NAG Library Function Document

nag_rand_field_2d_predef_setup (g05zrc)

1 Purpose

nag_rand_field_2d_predef_setup (g05zrc) performs the setup required in order to simulate stationary Gaussian random fields in two dimensions, for a preset variogram, using the circulant embedding method. Specifically, the eigenvalues of the extended covariance matrix (or embedding matrix) are calculated, and their square roots output, for use by nag_rand_field_2d_generate (g05zsc), which simulates the random field.

2 Specification

```c
#include <nag.h>
#include <nagg05.h>

void nag_rand_field_2d_predef_setup (const Integer ns[], double xmin, double xmax, double ymin, double ymax, const Integer maxm[], double var, Nag_Variogram cov, Nag_NormType norm, Integer np, const double params[], double xx[], double yy[], Integer m[], Integer *approx, double *rho, Integer *icount, double eig[], NagError *fail)
```

3 Description

A two-dimensional random field \( Z(x) \) in \( \mathbb{R}^2 \) is a function which is random at every point \( x \in \mathbb{R}^2 \), so \( Z(x) \) is a random variable for each \( x \). The random field has a mean function \( \mu(x) = \mathbb{E}[Z(x)] \) and a symmetric positive semidefinite covariance function \( C(x,y) = \mathbb{E}[(Z(x) - \mu(x))(Z(y) - \mu(y))] \). \( Z(x) \) is a Gaussian random field if for any choice of \( n \in \mathbb{N} \) and \( x_1, \ldots, x_n \in \mathbb{R}^2 \), the random vector \( [Z(x_1), \ldots, Z(x_n)]^T \) follows a multivariate Normal distribution, which would have a mean vector \( \tilde{\mu} \) with entries \( \tilde{\mu}_i = \mu(x_i) \) and a covariance matrix \( \tilde{C} \) with entries \( \tilde{C}_{ij} = C(x_i, x_j) \). A Gaussian random field \( Z(x) \) is stationary if \( \mu(x) \) is constant for all \( x \in \mathbb{R}^2 \) and \( C(x,y) = C(x+a,y+a) \) for all \( x,y,a \in \mathbb{R}^2 \) and hence we can express the covariance function \( C(x,y) \) as a function \( \gamma \) of one variable: \( C(x,y) = \gamma(x-y) \). \( \gamma \) is known as a variogram (or more correctly, a semivariogram) and includes the multiplicative factor \( \sigma^2 \) representing the variance such that \( \gamma(0) = \sigma^2 \).

The functions nag_rand_field_2d_predef_setup (g05zrc) and nag_rand_field_2d_generate (g05zsc) are used to simulate a two-dimensional stationary Gaussian random field, with mean function zero and variogram \( \gamma(x) \), over a domain \( [x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}] \), using an equally spaced set of \( N_1 \times N_2 \) points; \( N_1 \) points in the \( x \)-direction and \( N_2 \) points in the \( y \)-direction. The problem reduces to sampling a Gaussian random vector \( X \) of size \( N_1 \times N_2 \), with mean vector zero and a symmetric covariance matrix \( A \), which is an \( N_2 \times N_2 \) block Toeplitz matrix with Toeplitz blocks of size \( N_1 \times N_1 \). Since \( A \) is in general expensive to factorize, a technique known as the circulant embedding method is used. \( A \) is embedded into a larger, symmetric matrix \( B \), which is an \( M_2 \times M_2 \) block circulant matrix with circulant blocks of size \( M_1 \times M_1 \), where \( M_1 \geq 2(N_1 - 1) \) and \( M_2 \geq 2(N_2 - 1) \). \( B \) can now be factorized as \( B = WAW^* = R^*R \), where \( W \) is the two-dimensional Fourier matrix (\( W^* \) is the complex conjugate of \( W \)), \( A \) is the diagonal matrix containing the eigenvalues of \( B \) and \( R = A^{1/2} \). \( B \) is known as the embedding matrix. The eigenvalues can be calculated by performing a discrete Fourier transform of the first row (or column) of \( B \) and multiplying by \( M_1 \times M_2 \), and so only the first row (or column) of \( B \) is needed – the whole matrix does not need to be formed.

