

## NAG Library Function Document

### nag\_rand\_field\_2d\_user\_setup (g05zqc)

#### 1 Purpose

nag\_rand\_field\_2d\_user\_setup (g05zqc) performs the setup required in order to simulate stationary Gaussian random fields in two dimensions, for a user-defined 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_user_setup (const Integer ns[], double xmin,
    double xmax, double ymin, double ymax, const Integer maxm[], double var,
    void (*cov2)(double x, double y, double *gamma, Nag_Comm *comm),
    Nag_Parity parity, Nag_EmbedPad pad, Nag_EmbedScale corr, double lam[],
    double xx[], double yy[], Integer m[], Integer *approx, double *rho,
    Integer *icount, double eig[], Nag_Comm *comm, 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  $\mathbf{x}_1, \dots, \mathbf{x}_n \in \mathbb{R}^2$ , the random vector  $[Z(\mathbf{x}_1), \dots, Z(\mathbf{x}_n)]^T$  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  $\gamma(\mathbf{0}) = \sigma^2$ .

The functions nag\_rand\_field\_2d\_user\_setup (g05zqc) 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 Normal 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.

The symmetry of  $A$  as a block matrix, and the symmetry of each block of  $A$ , depends on whether the variogram  $\gamma$  is even or not.  $\gamma$  is even in its first coordinate if  $\gamma([-x_1, x_2]^T) = \gamma([x_1, x_2]^T)$ , even in its second coordinate if  $\gamma([x_1, -x_2]^T) = \gamma([x_1, x_2]^T)$ , and even if it is even in both coordinates (in two

dimensions it is impossible for  $\gamma$  to be even in one coordinate and uneven in the other). If  $\gamma$  is even then  $A$  is a symmetric block matrix and has symmetric blocks; if  $\gamma$  is uneven then  $A$  is not a symmetric block matrix and has non-symmetric blocks. In the uneven case,  $M_1$  and  $M_2$  are set to be odd in order to guarantee symmetry in  $B$ .

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  $\mathbf{Y}$  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  $A = A_+ + A_-$ , where  $A_+$  and  $A_-$  contain the non-negative and negative eigenvalues of  $B$  respectively. Then  $B$  is replaced by  $\rho B_+$  where  $B_+ = W\Lambda_+W^*$  and  $\rho \in (0, 1]$  is a scaling factor. The error  $\epsilon$  in approximating the distribution of the random field is given by

$$\epsilon = \sqrt{\frac{(1 - \rho)^2 \text{trace } A + \rho^2 \text{trace } A_-}{M}}.$$

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

setting **corr** = Nag\_EmbedScaleTraces sets

$$\rho = \frac{\text{trace } A}{\text{trace } A_+},$$

setting **corr** = Nag\_EmbedScaleSqrtTraces sets

$$\rho = \sqrt{\frac{\text{trace } A}{\text{trace } A_+}},$$

setting **corr** = Nag\_EmbedScaleOne sets  $\rho = 1$ .

nag\_rand\_field\_2d\_user\_setup (g05zqc) 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 (1994) Simulation of stationary Gaussian processes in  $[0, 1]^d$  *Journal of Computational and Graphical Statistics* **3(4)** 409–432

## 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:

$$\begin{aligned} \mathbf{ns}[0] &\geq 1; \\ \mathbf{ns}[1] &\geq 1. \end{aligned}$$

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] × **maxm**[1].

Constraints:

$$\begin{aligned} &\text{if } \mathbf{parity} = \text{Nag\_Even}, \mathbf{maxm}[i] \geq 2^k, \text{ where } k \text{ is the smallest integer satisfying} \\ &2^k \geq 2(\mathbf{ns}[i] - 1), \text{ for } i = 0, 1; \\ &\text{if } \mathbf{parity} = \text{Nag\_Odd}, \mathbf{maxm}[i] \geq 3^k, \text{ where } k \text{ is the smallest integer satisfying} \\ &3^k \geq 2(\mathbf{ns}[i] - 1), \text{ for } i = 0, 1. \end{aligned}$$

7: **var** – double *Input*

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

*Constraint:* **var** ≥ 0.0.

8: **cov2** – function, supplied by the user *External Function*

**cov2** must evaluate the variogram  $\gamma(\mathbf{x})$  for all  $\mathbf{x}$  if **parity** = Nag\_Odd, and for all  $\mathbf{x}$  with non-negative entries if **parity** = Nag\_Even. The value returned in **gamma** is multiplied internally by **var**.

The specification of **cov2** is:

```
void cov2 (double x, double y, double *gamma, Nag_Comm *comm)
```

1: **x** – double *Input*

*On entry:* the coordinate  $x$  at which the variogram  $\gamma(\mathbf{x})$  is to be evaluated.

