4.60, ;
       4.03, ;                          //  5
       3.71, ;
       3.50, ;
       3.36, ;
       3.25, ;
       3.17, ;                          // 10
       3.11, ;
       3.05, ;
       3.01, ;
       2.98, ;
       2.95, ;                          // 15
       2.92, ;
       2.90, ;
       2.88, ;
       2.86, ;
       2.85, ;                          // 20
       2.83, ;
       2.82, ;
       2.81, ;
       2.80, ;
       2.79, ;                          // 25
       2.78, ;
       2.77, ;
       2.76, ;
       2.76, ;
       2.75, ;                          // 30
       2.74, ;
       2.74, ;
       2.73, ;
       2.73, ;
       2.72, ;                          // 35
       2.72, ;
       2.72, ;
       2.71, ;
       2.71, ;
       2.70, ;                          // 40
       2.70, ;
       2.70, ;
```

Code File: EGMCMCAl.Prg

```
      2.70, ;
      2.69, ;
      2.69, ;                          // 45
      2.69, ;
      2.68, ;
      2.68, ;
      2.68, ;
      2.68, ;                          // 50
      2.66, ;                          // 60
      2.65, ;                          // 70
      2.64, ;                          // 80
      2.63, ;                          // 90
      2.63, ;                          // 100
      2.62, ;                          // 120
      2.58 }                           // infinity
return (NIL)
*** End of Func: InitTScores()

*** End of File: EGMCMCAl.Prg
```

INTERNAL DISTRIBUTION

1. B. A. Berven
2. B. Coleman
3 - 22. J. R. Davidson
23. P. V. Egidi
24. D. K. Halford
25. A. D. Laase
26 - 31. C. A. Little
32. J. M. Morris
33. P. T. Owen
34. G. H. Stevens
35. J. E. Wilson
36. Central Research Library
37 - 38. Laboratory Records
39. Laboratory Records - RC
40. ORNL Patent Section
41. ORNL Technical Library, Y-12

EXTERNAL DISTRIBUTION

42. Kathy Allen, Illinois Department of Nuclear Safety, 1035 Outer Park Dr., Springfield, IL 62704
43. Heather Baldy, Bechtel, 151 Lafayette Dr., Oak Ridge, TN 37831
44. Jim Berger, 412 Executive Drive, Suite 402, Knoxville, TN 37923.
45. Malcom J. Bertoni, Research Triangle Institute, Suite 740, Washington D.C. 20036
46. James A. Bowers, Westinghouse Savannah River Company, PO Box 616, Aiken, SC 29802
47. C. C. Britton, Mesa State College, PO Box 2647, Grand Junction, CO 81502
48. Bruce Buxton, Battelle, 505 King Ave., Columbus, OH 43201
49. Steve Conner, NUS Savannah River Center, 900 Trail Ridge Rd., Aiken, SC 29803-5297
50. Robert O. Devany, Weiss Associates, 5500 Shellmound St., Emeryville, CA 94608
51. Rich Engelder, RUST Geotech, Inc., PO Box 14000, Grand Junction, CO 81502

52. Richard O. Gilbert, Pacific Northwest Laboratory, ISB 1 Building, PO Box 999, Richland, WA 99352

53. Carl Gogolak, U.S. Department of Energy, Environmental Measurements Lab., 376 Hudson Street, New York, NY 10014-3621

54. Nancy Hassig, Suite 305, 625 Ellis St., Mountain View, CA 94043

55. Tim LeGore, Westinghouse Hanford Company, PO Box 1970, Richland, WA 99352

56. Don Mackenzie, U.S. Department of Energy/HQ, EM-442, Quince Orchard Bldg., Washington, DC 20585-0002

57. David E. Mathes, U.S. Department of Energy, Office of ER, EM-451 (GTN) Room D-427, Washington, DC 20545

58. T. J. Novotny, Mesa State College, PO Box 2647, Grand Junction, CO 81502

59. Jack Parks, Building 372, Argonne National Laboratory, Argonne, IL 60439

60. James D. Paulson, U.S. Department of Energy, 9800 S. Cass Ave., Argonne, IL 60439

61. Al Robinson, Northwest Instrument Systems, Inc., 3100 George Washington Way, Richland, WA 99352

62. Jim Rumbaugh, Geraghty & Miller, Inc., 10700 Park Ridge Blvd., Suite 600, Reston, VA 22091

63. Donald A. Singer, U.S. Geological Survey, 345 Middlefield Road, Menlo Park, CA 94025

64. Robert L. Stamnes, U.S. Environmental Protection Agency, 1200 Sixth Ave., Seattle, WA 98101

65. Andrew Wallo, III, U.S. Department of Energy, Air, Water & Radiation Division, EH-232, 1000 Independence Avenue, SW, Washington, DC 20585

66. Erik K. Webb, Sandia National Laboratories, Dept. 6331, Albuquerque, NM 87185-5800

67. Office of Assistant Manager, Energy Research and Development, Oak Ridge Operations Office, P.O. Box 2001, Oak Ridge, TN 37831-8600

68 - 69. Office of Scientific and Technical Information, U.S. Department of Energy, P.O. Box 62, Oak Ridge, TN 37831