# 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

```
#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[],
     Nag_EmbedPad pad, Nag_EmbedScale corr, double lam[], double xx[],
     double yy[], Integer m[], Integer *approx, double *rho, Integer *icount,
     double eig[], NagError *fail)
```

## 3 Description

A two-dimensional random field $Z(\mathbf{x})$ in $\mathbb{R}^2$ is a function which is random at every point $\mathbf{x} \in \mathbb{R}^2$, so $Z(\mathbf{x})$ is a random variable for each $\mathbf{x}$. The random field has a mean function $\mu(\mathbf{x}) = \mathbb{E}[Z(\mathbf{x})]$ and a symmetric positive semidefinite covariance function $C(\mathbf{x},\mathbf{y}) = \mathbb{E}[(Z(\mathbf{x}) - \mu(\mathbf{x}))(Z(\mathbf{y}) - \mu(\mathbf{y}))]$. $Z(\mathbf{x})$ is a Gaussian random field if for any choice of $n \in \mathbb{N}$ and $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^2$, the random vector $[Z(\mathbf{x}_1), \ldots, Z(\mathbf{x}_n)]^{\mathrm{T}}$ follows a multivariate Normal distribution, which would have a mean vector $\tilde{\boldsymbol{\mu}}$ with entries $\tilde{\mu}_i = \mu(\mathbf{x}_i)$ and a covariance matrix $\tilde{C}$ with entries $\tilde{C}_{ij} = C(\mathbf{x}_i, \mathbf{x}_j)$. A Gaussian random field $Z(\mathbf{x})$ is stationary if $\mu(\mathbf{x})$ is constant for all $\mathbf{x} \in \mathbb{R}^2$ and $C(\mathbf{x},\mathbf{y}) = C(\mathbf{x}+\mathbf{a}, \mathbf{y}+\mathbf{a})$ for all $\mathbf{x}, \mathbf{y}, \mathbf{a} \in \mathbb{R}^2$ and hence we can express the covariance function $C(\mathbf{x},\mathbf{y})$ as a function $\gamma$ of one variable: $C(\mathbf{x},\mathbf{y}) = \gamma(\mathbf{x}-\mathbf{y})$. $\gamma$ 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 $\gamma(\mathbf{x})$, 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 $\mathbf{X}$ of size $N_1 \times N_2$, with mean vector zero and a symmetric covariance matrix $A$, which is an $N_2$ by $N_2$ block Toeplitz matrix with Toeplitz blocks of size $N_1$ by $N_1$. 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 $M_2$ by $M_2$ block circulant matrix with circulant blocks of size $M_1$ by $M_1$, where $M_1 \geq 2(N_1 - 1)$ and $M_2 \geq 2(N_2 - 1)$. $B$ can now be factorized as $B = W \Lambda W^* = R^* R$, where $W$ is the two-dimensional Fourier matrix ($W^*$ is the complex conjugate of $W$), $\Lambda$ is the diagonal matrix containing the eigenvalues of $B$ and $R = \Lambda^{\frac{1}{2}} W^*$. $B$ 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 $\Lambda$ are non-negative (i.e., $B$ is positive semidefinite), $B$ is a covariance matrix for a random vector $\mathbf{Y}$ which has $M_2$ blocks of size $M_1$. Two samples of $\mathbf{Y}$ can now be simulated from the real and imaginary parts of $R^*(\mathbf{U} + i\mathbf{V})$, where $\mathbf{U}$ and $\mathbf{V}$ have elements from the standard Normal distribution. Since $R^*(\mathbf{U} + i\mathbf{V}) = W \Lambda^{\frac{1}{2}} (\mathbf{U} + i\mathbf{V})$, this calculation can be done using a discrete Fourier transform of the vector $\Lambda^{\frac{1}{2}}(\mathbf{U} + i\mathbf{V})$. Two samples of the random vector $\mathbf{X}$ can now be recovered by taking the first $N_1$ elements of the first $N_2$ blocks of each sample of $\mathbf{Y}$ – because the original covariance matrix $A$ is embedded in $B$, $\mathbf{X}$ 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 **maxm**, an approximation procedure is used. We write $\Lambda = \Lambda_+ + \Lambda_-$, where $\Lambda_+$ and $\Lambda_-$ contain the non-negative and negative eigenvalues of $B$ respectively. Then $B$ is replaced by $\rho B_+$ where $B_+ = W \Lambda_+ W^*$ and $\rho \in (0, 1]$ is a scaling factor. The error $\epsilon$ in approximating the distribution of the random field is given by

$$\epsilon = \sqrt{\frac{(1-\rho)^2 \operatorname{trace} \Lambda + \rho^2 \operatorname{trace} \Lambda_-}{M}}.$$

