NAG Library Function Document

nag_rand_field_1d_generate (g05zpc)

1 Purpose

nag_rand_field_1d_generate (g05zpc) produces realizations of a stationary Gaussian random field in one dimension, using the circulant embedding method. The square roots of the eigenvalues of the extended covariance matrix (or embedding matrix) need to be input, and can be calculated using nag_rand_field_1d_user_setup (g05zmc) or nag_rand_field_1d_predef_setup (g05znc).

2 Specification

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

```
void nag_rand_field_1d_generate (Integer ns, Integer s, Integer m,
     const double lam[], double rho, Integer state[], double z[],
     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 non-negative definite 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 $x_1, \ldots, x_n \in \mathbb{R}$, the random vector $[Z(x_1), \ldots, Z(x_n)]^{\mathrm{T}}$ 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 $\tilde{C}_{ij} = C(x_i, x_j)$. 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) or nag_rand_field_1d_predef_setup (g05znc), along with 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. 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 non-negative definite), 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 non-negative definite, 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. See the documentation of nag_rand_field_1d_user_setup (g05zmc) or nag_rand_field_1d_predef_setup (g05znc) for details of the approximation procedure.

nag_rand_field_1d_generate (g05zpc) takes the square roots of the eigenvalues of the embedding matrix B, and its size M, as input and outputs S realizations of the random field in Z.

One of the initialization functions nag_rand_init_repeatable (g05kfc) (for a repeatable sequence if computed sequentially) or nag_rand_init_nonrepeatable (g05kgc) (for a non-repeatable sequence) must be called prior to the first call to nag_rand_field_1d_generate (g05zpc).

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. This must be the same value as supplied to nag_rand_field_1d_user_setup (g05zmc) or nag_rand_field_1d_predef_setup (g05znc) when calculating the eigenvalues of the embedding matrix.

Constraint: $\mathbf{ns} \geq 1$.

2: **s** – Integer *Input*

On entry: S, the number of realizations of the random field to simulate.

Constraint: $\mathbf{s} \geq 1$.

3: **m** – Integer *Input*

On entry: M, the size of the embedding matrix, as returned by nag_rand_field_1d_user_setup (g05zmc) or nag_rand_field_1d_predef_setup (g05znc).

Constraint: $\mathbf{m} \geq \max(1, 2(\mathbf{ns} - 1))$.

4: **lam**[**m**] – const double *Input*

On entry: must contain the square roots of the eigenvalues of the embedding matrix, as returned by nag_rand_field_1d_user_setup (g05zmc) or nag_rand_field_1d_predef_setup (g05znc).

Constraint: $\mathbf{lam}[i-1] \geq 0, i = 1, 2, \dots, \mathbf{m}$.

5: **rho** – double *Input*

On entry: indicates the scaling of the covariance matrix, as returned by nag_rand_field_1d_user_setup (g05zmc) or nag_rand_field_1d_predef_setup (g05znc).

Constraint: $0.0 < \mathbf{rho} \leq 1.0$.

6: **state**[*dim*] – Integer *Communication Array*

Note: the dimension, *dim*, of this array is dictated by the requirements of associated functions that must have been previously called. This array MUST be the same array passed as argument **state** in the previous call to nag_rand_init_repeatable (g05kfc) or nag_rand_init_nonrepeatable (g05kgc).

On entry: contains information on the selected base generator and its current state.

On exit: contains updated information on the state of the generator.

7: **z**[**ns** $\times$ **s**] – double *Output*

On exit: contains the realizations of the random field. The jth realization, for the **ns** sample points, is stored in **z**[$(j-1) \times$ **ns** $+ i - 1$], for $i = 1, 2, \ldots,$ **ns**. The sample points are as returned in **xx** by nag_rand_field_1d_user_setup (g05zmc) or nag_rand_field_1d_predef_setup (g05znc).

8: **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 $\langle value \rangle$ had an illegal value.

NE_INT

On entry, **ns** $= \langle value \rangle$.
Constraint: **ns** ≥ 1.

On entry, **s** $= \langle value \rangle$.
Constraint: **s** ≥ 1.

NE_INT_2

On entry, **m** $= \langle value \rangle$ and **ns** $= \langle value \rangle$.
Constraint: **m** $\geq \max(1, 2 \times ($**ns** $- 1))$.

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_INVALID_STATE

On entry, **state** vector has been corrupted or not initialized.

NE_NEG_ELEMENT

On entry, at least one element of **lam** was negative.
Constraint: all elements of **lam** must be non-negative.

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, **rho** $= \langle value \rangle$.
Constraint: $0.0 \leq$ **rho** ≤ 1.0.

7 Accuracy

Not applicable.

8 Parallelism and Performance

nag_rand_field_1d_generate (g05zpc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

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

Because samples are generated in pairs, calling this function k times, with $\mathbf{s} = s$, say, will generate a different sequence of numbers than calling the function once with $\mathbf{s} = ks$, unless s is even.

10 Example

This example calls nag_rand_field_1d_generate (g05zpc) to generate 5 realizations of a random field on 8 sample points using eigenvalues calculated by nag_rand_field_1d_predef_setup (g05znc) for a symmetric stable variogram.

10.1 Program Text