As long as all of the values of \( A \) are non-negative (i.e., \( B \) is positive semidefinite), \( B \) is a covariance matrix for a random vector \( Y \) which has \( M_2 \) blocks of size \( M_1 \). Two samples of \( Y \) can now be simulated from the real and imaginary parts of \( R^*(U + iV) \), where \( U \) and \( V \) have elements from the standard Normal distribution. Since \( R^*(U + iV) = WAW^{1/2}(U + iV) \), this calculation can be done using a discrete Fourier transform of the vector \( A^{1/2}(U + iV) \). Two samples of the random vector \( X \) can now be recovered.
by taking the first \( N_1 \) elements of the first \( N_2 \) blocks of each sample of \( Y \) – because the original covariance matrix \( A \) is embedded in \( B \), \( X \) will have the correct distribution.

If \( B \) is not positive semidefinite, larger embedding matrices \( B \) can be tried; however if the size of the matrix would have to be larger than \( \text{maxm} \), an approximation procedure is used. We write \( A = A_+ + A_- \), where \( A_+ \) and \( A_- \) contain the non-negative and negative eigenvalues of \( B \) respectively. Then \( B \) is replaced by \( \rho B_+ \) where \( B_+ = W A_+ W^\top \) and \( \rho \in (0, 1] \) is a scaling factor. The error \( \epsilon \) in approximating the distribution of the random field is given by

\[
\epsilon = \sqrt{(1 - \rho)^2 \text{trace } A + \rho^2 \text{trace } A_-}
\]

The error \( \epsilon \) in approximating the distribution of the random field is given by

Three choices for \( \rho \) are available, and are determined by the input argument \( \text{corr} \):

- setting \( \text{corr} = \text{Nag EmbedScaleTraces} \) sets
  \[
  \rho = \frac{\text{trace } A}{\text{trace } A_+},
  \]
- setting \( \text{corr} = \text{Nag EmbedScaleSqrtTraces} \) sets
  \[
  \rho = \sqrt{\frac{\text{trace } A}{\text{trace } A_+}},
  \]
- setting \( \text{corr} = \text{Nag EmbedScaleOne} \) sets \( \rho = 1 \).

\text{nag_rand_field_2d_predef_setup (g05zrc)} finds a suitable positive semidefinite embedding matrix \( B \) and outputs information regarding the accuracy of the approximation is output. Note that only the first row (or column) of \( B \) is actually formed and stored.

### 4 References


Schlather M (1999) Introduction to positive definite functions and to unconditional simulation of random fields \( \text{Technical Report ST 99–10} \) Lancaster University


### 5 Arguments

1: \( \text{ns}[2] \) – const Integer

\( \text{Input} \)

\( On \ entry: \) the number of sample points to use in each direction, with \( \text{ns}[0] \) sample points in the \( x \)-direction, \( N_1 \) and \( \text{ns}[1] \) sample points in the \( y \)-direction, \( N_2 \). The total number of sample points on the grid is therefore \( \text{ns}[0] \times \text{ns}[1] \).

\( Constraints: \)

\[
\text{ns}[0] \geq 1;
\text{ns}[1] \geq 1.
\]

2: \( \text{xmin} \) – double

\( \text{Input} \)

\( On \ entry: \) the lower bound for the \( x \)-coordinate, for the region in which the random field is to be simulated.

\( Constraint: \) \( \text{xmin} < \text{xmax} \).
3: \textbf{xmax} – double \hspace{1cm} \textit{Input}

\textit{On entry:} the upper bound for the \textit{x}-coordinate, for the region in which the random field is to be simulated.

\textit{Constraint:} xmin < xmax.

4: \textbf{ymin} – double \hspace{1cm} \textit{Input}

\textit{On entry:} the lower bound for the \textit{y}-coordinate, for the region in which the random field is to be simulated.

\textit{Constraint:} ymin < ymax.

5: \textbf{ymax} – double \hspace{1cm} \textit{Input}

\textit{On entry:} the upper bound for the \textit{y}-coordinate, for the region in which the random field is to be simulated.

\textit{Constraint:} ymin < ymax.