Three choices for $\rho$ are available, and are determined by the input argument **corr**:

setting **corr** = Nag_EmbedScaleTraces sets

$$\rho = \frac{\operatorname{trace} \Lambda}{\operatorname{trace} \Lambda_+},$$

setting **corr** = Nag_EmbedScaleSqrtTraces sets

$$\rho = \sqrt{\frac{\operatorname{trace} \Lambda}{\operatorname{trace} \Lambda_+}},$$

setting **corr** = Nag_EmbedScaleOne sets $\rho = 1$.

nag_rand_field_2d_predef_setup (g05zrc) finds a suitable positive semidefinite embedding matrix $B$ and outputs its sizes in the vector **m** and the square roots of its eigenvalues in **lam**. If approximation is used, 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

Dietrich C R and Newsam G N (1997) Fast and exact simulation of stationary Gaussian processes through circulant embedding of the covariance matrix *SIAM J. Sci. Comput.* **18** 1088–1107

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

Wood A T A and Chan G (1997) Algorithm AS 312: An Algorithm for Simulating Stationary Gaussian Random Fields *Journal of the Royal Statistical Society, Series C (Applied Statistics) (Volume 46)* **1** 171–181

## 5 Arguments

1: **ns**[**2**] – const Integer *Input*

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

*Constraints*:

**ns**[0] $\geq 1$;
**ns**[1] $\geq 1$.

2: **xmin** – double *Input*

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

*Constraint*: **xmin** < **xmax**.

3: **xmax** – double *Input*

*On entry*: the upper bound for the $x$-coordinate, for the region in which the random field is to be simulated.

*Constraint*: **xmin** $<$ **xmax**.

4: **ymin** – double *Input*

*On entry*: the lower bound for the $y$-coordinate, for the region in which the random field is to be simulated.

*Constraint*: **ymin** $<$ **ymax**.

5: **ymax** – double *Input*

*On entry*: the upper bound for the $y$-coordinate, for the region in which the random field is to be simulated.

*Constraint*: **ymin** $<$ **ymax**.

6: **maxm**[**2**] – const Integer *Input*

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

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

7: **var** – double *Input*

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

*Constraint*: **var** $\geq 0.0$.

8: **cov** – Nag_Variogram *Input*

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

**cov** $=$ Nag_VgmSymmStab
Symmetric stable variogram

$$\gamma(\mathbf{x}) = \sigma^2 \exp\left(-(x')^\nu\right),$$

where

$\ell_1 = \mathbf{params}[0]$, $\ell_1 > 0$,

$\ell_2 = \mathbf{params}[1]$, $\ell_2 > 0$,

$\nu = \mathbf{params}[2]$, $0 < \nu \leq 2$.

**cov** $=$ Nag_VgmCauchy
Cauchy variogram

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

where

$\ell_1 = \mathbf{params}[0]$, $\ell_1 > 0$,

$\ell_2 = \mathbf{params}[1]$, $\ell_2 > 0$,

$\nu = \mathbf{params}[2]$, $\nu > 0$.

**cov** = Nag_VgmDifferential
Differential variogram with compact support

$$\gamma(\mathbf{x}) = \begin{cases} \sigma^2\left(1 + 8x' + 25(x')^2 + 32(x')^3\right)(1 - x')^8, & x' < 1, \\ 0, & x' \geq 1, \end{cases}$$

where

$\ell_1 =$ **params**[0], $\ell_1 > 0$,

$\ell_2 =$ **params**[1], $\ell_2 > 0$.

**cov** = Nag_VgmExponential
Exponential variogram

$$\gamma(\mathbf{x}) = \sigma^2 \exp(-x'),$$

where

$\ell_1 =$ **params**[0], $\ell_1 > 0$,

$\ell_2 =$ **params**[1], $\ell_2 > 0$.

