NAG Library Function Document

nag_rand_field_1d_user_setup (g05zmc)

1 Purpose

nag_rand_field_1d_user_setup (g05zmc) performs the setup required in order to simulate stationary Gaussian random fields in one dimension, 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_1d_generate (g05zpc), which simulates the random field.

2 Specification

```
#include <nag.h>
#include <nagg05.h>
void nag_rand_field_1d_user_setup (Integer ns, double xmin, double xmax,
     Integer maxm, double var,
     void  (*cov1)(double x, double *gamma, Nag_Comm *comm),
     Nag_EmbedPad pad, Nag_EmbedScale corr, double lam[], double xx[],
     Integer *m, Integer *approx, double *rho, Integer *icount, double eig[],
     Nag_Comm *comm, NagError *fail)
```

3 Description

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

The functions nag_rand_field_1d_user_setup (g05zmc) and nag_rand_field_1d_generate (g05zpc) are used to simulate a one-dimensional stationary Gaussian random field, with mean function zero and variogram $\gamma(x)$, over an interval $[x_{\min}, x_{\max}]$, using an equally spaced set of N points on the interval. The problem reduces to sampling a Normal random vector $\mathbf{X}$ of size N, with mean vector zero and a symmetric Toeplitz covariance matrix A. 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 circulant matrix B of size $M \geq 2(N-1)$, which can now be factorized as $B = W\Lambda W^* = R^* R$, where W is the Fourier matrix (W^* is the complex conjugate of W), Λ 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, 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 Λ are non-negative (i.e., B is positive semidefinite), B is a covariance matrix for a random vector $\mathbf{Y}$, two samples of which 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 elements 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 Λ_+ and Λ_- contain the non-negative and negative eigenvalues of B respectively. Then B is replaced by ρB_+ where $B_+ = W\Lambda_+ W^*$ and $\rho \in (0,1]$ is a scaling factor. The error ϵ 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 ρ 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_1d_user_setup (g05zmc) finds a suitable positive semidefinite embedding matrix B and outputs its size, **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** – Integer *Input*

On entry: the number of sample points to be generated in realizations of the random field.

Constraint: **ns** ≥ 1.

2: **xmin** – double *Input*

On entry: the lower bound for the interval over which the random field is to be simulated.

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

3: **xmax** – double *Input*

On entry: the upper bound for the interval over which the random field is to be simulated.

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

4: **maxm** – Integer *Input*

On entry: the maximum size of the circulant matrix to use. For example, if the embedding matrix is to be allowed to double in size three times before the approximation procedure is used, then choose **maxm** $= 2^{k+2}$ where $k = 1 + \lceil \log_2(\mathbf{ns} - 1) \rceil$.

Suggested value: 2^{k+2} where $k = 1 + \lceil \log_2 (\mathbf{ns} - 1) \rceil$

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

5: **var** – double *Input*

On entry: the multiplicative factor σ^2 of the variogram $\gamma(x)$.

Constraint: **var** ≥ 0.0.

6: **cov1** – function, supplied by the user *External Function*

cov1 must evaluate the variogram $\gamma(x)$, without the multiplicative factor σ^2, for all $x \geq 0$. The value returned in **gamma** is multiplied internally by **var**.

The specification of **cov1** is:

```
void  cov1 (double x, double *gamma, Nag_Comm *comm)
```

1: **x** – double *Input*

On entry: the value x at which the variogram $\gamma(x)$ is to be evaluated.

2: **gamma** – double * *Output*

On exit: the value of the variogram $\frac{\gamma(x)}{\sigma^2}$.

3: **comm** – Nag_Comm * *Communication Structure*

Pointer to structure of type Nag_Comm; the following members are relevant to **cov1**.

user – double *
iuser – Integer *
p – Pointer

The type Pointer will be `void *`. Before calling nag_rand_field_1d_user_setup (g05zmc) you may allocate memory and initialize these pointers with various quantities for use by **cov1** when called from nag_rand_field_1d_user_setup (g05zmc) (see Section 3.2.1.1 in the Essential Introduction).

7: **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.

8: **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.

9: **lam**[**maxm**] – double *Output*

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

10: **xx**[**ns**] – double *Output*

On exit: the points at which values of the random field will be output.

11: **m** – Integer * *Output*

On exit: the size of the embedding matrix.

12: **approx** – Integer * *Output*

On exit: indicates whether approximation was used.

approx = 0
No approximation was used.

approx = 1
Approximation was used.

13: **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.

14: **icount** – Integer * *Output*

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

15: **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.

16: **comm** – Nag_Comm * *Communication Structure*

The NAG communication argument (see Section 3.2.1.1 in the Essential Introduction).

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

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

6 Error Indicators and Warnings

NE_BAD_PARAM

On entry, argument ⟨*value*⟩ had an illegal value.

NE_INT

On entry, **maxm** = ⟨*value*⟩.
Constraint: the minimum calculated value for **maxm** is ⟨*value*⟩.
Where the minimum calculated value is given by 2^k, where k is the smallest integer satisfying $2^k \geq 2(\mathbf{ns} - 1)$.

On entry, **ns** = ⟨*value*⟩.
Constraint: $\mathbf{ns} \geq 1$.

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.

NE_REAL

On entry, **var** = ⟨*value*⟩.
Constraint: **var** ≥ 0.0.

NE_REAL_2

On entry, **xmin** = ⟨*value*⟩ and **xmax** = ⟨*value*⟩.
Constraint: **xmin** < **xmax**.

7 Accuracy

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

8 Parallelism and Performance

Not applicable.