6: \textbf{maxm[2]} – const Integer \hspace{1cm} \textit{Input}

\textit{On entry:} determines the maximum size of the circulant matrix to use – a maximum of maxm[0] elements in the \textit{x}-direction, and a maximum of maxm[1] elements in the \textit{y}-direction. The maximum size of the circulant matrix is thus maxm[0] × maxm[1].

\textit{Constraint:} maxm[i] ≥ 2^k, where \( k \) is the smallest integer satisfying \( 2^k ≥ 2(\text{ns}[i] - 1) \), for \( i = 0, 1 \).

7: \textbf{var} – double \hspace{1cm} \textit{Input}

\textit{On entry:} the multiplicative factor \( \sigma^2 \) of the variogram \( \gamma(x) \).

\textit{Constraint:} var ≥ 0.0.

8: \textbf{cov} – Nag_Variogram \hspace{1cm} \textit{Input}

\textit{On entry:} determines which of the preset variograms to use. The choices are given below. Note that \( x' = \|\frac{x}{\ell_1}, \frac{y}{\ell_2}\| \), where \( \ell_1 \) and \( \ell_2 \) are correlation lengths in the \textit{x} and \textit{y} directions respectively and are parameters for most of the variograms, and \( \sigma^2 \) is the variance specified by var.

\textbf{cov} = Nag_VgmSymmStab

\textit{Symmetric stable variogram}

\[ \gamma(x) = \sigma^2 \exp(-x'^2), \]

where

\[ \ell_1 = \text{params}[0], \ell_1 > 0, \]
\[ \ell_2 = \text{params}[1], \ell_2 > 0, \]
\[ \nu = \text{params}[2], 0 < \nu \leq 2. \]

\textbf{cov} = Nag_VgmCauchy

\textit{Cauchy variogram}

\[ \gamma(x) = \sigma^2 \left(1 + x'^2\right)^{-\nu}, \]

where

\[ \ell_1 = \text{params}[0], \ell_1 > 0, \]
\[ \ell_2 = \text{params}[1], \ell_2 > 0, \]
\[ \nu = \text{params}[2], \nu > 0. \]
\textbf{cov} = \texttt{Nag\_VgmDifferential} \\
Differential variogram with compact support
\[
\gamma(x) = \begin{cases} 
\sigma^2 \left( 1 + 8x' + 25(x')^2 + 32(x')^3 \right)(1 - x')^8, & x' < 1, \\
0, & x' \geq 1,
\end{cases}
\]
where
\[
\ell_1 = \text{params}[0], \ell_1 > 0, \\
\ell_2 = \text{params}[1], \ell_2 > 0.
\]
\textbf{cov} = \texttt{Nag\_VgmExponential} \\
Exponential variogram
\[
\gamma(x) = \sigma^2 \exp(-x'),
\]
where
\[
\ell_1 = \text{params}[0], \ell_1 > 0, \\
\ell_2 = \text{params}[1], \ell_2 > 0.
\]
\textbf{cov} = \texttt{Nag\_VgmGauss} \\
Gaussian variogram
\[
\gamma(x) = \sigma^2 \exp\left(-\left(x'\right)^2\right),
\]
where
\[
\ell_1 = \text{params}[0], \ell_1 > 0, \\
\ell_2 = \text{params}[1], \ell_2 > 0.
\]
\textbf{cov} = \texttt{Nag\_VgmNugget} \\
Nugget variogram
\[
\gamma(x) = \begin{cases} 
\sigma^2, & x = 0, \\
0, & x \neq 0.
\end{cases}
\]
No parameters need be set for this value of \textbf{cov}.
\textbf{cov} = \texttt{Nag\_VgmSpherical} \\
Spherical variogram
\[
\gamma(x) = \begin{cases} 
\sigma^2 \left( 1 - 1.5x' + 0.5(x')^3 \right), & x' < 1, \\
0, & x' \geq 1,
\end{cases}
\]
where
\[
\ell_1 = \text{params}[0], \ell_1 > 0, \\
\ell_2 = \text{params}[1], \ell_2 > 0.
\]
\textbf{cov} = \texttt{Nag\_VgmBessel} \\
Bessel variogram
\[
\gamma(x) = \sigma^2 \frac{2\nu \Gamma(\nu + 1) J_{\nu}(x')}{(x')^\nu},
\]
where
\[
J_{\nu}(\cdot) \text{ is the Bessel function of the first kind,} \\
\ell_1 = \text{params}[0], \ell_1 > 0, \\
\ell_2 = \text{params}[1], \ell_2 > 0, \\
\nu = \text{params}[2], \nu \geq 0.
\]
\[
\gamma(x) = \sigma^2 \frac{\sin(x')}{x'},
\]