**cov** = Nag_VgmGauss
Gaussian variogram

$$\gamma(\mathbf{x}) = \sigma^2 \exp\left(-(x')^2\right),$$

where

$\ell_1 =$ **params**[0], $\ell_1 > 0$,

$\ell_2 =$ **params**[1], $\ell_2 > 0$.

**cov** = Nag_VgmNugget
Nugget variogram

$$\gamma(\mathbf{x}) = \begin{cases} \sigma^2, & \mathbf{x} = \mathbf{0}, \\ 0, & \mathbf{x} \neq \mathbf{0}. \end{cases}$$

No parameters need be set for this value of **cov**.

**cov** = Nag_VgmSpherical
Spherical variogram

$$\gamma(\mathbf{x}) = \begin{cases} \sigma^2\left(1 - 1.5x' + 0.5(x')^3\right), & x' < 1, \\ 0, & x' \geq 1, \end{cases}$$

where

$\ell_1 =$ **params**[0], $\ell_1 > 0$,

$\ell_2 =$ **params**[1], $\ell_2 > 0$.

**cov** = Nag_VgmBessel
Bessel variogram

$$\gamma(\mathbf{x}) = \sigma^2 \frac{2^\nu \Gamma(\nu + 1) J_\nu(x')}{(x')^\nu},$$

where

$J_\nu(\cdot)$ is the Bessel function of the first kind,

$\ell_1 =$ **params**[0], $\ell_1 > 0$,

$\ell_2 =$ **params**[1], $\ell_2 > 0$,

$\nu =$ **params**[2], $\nu \geq 0$.

**cov** = Nag_VgmHole
Hole effect variogram

$$\gamma(\mathbf{x}) = \sigma^2 \frac{\sin(x')}{x'},$$

where

$\ell_1 = \textbf{params}[0],\ \ell_1 > 0,$

$\ell_2 = \textbf{params}[1],\ \ell_2 > 0.$

**cov** = Nag_VgmWhittleMatern
Whittle-Matérn variogram

$$\gamma(\mathbf{x}) = \sigma^2 \frac{2^{1-\nu}(x')^{\nu} K_\nu(x')}{\Gamma(\nu)},$$

where

$K_\nu(\cdot)$ is the modified Bessel function of the second kind,

$\ell_1 = \textbf{params}[0],\ \ell_1 > 0,$

$\ell_2 = \textbf{params}[1],\ \ell_2 > 0,$

$\nu = \textbf{params}[2],\ \nu > 0.$

**cov** = Nag_VgmContParam
Continuously parameterised variogram with compact support

$$\gamma(\mathbf{x}) = \begin{cases} \sigma^2 \frac{2^{1-\nu}(x')^{\nu} K_\nu(x')}{\Gamma(\nu)} \left(1 + 8x'' + 25(x'')^2 + 32(x'')^3\right)(1 - x'')^8, & x'' < 1, \\ 0, & x'' \geq 1, \end{cases}$$

where

$x'' = \left\| \frac{x'}{\ell_1 s_1}, \frac{y'}{\ell_2 s_2} \right\|,$

$K_\nu(\cdot)$ is the modified Bessel function of the second kind,

$\ell_1 = \textbf{params}[0],\ \ell_1 > 0,$

$\ell_2 = \textbf{params}[1],\ \ell_2 > 0,$

$s_1 = \textbf{params}[2],\ s_1 > 0,$

$s_2 = \textbf{params}[3],\ s_2 > 0,$

$\nu = \textbf{params}[4],\ \nu > 0.$

**cov** = Nag_VgmGenHyp
Generalized hyperbolic distribution variogram

$$\gamma(\mathbf{x}) = \sigma^2 \frac{\left(\delta^2 + (x')^2\right)^{\frac{\lambda}{2}}}{\delta^\lambda K_\lambda(\kappa\delta)} K_\lambda\left(\kappa\left(\delta^2 + (x')^2\right)^{\frac{1}{2}}\right),$$

where

$K_\lambda(\cdot)$ is the modified Bessel function of the second kind,

$\ell_1 = \textbf{params}[0],\ \ell_1 > 0,$

$\ell_2 = \textbf{params}[1],\ \ell_2 > 0,$

$\lambda = \textbf{params}[2]$, no constraint on $\lambda$,

$\delta = \textbf{params}[3],\ \delta > 0,$

$\kappa =$ **params**$[4]$, $\kappa > 0$.