- |    |  |               |
|----|--|---------------|
| 2: | <b>y</b> – double<br><i>On entry:</i> the coordinate $y$ at which the variogram $\gamma(\mathbf{x})$ is to be evaluated.   | <i>Input</i>  |
| 3: | <b>gamma</b> – double *<br><i>On exit:</i> the value of the variogram $\gamma(\mathbf{x})$ .   | <i>Output</i> |
| 4: | <b>comm</b> – Nag_Comm *<br>Pointer to structure of type Nag_Comm; the following members are relevant to <b>cov2</b> .<br><br><b>user</b> – double *<br><b>iuser</b> – Integer *<br><b>p</b> – Pointer<br><br>The type Pointer will be void *. Before calling nag_rand_field_2d_user_setup (g05zqc) you may allocate memory and initialize these pointers with various quantities for use by <b>cov2</b> when called from nag_rand_field_2d_user_setup (g05zqc) (see Section 2.3.1.1 in How to Use the NAG Library and its Documentation). |               |
- 9: **parity** – Nag\_Parity *Input*  
*On entry:* indicates whether the covariance function supplied is even or uneven.  
**parity** = Nag\_Odd  
 The covariance function is uneven.  
**parity** = Nag\_Even  
 The covariance function is even.  
*Constraint:* **parity** = Nag\_Odd or Nag\_Even.
- 10: **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.
- 11: **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.
- 12: **lam**[**maxm**[0] × **maxm**[1]] – double *Output*  
*On exit:* contains the square roots of the eigenvalues of the embedding matrix.
- 13: **xx**[**ns**[0]] – double *Output*  
*On exit:* the points of the  $x$ -coordinates at which values of the random field will be output.

- 14: **yy[ns[1]]** – double *Output*  
*On exit:* the points of the  $y$ -coordinates at which values of the random field will be output.
- 15: **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  $M_1 \times M_2$ .
- 16: **approx** – Integer \* *Output*  
*On exit:* indicates whether approximation was used.  
**approx** = 0  
 No approximation was used.  
**approx** = 1  
 Approximation was used.
- 17: **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.
- 18: **icount** – Integer \* *Output*  
*On exit:* indicates the number of negative eigenvalues in the embedding matrix which have had to be set to zero.
- 19: **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.
- 20: **comm** – Nag\_Comm \*  
 The NAG communication argument (see Section 2.3.1.1 in How to Use the NAG Library and its Documentation).
- 21: **fail** – NagError \* *Input/Output*  
 The NAG error argument (see Section 2.7 in How to Use the NAG Library and its Documentation).

## 6 Error Indicators and Warnings

### NE\_ALLOC\_FAIL

Dynamic memory allocation failed.

See Section 2.3.1.2 in How to Use the NAG Library and its Documentation for further information.

### NE\_BAD\_PARAM

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

### NE\_INT\_ARRAY

On entry, **maxm** = [ $\langle value \rangle$ ,  $\langle value \rangle$ ].

Constraint: the minima for **maxm** are [ $\langle value \rangle$ ,  $\langle value \rangle$ ].

Where, if **parity** = Nag\_Even, the minimum calculated value of **maxm**[ $i - 1$ ] is given by  $2^k$ ,

where  $k$  is the smallest integer satisfying  $2^k \geq 2(\mathbf{ns}[i-1] - 1)$ , and if **parity** = Nag\_Odd, the minimum calculated value of **maxm**[ $i-1$ ] is given by  $3^k$ , where  $k$  is the smallest integer satisfying  $3^k \geq 2(\mathbf{ns}[i-1] - 1)$ , 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 2.7.6 in How to Use the NAG Library and its Documentation for further information.

### NE\_NO\_LICENCE

Your licence key may have expired or may not have been installed correctly.

See Section 2.7.5 in How to Use the NAG Library and its Documentation 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\_user\_setup (g05zqc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag\_rand\_field\_2d\_user\_setup (g05zqc) 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\_user\_setup (g05zqc) 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:

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

where  $x' = \left| \frac{x}{\ell_1} + \frac{y}{\ell_2} \right|$ , and  $\ell_1$ ,  $\ell_2$  and  $\nu$  are parameters.

It should be noted that the symmetric stable variogram is one of the pre-defined variograms available in `nag_rand_field_2d_predef_setup` (g05zrc). It is used here purely for illustrative purposes.