where

\[
\ell_1 = \text{params}[0], \ \ell_1 > 0,
\]
\[
\ell_2 = \text{params}[1], \ \ell_2 > 0.
\]

\[
\gamma(x) = \sigma^2 \frac{2^{1-v}(x')^v K_v(x')}{\Gamma(v)},
\]

where

\[K_v(\cdot)\] is the modified Bessel function of the second kind,

\[
\ell_1 = \text{params}[0], \ \ell_1 > 0,
\]
\[
\ell_2 = \text{params}[1], \ \ell_2 > 0,
\]
\[
\nu = \text{params}[2], \ \nu > 0.
\]

\[
\gamma(x) = \sigma^2 \frac{2^{1-v}(x')^v K_v(x')}{\Gamma(v)} \left(1 + 8x'' + 25(x'')^2 + 32(x'')^3 \right) \left(1 - x'' \right)^8, \ \ x'' < 1,
\]
\[
0, \ \ x'' \geq 1,
\]

where

\[
x'' = \left\| \frac{x'}{\ell_1 s_1}, \frac{x'}{\ell_2 s_2} \right\|
\]

\[K_v(\cdot)\] is the modified Bessel function of the second kind,

\[
\ell_1 = \text{params}[0], \ \ell_1 > 0,
\]
\[
\ell_2 = \text{params}[1], \ \ell_2 > 0,
\]
\[
s_1 = \text{params}[2], \ s_1 > 0,
\]
\[
s_2 = \text{params}[3], \ s_2 > 0,
\]
\[
\nu = \text{params}[4], \ \nu > 0.
\]

\[
\gamma(x) = \sigma^2 \frac{\left(\delta^2 + (x')^2\right)^{\frac{v}{2}}}{\delta^v K_v(\kappa \delta)} K_\lambda \left(\kappa \left(\delta^2 + (x')^2\right)^{\frac{v}{2}}\right),
\]

where

\[K_v(\cdot)\] is the modified Bessel function of the second kind,

\[
\ell_1 = \text{params}[0], \ \ell_1 > 0,
\]
\[
\ell_2 = \text{params}[1], \ \ell_2 > 0,
\]
\[
\lambda = \text{params}[2], \ \text{no constraint on } \lambda,
\]
\[
\delta = \text{params}[3], \ \delta > 0,
\]
\( \kappa = \text{params}[4], \kappa > 0. \)

\textit{Constraint}: \( \text{cov} = \text{Nag\_VgmSymmStab}, \text{Nag\_VgmCauchy}, \text{Nag\_VgmDifferential}, \text{Nag\_VgmExponential}, \text{Nag\_VgmGauss}, \text{Nag\_VgmNugget}, \text{Nag\_VgmSpherical}, \text{Nag\_VgmBessel}, \text{Nag\_VgmHole}, \text{Nag\_VgmWhittleMatern}, \text{Nag\_VgmContParam} \) or \( \text{Nag\_VgmGenHyp} \).

9: \[ \text{norm} \quad \text{Nag\_NormType} \quad \text{Input} \]

\textit{On entry}: determines which norm to use when calculating the variogram.

- \text{norm} = \text{Nag\_OneNorm}
  - The 1-norm is used, i.e., \( \|x, y\| = |x| + |y| \).
- \text{norm} = \text{Nag\_TwoNorm}
  - The 2-norm (Euclidean norm) is used, i.e., \( \|x, y\| = \sqrt{x^2 + y^2} \).

\textit{Suggested value}: \text{norm} = \text{Nag\_TwoNorm}.

\textit{Constraint}: \( \text{norm} = \text{Nag\_OneNorm} \) or \( \text{Nag\_TwoNorm} \).