*Constraint*: **cov** = Nag_VgmSymmStab, Nag_VgmCauchy, Nag_VgmDifferential, Nag_VgmExponential, Nag_VgmGauss, Nag_VgmNugget, Nag_VgmSpherical, Nag_VgmBessel, Nag_VgmHole, Nag_VgmWhittleMatern, Nag_VgmContParam or Nag_VgmGenHyp.

9: **norm** – Nag_NormType *Input*

*On entry*: determines which norm to use when calculating the variogram.

**norm** = Nag_OneNorm
The 1-norm is used, i.e., $\|x, y\| = |x| + |y|$.

**norm** = Nag_TwoNorm
The 2-norm (Euclidean norm) is used, i.e., $\|x, y\| = \sqrt{x^2 + y^2}$.

*Suggested value*: **norm** = Nag_TwoNorm.

*Constraint*: **norm** = Nag_OneNorm or Nag_TwoNorm.

10: **np** – Integer *Input*

*On entry*: the number of parameters to be set. Different covariance functions need a different number of parameters.

**cov** = Nag_VgmNugget
**np** must be set to 0.

**cov** = Nag_VgmDifferential, Nag_VgmExponential, Nag_VgmGauss, Nag_VgmSpherical or Nag_VgmHole
**np** must be set to 2.

**cov** = Nag_VgmSymmStab, Nag_VgmCauchy, Nag_VgmBessel or Nag_VgmWhittleMatern
**np** must be set to 3.

**cov** = Nag_VgmContParam or Nag_VgmGenHyp
**np** must be set to 5.

11: **params**[**np**] – const double *Input*

*On entry*: the parameters for the variogram as detailed in the description of **cov**.

*Constraint*: see **cov** for a description of the individual parameter constraints.

12: **pad** – Nag_EmbedPad *Input*

*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.

**pad** = Nag_EmbedPadZeros
The embedding matrix is padded with zeros.

**pad** = Nag_EmbedPadValues
The embedding matrix is padded with values of the variogram.

*Suggested value*: **pad** = Nag_EmbedPadValues.

*Constraint*: **pad** = Nag_EmbedPadZeros or Nag_EmbedPadValues.

13: **corr** – Nag_EmbedScale *Input*

*On entry*: determines which approximation to implement if required, as described in Section 3.

*Suggested value*: **corr** = Nag_EmbedScaleTraces.

*Constraint*: **corr** = Nag_EmbedScaleTraces, Nag_EmbedScaleSqrtTraces or Nag_EmbedScaleOne.

14: **lam**[**maxm**[0] × **maxm**[1]] – double *Output*

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

15: **xx**[**ns**[0]] – double *Output*

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

16: **yy**[**ns**[1]] – double *Output*

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

17: **m**[**2**] – Integer *Output*

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

18: **approx** – Integer * *Output*

*On exit*: indicates whether approximation was used.

**approx** = 0
No approximation was used.

**approx** = 1
Approximation was used.

19: **rho** – double * *Output*

*On exit*: indicates the scaling of the covariance matrix. **rho** = 1.0 unless approximation was used with **corr** = Nag_EmbedScaleTraces or Nag_EmbedScaleSqrtTraces.

20: **icount** – Integer * *Output*

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

21: **eig**[**3**] – double *Output*

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

22: **fail** – NagError * *Input/Output*

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 ⟨*value*⟩ had an illegal value.

**NE_ENUM_INT**

On entry, **np** = ⟨*value*⟩.
Constraint: for **cov** = ⟨*value*⟩, **np** = ⟨*value*⟩.

**NE_ENUM_REAL_1**

On entry, **params**[$\langle value\rangle$] = $\langle value\rangle$.
Constraint: dependent on **cov**, see documentation.

**NE_INT_ARRAY**

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

On entry, **ns** = [$\langle value\rangle, \langle value\rangle$].
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 (**cov** = Nag_VgmSymmStab).