9 Further Comments

None.

10 Example

This example calls nag_rand_field_1d_user_setup (g05zmc) to calculate the eigenvalues of the embedding matrix for 8 sample points of a random field characterized by the symmetric stable variogram:

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

where $x' = \frac{x}{\ell}$, and ℓ and ν are parameters.

It should be noted that the symmetric stable variogram is one of the pre-defined variograms available in nag_rand_field_1d_predef_setup (g05znc). It is used here purely for illustrative purposes.

10.1 Program Text

```
/* nag_rand_field_1d_user_setup (g05zmc) Example Program.
 *
 * Copyright 2013 Numerical Algorithms Group.
 *
 * Mark 24, 2013.
 */
#include <math.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagg05.h>
#include <nagx04.h>

#ifdef __cplusplus
extern "C" {
#endif
static void NAG_CALL cov1(double x, double *gamma, Nag_Comm *comm);
#ifdef __cplusplus
}
#endif
static void read_input_data(double *l, double *nu, double *var, double *xmin,
                            double *xmax, Integer *ns, Integer *maxm,
```

```
                            Nag_EmbedScale *corr, Nag_EmbedPad *pad);
static void display_results(Integer approx, Integer m, double rho,
                            double *eig, Integer icount, double *lam);
int main(void)
{
  Integer        exit_status = 0;
  /* Scalars */
  double         c, nu, rho, var, xmax, xmin;
  Integer        approx, icount, m, maxm, ns;
  /* Arrays */
  double         eig[3], *lam = 0, *xx = 0;
  /* Nag types */
  Nag_EmbedScale corr;
  Nag_EmbedPad   pad;
  Nag_Comm       comm;
  NagError       fail;

  INIT_FAIL(fail);

  printf("nag_rand_field_1d_user_setup (g05zmc) Example Program Results\n\n");

  /* Get problem specifications from data file*/
  read_input_data(&c, &nu, &var, &xmin, &xmax, &ns, &maxm, &corr, &pad);
  if (!(lam = NAG_ALLOC(maxm, double))||
      !(xx = NAG_ALLOC(ns, double)) ||
      !(comm.user = NAG_ALLOC(2, double)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }
  /* Put covariance parameters in communication array*/
  comm.user[0] = c;
  comm.user[1] = nu;
  /* Get square roots of the eigenvalues of the embedding matrix. These are
   * obtained from the setup for simulating one-dimensional random fields,
   * with a user-defined variogram, by the circulant embedding method using
   * nag_rand_field_1d_user_setup (g05zmc).
   */
  nag_rand_field_1d_user_setup(ns, xmin, xmax, maxm, var, cov1, pad,
                               corr, lam, xx, &m, &approx, &rho, &icount,
                               eig, &comm, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_rand_field_1d_user_setup (g05zmc).\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(comm.user);
  return exit_status;
}

void read_input_data(double *l, double *nu, double *var, double *xmin,
                     double *xmax, Integer *ns, Integer *maxm,
                     Nag_EmbedScale *corr, Nag_EmbedPad *pad)
{
  /* Arrays */
  char           nag_enum_arg[40];
   /* Skip heading and get l and nu for cov1 function. */
  scanf("%*[^\n] %lf %lf%*[^\n]",l,nu);
  /* Read in variance of random field. */
  scanf("%lf%*[^\n]",var);
  /* Read in domain endpoints. */
  scanf("%lf %lf%*[^\n]",xmin,xmax);
  /* Read in number of sample points. */
  scanf("%ld%*[^\n]",ns);
```

```
  /* Read in maximum size of embedding matrix. */
  scanf("%ld%*[^\n]",maxm);
  /* Read in choice of scaling in case of approximation. */
  scanf(" %39s%*[^\n]", nag_enum_arg);
  /* nag_enum_name_to_value (x04nac).
   * Converts NAG enum member name to value
   */
  *corr = (Nag_EmbedScale) nag_enum_name_to_value(nag_enum_arg);
  /* Read in choice of padding and convert to enum name to value. */
  scanf(" %39s%*[^\n]", nag_enum_arg);
  *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)
{
  Integer j;
  /* Display size of embedding matrix*/
  printf("\nSize of embedding matrix = %ld\n\n", m);
  /* 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 = %ld\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 (j=0; j<m; j++)
    printf("%10.5f%s", lam[j], j%4==3?"\n":"");
  printf("\n");
}

static void NAG_CALL cov1(double x, double *gamma, Nag_Comm *comm)
{
  /* Scalars */
  double    dummy, l, nu;

  /* Correlation length and exponent in comm->ruser.*/
  l = comm->user[0];
  nu = comm->user[1];
  if (x==0.0) {
    *gamma = 1.0;
  } else {
    dummy =  pow(x/l,  nu);
    *gamma = exp(-dummy);
  }
}
```

10.2 Program Data

```
nag_rand_field_1d_user_setup (g05zmc) Example Program Data
 0.1    1.2               : c, nu
 0.5                      : var
-1.0    1.0               : xmin, xmax
 8                        : ns
 64                       : maxm
 Nag_EmbedScaleOne        : icorr
 Nag_EmbedPadValues       : pad
```

10.3 Program Results

```
nag_rand_field_1d_user_setup (g05zmc) Example Program Results


Size of embedding matrix = 16

Approximation not required

Square roots of eigenvalues of embedding matrix:

   0.74207   0.73932   0.73150   0.71991
   0.70639   0.69304   0.68184   0.67442
   0.67182   0.67442   0.68184   0.69304
   0.70639   0.71991   0.73150   0.73932
```