10: \[ \text{np} \quad \text{Integer} \quad \text{Input} \]

\textit{On entry}: the number of parameters to be set. Different covariance functions need a different number of parameters.

- \text{cov} = \text{Nag\_VgmNugget} \quad \text{np} \text{ must be set to 0.}
- \text{cov} = \text{Nag\_VgmDifferential}, \text{Nag\_VgmExponential}, \text{Nag\_VgmGauss}, \text{Nag\_VgmSpherical} or \text{Nag\_VgmHole} \quad \text{np} \text{ must be set to 2.}
- \text{cov} = \text{Nag\_VgmSymmStab}, \text{Nag\_VgmCauchy}, \text{Nag\_VgmBessel} or \text{Nag\_VgmWhittleMatern} \quad \text{np} \text{ must be set to 3.}
- \text{cov} = \text{Nag\_VgmContParam} \text{ or } \text{Nag\_VgmGenHyp} \quad \text{np} \text{ must be set to 5.}

11: \[ \text{params[np]} \quad \text{const double} \quad \text{Input} \]

\textit{On entry}: the parameters for the variogram as detailed in the description of \text{cov}.

\textit{Constraint}: see \text{cov} for a description of the individual parameter constraints.

12: \[ \text{pad} \quad \text{Nag\_EmbedPad} \quad \text{Input} \]

\textit{On entry}: determines whether the embedding matrix is padded with zeros, or padded with values of the variogram. The choice of padding may affect how big the embedding matrix must be in order to be positive semidefinite.

- \text{pad} = \text{Nag\_EmbedPadZeros}
  - The embedding matrix is padded with zeros.
- \text{pad} = \text{Nag\_EmbedPadValues}
  - The embedding matrix is padded with values of the variogram.

\textit{Suggested value}: \text{pad} = \text{Nag\_EmbedPadValues}.

\textit{Constraint}: \text{pad} = \text{Nag\_EmbedPadZeros} \text{ or } \text{Nag\_EmbedPadValues}.

13: \[ \text{corr} \quad \text{Nag\_EmbedScale} \quad \text{Input} \]

\textit{On entry}: determines which approximation to implement if required, as described in Section 3.

\textit{Suggested value}: \text{corr} = \text{Nag\_EmbedScaleTraces}.

\textit{Constraint}: \text{corr} = \text{Nag\_EmbedScaleTraces} \text{, Nag\_EmbedScaleSqrtTraces} \text{ or } \text{Nag\_EmbedScaleOne}.
14: $\text{lamb}[\text{maxm}[0] \times \text{maxm}[1]]$ – double

On exit: contains the square roots of the eigenvalues of the embedding matrix.

15: $\text{xx}[\text{ns}[0]]$ – double

On exit: the points of the $x$-coordinates at which values of the random field will be output.

16: $\text{yy}[\text{ns}[1]]$ – double

On exit: the points of the $y$-coordinates at which values of the random field will be output.

17: $\text{m}[2]$ – integer

On exit: $\text{m}[0]$ contains $M_1$, the size of the circulant blocks and $\text{m}[1]$ contains $M_2$, the number of blocks, resulting in a final square matrix of size $M_1 \times M_2$.

18: $\text{approx}$ – integer *

On exit: indicates whether approximation was used.

$\text{approx} = 0$

No approximation was used.

$\text{approx} = 1$

Approximation was used.

19: $\text{rho}$ – double *

On exit: indicates the scaling of the covariance matrix. $\text{rho} = 1.0$ unless approximation was used with $\text{corr} = \text{NagEmbedScaleTraces}$ or $\text{NagEmbedScaleSqrtTraces}$.

20: $\text{icount}$ – integer *

On exit: indicates the number of negative eigenvalues in the embedding matrix which have had to be set to zero.

21: $\text{eig}[3]$ – double

On exit: indicates information about the negative eigenvalues in the embedding matrix which have had to be set to zero. $\text{eig}[0]$ contains the smallest eigenvalue, $\text{eig}[1]$ contains the sum of the squares of the negative eigenvalues, and $\text{eig}[2]$ contains the sum of the absolute values of the negative eigenvalues.

22: $\text{fail}$ – NagError *

The NAG error argument (see Section 3.6 in the Essential Introduction).