## 10.1 Program Text

```

/* nag_rand_field_2d_user_setup (g05zqc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */
#include <math.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagg05.h>

#ifdef __cplusplus
extern "C"
{
#endif
    static void NAG_CALL cov2(double t1, double t2, double *gamma,
                             Nag_Comm *comm);
#ifdef __cplusplus
}
#endif

static void display_results(Integer approx, Integer *m, double rho,
                           double *eig, Integer icount, double *lam);
static void read_input_data(Nag_NormType *norm, double *l1, double *l2,
                           double *nu, 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 l1, l2, nu, rho, var, xmax, xmin, ymax, ymin;
    Integer approx, icount;
    /* Arrays */
    double eig[3];
    double *lam = 0, *xx = 0, *yy = 0;
    Integer m[2], maxm[2], ns[2];
    /* Nag types */
    Nag_NormType norm;
    Nag_EmbedPad pad;
    Nag_EmbedScale corr;
    Nag_Parity even = Nag_Even;
    Nag_Comm comm;
    NagError fail;

    INIT_FAIL(fail);

    printf("nag_rand_field_2d_user_setup (g05zqc) Example Program Results\n\n");
    /* Get problem specifications from data file */
    read_input_data(&norm, &l1, &l2, &nu, &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)) ||
        !(comm.iuser = NAG_ALLOC(1, Integer)) ||

```

```

        !(comm.user = NAG_ALLOC(3, double)))
    {
        printf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }
    /* Put covariance parameters in communication arrays */
    comm.iuser[0] = (Integer) norm;
    comm.user[0] = l1;
    comm.user[1] = l2;
    comm.user[2] = nu;
    /* Get square roots of the eigenvalues of the embedding matrix. These are
     * obtained from the setup for simulating two-dimensional random fields,
     * with a user-defined variogram, by the circulant embedding method using
     * nag_rand_field_2d_user_setup (g05zqc).
     */
    nag_rand_field_2d_user_setup(ns, xmin, xmax, ymin, ymax, maxm, var,
                                cov2, even, pad, corr, lam, xx, yy, m,
                                &approx, &rho, &icount, eig, &comm, &fail);
    if (fail.code != NE_NOERROR) {
        printf("Error from nag_rand_field_2d_user_setup (g05zqc).\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);
    NAG_FREE(comm.iuser);
    NAG_FREE(comm.user);
    return exit_status;
}

void read_input_data(Nag_NormType *norm, double *l1, double *l2, double *nu,
                    double *var, double *xmin, double *xmax, double *ymin,
                    double *ymax, Integer *ns, Integer *maxm,
                    Nag_EmbedScale *corr, Nag_EmbedPad *pad)
{
    char nag_enum_arg[40];

    /* Read in norm type by name and convert to value using
     * nag_enum_name_to_value (x04nac).
     */
#ifdef _WIN32
    scanf_s("%*[\n] %39s%*[\n]", nag_enum_arg,
            (unsigned)_countof(nag_enum_arg));
#else
    scanf("%*[\n] %39s%*[\n]", nag_enum_arg);
#endif
    *norm = (Nag_NormType) nag_enum_name_to_value(nag_enum_arg);
    /* read in l1, l2 and nu for cov function */
#ifdef _WIN32
    scanf_s("%lf %lf %lf%*[\n]", l1, l2, nu);
#else
    scanf("%lf %lf %lf%*[\n]", l1, l2, nu);
#endif
    /* 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, (unsigned)_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, (unsigned)_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");
    }
}

static void NAG_CALL cov2(double t1, double t2, double *gamma, Nag_Comm *comm)
{
    /* Scalars */
    double l1, l2, nu, rnorm, tc1, tc2;
    Integer norm;

    /* Covariance parameters stored in user array. */
    norm = comm->iuser[0];

```

```

l1 = comm->user[0];
l2 = comm->user[1];
nu = comm->user[2];

tc1 = fabs(t1) / l1;
tc2 = fabs(t2) / l2;
if (norm == (Integer) Nag_OneNorm) {
    rnorm = tc1 + tc2;
}
else if (norm == (Integer) Nag_TwoNorm) {
    rnorm = sqrt(tc1 * tc1 + tc2 * tc2);
}
else
    rnorm = 0.0;
*gamma = exp(-(pow(rnorm, nu)));
}

```

## 10.2 Program Data

nag\_rand\_field\_2d\_user\_setup (g05zqc) Example Program Data

```

Nag_TwoNorm      : norm
0.1  0.15 1.2    : c1, c2, nu
0.5          : var
-1    1          : xmin, xmax
-0.5  0.5       : ymin, ymax
5     5          : ns
81    81         : maxm
Nag_EmbedScaleOne : corr
Nag_EmbedPadValues : pad

```

## 10.3 Program Results

nag\_rand\_field\_2d\_user\_setup (g05zqc) 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

---