### 10.1 Program Text

```
/* 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%s\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("%lf%*[^\n]", var);
#else
  scanf("%lf%*[^\n]", var);
#endif
  /* Read in domain endpoints*/
#ifdef _WIN32
  scanf_s("%lf %lf%*[^\n]", xmin, xmax);
#else
  scanf("%lf %lf%*[^\n]", xmin, xmax);
#endif
#ifdef _WIN32
  scanf_s("%lf %lf%*[^\n]", ymin, ymax);
#else
  scanf("%lf %lf%*[^\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("\nicount = %"NAG_IFMT"\n", icount);
  } else {
    printf("Approximation not required\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

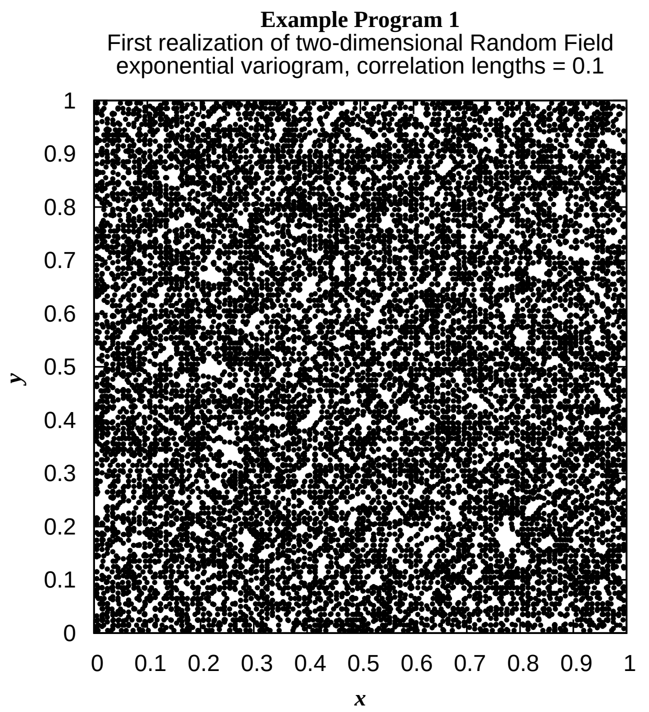
```
nag_rand_field_2d_predef_setup (g05zrc) Example Program Data
 Nag_VgmSymmStab      : cov
  3                   : np (3 parameters for 2D Nag_VgmSymmStab)
  0.1   0.15  1.2     : params (c1, c2 and nu)
  Nag_TwoNorm         : norm
  0.5                 : var
 -1.0   1.0           : xmin, xmax
```

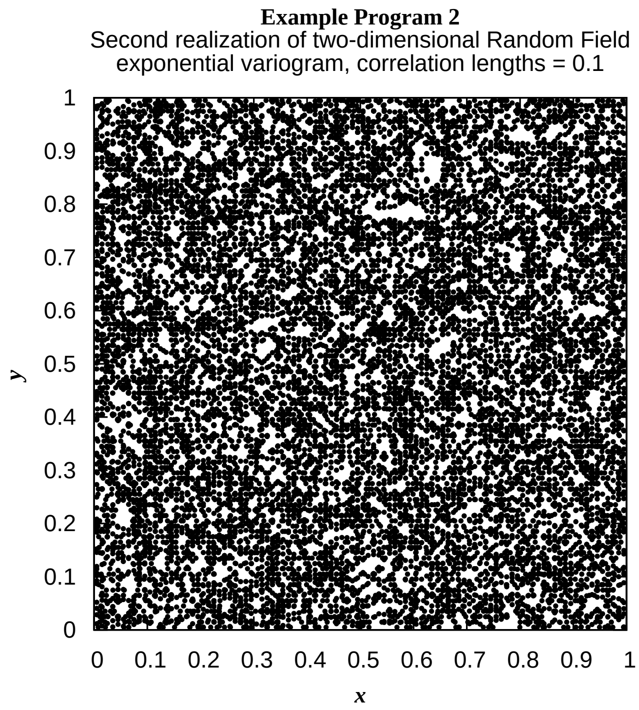
```
 -0.5   0.5          : ymin, ymax
  5     5            : ns
 64    64            : maxm
 Nag_EmbedScaleOne   : corr
 Nag_EmbedPadValues  : pad
```

### 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  0.5391  0.5754  0.6792  0.8175
  0.8813  0.8133  0.6780  0.5751  0.5390  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  0.5390  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 (**cov** = Nag_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
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
y
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1
x

---