6 Error Indicators and Warnings

**NE_ALLOC_FAIL**

Dynamic memory allocation failed.

See Section 3.2.1.2 in the Essential Introduction for further information.

**NE_BAD_PARAM**

On entry, argument $\langle$value$\rangle$ had an illegal value.

**NE_ENUM_INT**

On entry, $\text{np} = \langle$value$\rangle$.

Constraint: for $\text{cov} = \langle$value$\rangle$, $\text{np} = \langle$value$\rangle$. 
NE_ENUM_REAL_1
On entry, params\(\langle value\rangle\rangle = \langle value\rangle\).  
Constraint: dependent on cov, see documentation.

NE_INT_ARRAY
On entry, maxm = \(\langle value\rangle, \langle value\rangle\rangle\).  
Constraint: the minimum calculated value for maxm are \(\langle value\rangle, \langle value\rangle\rangle\).  
Where the minima of maxm\(i - 1\) is given by \(2^k\), where \(k\) is the smallest integer satisfying 
\(2^k \geq 2(ns[i - 1] - 1)\), for \(i = 1, 2\).  
On entry, ns = \(\langle value\rangle, \langle value\rangle\rangle\).  
Constraint: ns\(0 \geq 1\), ns\(1 \geq 1\).

NE_INTERNAL_ERROR
An internal error has occurred in this function. Check the function call and any array sizes. If the 
call is correct then please contact NAG for assistance.  
An unexpected error has been triggered by this function. Please contact NAG.  
See Section 3.6.6 in the Essential Introduction for further information.

NE_NO_LICENCE
Your licence key may have expired or may not have been installed correctly.  
See Section 3.6.5 in the Essential Introduction for further information.

NE_REAL
On entry, var = \(\langle value\rangle\).  
Constraint: var \(\geq 0.0\).

NE_REAL_2
On entry, xmin = \(\langle value\rangle\) and xmax = \(\langle value\rangle\).  
Constraint: xmin < xmax.  
On entry, ymin = \(\langle value\rangle\) and ymax = \(\langle value\rangle\).  
Constraint: ymin < ymax.

7 Accuracy
If on exit approx = 1, see the comments in Section 3 regarding the quality of approximation; increase 
the values in maxm to attempt to avoid approximation.

8 Parallelism and Performance
nag_rand_field_2d_predef_setup (g05zrc) is threaded by NAG for parallel execution in multithreaded 
implementations of the NAG Library.  
nag_rand_field_2d_predef_setup (g05zrc) makes calls to BLAS and/or LAPACK routines, which may be 
threaded within the vendor library used by this implementation. Consult the documentation for the 
vendor library for further information.  
Please consult the X06 Chapter Introduction for information on how to control and interrogate the 
OpenMP environment used within this function. Please also consult the Users’ Note for your 
implementation for any additional implementation-specific information.

9 Further Comments
None.
10 Example

This example calls nag_rand_field_2d_predef_setup (g05zrc) to calculate the eigenvalues of the embedding matrix for 25 sample points on a 5 by 5 grid of a two-dimensional random field characterized by the symmetric stable variogram ($\text{cov} = \text{Nag\_VgmSymmStab}$).