```
/* nag_rand_field_1d_generate (g05zpc) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 24, 2013.
 */

#include <math.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagg05.h>
#include <nagx04.h>

#define NPMAX 4
#define LENST 17
#define LSEED 1
static void read_input_data(Nag_Variogram *cov, Integer *np, double *params,
                            double *var, double *xmin, double *xmax,
                            Integer *ns, Integer *maxm, Nag_EmbedScale *corr,
                            Nag_EmbedPad *pad, Integer *s);
static void display_embedding_results(Integer approx, Integer m, double rho,
                                      double *eig, Integer icount, double *lam);
static void initialize_state(Integer *state);
static void display_realizations(Integer ns, Integer s, double *xx, double *z,
                                 Integer *exit_status);

int main(void)
{
  /*  Scalars */
  Integer        exit_status = 0;
  double         rho, var, xmax, xmin;
  Integer        approx, icount, m, maxm, np, ns, s;
  /*  Arrays */
  double         eig[3], params[NPMAX];
  double         *lam = 0, *xx = 0, *z = 0;
  Integer        state[LENST];
  /* Nag types */
  Nag_Variogram  cov;
  Nag_EmbedPad   pad;
```

```
  Nag_EmbedScale corr;
  NagError       fail;

  INIT_FAIL(fail);

  printf("nag_rand_field_1d_generate (g05zpc) Example Program Results\n\n");
  fflush(stdout);
  /* Get problem specifications from data file*/
  read_input_data(&cov, &np, params, &var, &xmin, &xmax, &ns, &maxm, &corr,
                  &pad, &s);
  if (!(lam = NAG_ALLOC(maxm, double)) ||
      !(xx = NAG_ALLOC(ns, double)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }
  /* Get square roots of the eigenvalues of the embedding matrix.
   * nag_rand_field_1d_predef_setup (g05znc).
   * Setup for simulating one-dimensional random fields, preset variogram,
   * circulant embedding method
   */
  nag_rand_field_1d_predef_setup(ns, xmin, xmax, maxm, var, cov, np,
                                 params, pad, corr, lam, xx, &m, &approx,
                                 &rho, &icount, eig, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_rand_field_1d_predef_setup (g05znc).\n%s\n",
             fail.message);
      exit_status = 1;
      goto END;
    }

  fflush(stdout);
  display_embedding_results(approx, m, rho, eig, icount, lam);
  /* Initialize state array*/
  initialize_state(state);
  if (!(z = NAG_ALLOC(ns*s, double)))
    {
      printf("Allocation failure\n");
      exit_status = -2;
      goto END;
    }
  /* Compute s random field realisations.
   * nag_rand_field_1d_generate (g05zpc).
   * Generates s realisations of a one-dimensional random field by the
   * circulant embedding method.
   */
  nag_rand_field_1d_generate(ns, s, m, lam, rho, state, z, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_rand_field_1d_generate (g05zpc).\n%s\n",
             fail.message);
      exit_status = 2;
      goto END;
    }
  display_realizations(ns, s, xx, z, &exit_status);
 END:
  NAG_FREE(lam);
  NAG_FREE(xx);
  NAG_FREE(z);
  return exit_status;
}

void read_input_data(Nag_Variogram *cov, Integer *np, double *params,
                     double *var, double *xmin, double *xmax,
                     Integer *ns, Integer *maxm, Nag_EmbedScale *corr,
                     Nag_EmbedPad *pad, Integer *s)
{
  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 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
  /* Read in number of sample points. */
#ifdef _WIN32
  scanf_s("%"NAG_IFMT"%*[^\n]", ns);
#else
  scanf("%"NAG_IFMT"%*[^\n]", ns);
#endif
  /* Read in maximum size of embedding matrix. */
#ifdef _WIN32
  scanf_s("%"NAG_IFMT"%*[^\n]", maxm);
#else
  scanf("%"NAG_IFMT"%*[^\n]", maxm);
#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);
  /* Read in number of realization samples to be generated*/
#ifdef _WIN32
  scanf_s("%"NAG_IFMT"%*[^\n]", s);
#else
  scanf("%"NAG_IFMT"%*[^\n]", s);
#endif
```