10.1 Program Text

```c
/* nag_rand_field_2d_predef_setup (g05zrc) Example Program. */
/* Copyright 2014 Numerical Algorithms Group. */
/* Mark 24, 2013. */

#include <math.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagg05.h>

static void display_results(Integer approx, Integer *m, double rho,
                        double *eig, Integer icount, double *lam);
static void read_input_data(Nag_Variogram *cov, Integer *np, double *params,
                        Nag_NormType *norm, double *var, double *xmin,
                        double *xmax, double *ymin, double *ymax,
                        Integer *ns, Integer *maxm, Nag_EmbedScale *corr,
                        Nag_EmbedPad *pad);

int main(void)
{
    /* Scalars */
    Integer exit_status = 0;
    double rho, var, xmax, xmin, ymax, ymin;
    Integer approx, icount, np;
    /* Arrays */
    double eig[3], params[5];
    double *lam = 0, *xx = 0, *yy = 0;
    Integer m[2], maxm[2], ns[2];
    /* Nag types */
    Nag_Variogram cov;
    Nag_NormType norm;
    Nag_EmbedPad pad;
    Nag_EmbedScale corr;
    NagError fail;

    INIT_FAIL(fail);

    printf("nag_rand_field_2d_predef_setup (g05zrc) Example Program Results\n\n");
    /* Get problem specifications from data file*/
    read_input_data(&cov, &np, params, &norm, &var, &xmin, &xmax, &ymin, &ymax,
                    &ns, maxm, &corr, &pad);
    if (!(lam = NAG_ALLOC(maxm[0]*maxm[1], double))||
        !(xx = NAG_ALLOC(ns[0], double))||
        !(yy = NAG_ALLOC(ns[1], double)))
    {
        printf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }

    /* Get square roots of the eigenvalues of the embedding matrix. These are
     * obtained from the setup for simulating two-dimensional random fields,
     * with a predefined variogram, by the circulant embedding method using
     * nag_rand_field_2d_predef_setup (g05zrc).
     */
    nag_rand_field_2d_predef_setup(ns, xmin, xmax, ymin, ymax, maxm, var, cov,
                                   norm, np, params, pad, corr, lam, xx, yy, m,
                                   approx, &rho, &icount, eig, &fail);

    if (fail.code != NE_NOERROR) {
        printf("Error from nag_rand_field_2d_predef_setup (g05zrc).\n%\n", fail.message);
    }
}
```
exit_status = 1;
goto END;
}
/* Output results*/
display_results(approx, m, rho, eig, icount, lam);
END:
NAG_FREE(lam);
NAG_FREE(xx);
NAG_FREE(yy);
return exit_status;
}

void read_input_data(Nag_Variogram *cov, Integer *np, double *params,
Nag_NormType *norm, double *var, double *xmin,
double *xmax, double *ymin, double *ymax,
Integer *ns, Integer *maxm, Nag EmbedScale *corr,
Nag EmbedPad *pad)
{
    Integer j;
    char nag_enum_arg[40];

    /* Read in covariance function name and convert to value using
     * nag_enum_name_to_value (x04nac).
    */
    #ifdef _WIN32
        scanf_s("%*[\n] %39s%*[\n]", nag_enum_arg, _countof(nag_enum_arg));
    #else
        scanf("%*[\n] %39s%*[\n]", nag_enum_arg);
    #endif
    *cov = (Nag_Variogram) nag_enum_name_to_value(nag_enum_arg);
    /* Read in parameters */
    #ifdef _WIN32
        scanf_s("%"NAG_IFMT"%*[\n]",np);
    #else
        scanf("%"NAG_IFMT"%*[\n]",np);
    #endif
    for (j=0; j<*np; j++)
        #ifdef _WIN32
            scanf_s("%lf", &params[j]);
        #else
            scanf("%lf", &params[j]);
        #endif
    #ifdef _WIN32
        scanf_s("%*[\n]"),
    #else
        scanf("%*[\n]"
    #endif
    /* Read choice of norm to use, and convert name to value. */
    #ifdef _WIN32
        scanf_s("%39s%*[\n]", nag_enum_arg, _countof(nag_enum_arg));
    #else
        scanf("%39s%*[\n]", nag_enum_arg);
    #endif
    *norm = (Nag NormType) nag_enum_name_to_value(nag_enum_arg);
    /* Read in variance of random field*/
    #ifdef _WIN32
        scanf_s("%1f%*[\n]", var);
    #else
        scanf("%1f%*[\n]", var);
    #endif
    /* Read in domain endpoints*/
    #ifdef _WIN32
        scanf_s("%1f %1f%*[\n]", xmin, xmax);
    #else
        scanf("%1f %1f%*[\n]", xmin, xmax);
    #endif
    #ifdef _WIN32
        scanf_s("%1f %1f%*[\n]", ymin, ymax);
    #else
        scanf("%1f %1f%*[\n]", ymin, ymax);
    #endif
/* Read in number of sample points in each direction*/
#ifdef _WIN32
    scanf_s("%"NAG_IFMT" %"NAG_IFMT"%*[\n]", &ns[0], &ns[1]);
#else
    scanf("%"NAG_IFMT" %"NAG_IFMT"%*[\n]", &ns[0], &ns[1]);
#endif
/* Read in maximum size of embedding matrix*/
#ifdef _WIN32
    scanf_s("%"NAG_IFMT" %"NAG_IFMT"%*[\n]", &maxm[0], &maxm[1]);
#else
    scanf("%"NAG_IFMT" %"NAG_IFMT"%*[\n]", &maxm[0], &maxm[1]);
#endif
/* Read name of scaling in case of approximation and convert to value. */
#ifdef _WIN32
    scanf_s(" %39s%*[\n]", nag_enum_arg, _countof(nag_enum_arg));
#else
    scanf(" %39s%*[\n]", nag_enum_arg);
#endif
corr = (Nag_EmbedScale) nag_enum_name_to_value(nag_enum_arg);
/* Read in choice of padding and convert name to value. */
#ifdef _WIN32
    scanf_s(" %39s%*[\n]", nag_enum_arg, _countof(nag_enum_arg));
#else
    scanf(" %39s%*[\n]", nag_enum_arg);
#endif
*pad = (Nag_EmbedPad) nag_enum_name_to_value(nag_enum_arg);
}