```
}

void display_embedding_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 = %"NAG_IFMT"\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 = %"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 (j = 0; j < m; j++)
    printf("%10.5f%s", lam[j], j%4 == 3 ? "\n" : "");
  printf("\n");
  fflush(stdout);
}
void initialize_state(Integer *state)
{
  /*  Scalars */
  Integer  inseed = 14965, lseed = LSEED, subid = 1;
  Integer  lstate;
  /*  Arrays */
  Integer  seed[LSEED];
  /* Nag types */
  NagError fail;

  INIT_FAIL(fail);
  lstate = LENST;
  seed[0] = inseed;
  /* nag_rand_init_repeatable (g05kfc).
   * Initializes a pseudorandom number generator to give a repeatable sequence.
   */
  nag_rand_init_repeatable(Nag_Basic, subid, seed, lseed, state, &lstate,
                           &fail);
}

void display_realizations(Integer ns, Integer s, double *xx, double *z,
                          Integer *exit_status)
{
  /*  Scalars */
  Integer  indent = 0, ncols = 80;
  Integer  i;
  /*  Arrays */
  char     **rlabs = 0;
  /* Nag types */
  NagError fail;

  INIT_FAIL(fail);

  if (!(rlabs = NAG_ALLOC(ns, char *)))
    {
      printf("Allocation failure\n");
      *exit_status = -3;
      goto END;
    }
  /* Set row labels to mesh points (column label is realization number).*/
  for (i = 0; i < ns; i++)
    {
```

```
      if (!(rlabs[i] = NAG_ALLOC(11, char)))
        {
          printf("Allocation failure\n");
          *exit_status = -4;
          goto END;
        }
#ifdef _WIN32
      sprintf_s(rlabs[i], 11, "%10.5f", xx[i]);
#else
      sprintf(rlabs[i], "%10.5f", xx[i]);
#endif
    }
  printf("\n");
  fflush(stdout);
  /* Display random field results, z, using the comprehensive real general
   * matrix print routine nag_gen_real_mat_print_comp (x04cbc).
   */
  nag_gen_real_mat_print_comp(Nag_ColMajor, Nag_GeneralMatrix, Nag_NonUnitDiag,
                              ns, s, z, ns, "%10.5f",
                              "Random field realisations:",
                              Nag_CharacterLabels,
                              (const char **) rlabs, Nag_IntegerLabels, NULL,
                              ncols, indent, 0,
                              &fail);
 END:
  for (i = 0; i < ns; i++)
    {
      NAG_FREE(rlabs[i]);
    }
  NAG_FREE(rlabs);
}
```

10.2 Program Data

```
nag_rand_field_1d_generate (g05zpc) Example Program Data
 Nag_VgmSymmStab     : cov
 2                   : np (2 parameters for Nag_VgmSymmStab)
 0.1   1.2           : params (c and nu)
 0.5                 : var
-1     1             : xmin, xmax
 8                   : ns
 64                  : maxm
 Nag_EmbedScaleOne   : corr
 Nag_EmbedPadValues  : pad
 5                   : s
```

10.3 Program Results

```
nag_rand_field_1d_generate (g05zpc) 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


 Random field realisations:
                    1          2          3          4          5
 -0.87500   -0.41663   -0.81847   -0.97692    0.67410   -0.67616
 -0.62500    0.01457    1.45384    0.02481    0.52178    1.94664
 -0.37500   -0.55557    0.29127   -0.08534    0.42145   -0.13891
```

```
 -0.12500   -0.55678    0.31985   -0.60936    0.20194    0.90846
  0.12500   -0.04230    0.04860    1.45897    0.36077   -0.52877
  0.37500   -0.28057   -0.79688    0.23301    0.13351    0.40119
  0.62500    0.92981   -0.39561   -0.84545   -0.27487    0.52703
  0.87500    0.32217    1.52273   -2.16445    0.17941    1.19373
```