void display_results(Integer approx, Integer *m, double rho, double *eig,
                     Integer icount, double *lam)
{
    /* Scalars */
    Integer i, j;

    /* Display size of embedding matrix*/
    printf("\nSize of embedding matrix = %"NAG_IFMT"\n\n", m[0]*m[1]);
    /* Display approximation information if approximation used. */
    if (approx==1) {
        printf("Approximation required\n\n");
        printf("rho = %10.5f\n", rho);
        printf("eig = ");
        for (j=0; j<3; j++)
            printf("%10.5f", eig[j]);
        printf("icount = %"NAG_IFMT"\n", icount);
    } else {
        printf("Approximation not required\n\n");
    }
    /* Display square roots of the eigenvalues of the embedding matrix. */
    printf("\nSquare roots of eigenvalues of embedding matrix:\n\n");
    for (i=0; i<m[0]; i++) {
        for (j=0; j<m[1]; j++) {
            printf("%8.4f", lam[i+j*m[0]]);
        }
        printf("\n");
    }
}

10.2 Program Data
g05 – Random Number Generators
g05zrc

nag_rand_field_2d_predef_setup (g05zrc) Example Program Data

Nag_VgmSymmStab : cov
  3 : np (3 parameters for 2D Nag_VgmSymmStab)
  0.1 0.15 1.2 : params (c1, c2 and nu)
Nag_TwoNorm : norm
  0.5 : var
-1.0 1.0 : xmin, xmax

Mark 25 g05zrc.11
10.3 Program Results

nag_rand_field_2d_predef_setup (g05zrc) Example Program Results

Size of embedding matrix = 64
Approximation not required

Square roots of eigenvalues of embedding matrix:

```
0.8966 0.8234 0.6810 0.5757 0.5391 0.5757 0.6810 0.8234
0.8940 0.8217 0.6804 0.5756 0.5391 0.5756 0.6804 0.8217
0.8877 0.8175 0.6792 0.5754 0.5390 0.5754 0.6792 0.8175
0.8813 0.8133 0.6780 0.5751 0.5389 0.5751 0.6780 0.8133
0.8787 0.8116 0.6774 0.5750 0.5390 0.5750 0.6774 0.8116
0.8813 0.8133 0.6780 0.5751 0.5391 0.5751 0.6780 0.8133
0.8877 0.8175 0.6792 0.5754 0.5391 0.5754 0.6792 0.8175
0.8940 0.8217 0.6804 0.5756 0.5391 0.5756 0.6804 0.8217
```

The two plots shown below illustrate the random fields that can be generated by nag_rand_field_2d_generate (g05zsc) using the eigenvalues calculated by nag_rand_field_2d_predef_setup (g05zrc). These are for two realizations of a two-dimensional random field, based on eigenvalues of the embedding matrix for points on a 100 by 100 grid. The random field is characterized by the exponential variogram ($\text{cov} = \text{Nag}\_\text{VgmExponential}$) with correlation lengths both equal to 0.1.
Example Program 2
Second realization of two-dimensional Random Field exponential variogram, correlation lengths = 0.1