NAG Library Function Document

nag_rand_field_fracbm_generate (g05ztc)

1 Purpose

nag_rand_field_fracbm_generate (g05ztc) produces realizations of a fractional Brownian motion, 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_predef_setup (g05znc).

2 Specification

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

```
void nag_rand_field_fracbm_generate (Integer ns, Integer s, Integer m,
     double xmax, double h, const double lam[], double rho, Integer state[],
     double z[], double xx[], NagError *fail)
```

3 Description

The functions nag_rand_field_1d_predef_setup (g05znc) and nag_rand_field_fracbm_generate (g05ztc) are used to simulate a fractional Brownian motion process with Hurst argument H over an interval $[0, x_{\max}]$, using a set of equally spaced points. Fractional Brownian motion itself cannot be simulated directly using this method, since it is not a stationary Gaussian random field; however its increments can be simulated like a stationary Gaussian random field. The circulant embedding method is described in the documentation for nag_rand_field_1d_predef_setup (g05znc).

nag_rand_field_fracbm_generate (g05ztc) takes the square roots of the eigenvalues of the embedding matrix as returned by nag_rand_field_1d_predef_setup (g05znc) when **cov** = Nag_VgmBrownian, and its size M, as input and outputs S realizations of the fractional Brownian motion 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_fracbm_generate (g05ztc).

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 steps (points) to be generated in realizations of the increments of the fractional Brownian motion. This must be the same value as supplied to nag_rand_field_1d_predef_setup (g05znc) when calculating the eigenvalues of the embedding matrix.

Note: in the context of fractional Brownian motion, **ns** represents the number of *steps* from a zero starting state. Realizations returned in **z** include this starting state and so $\mathbf{ns} + 1$ values are returned for each realization..

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

2: **s** – Integer *Input*

On entry: S, the number of realizations of the fractional Brownian motion to simulate.

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

3: **m** – Integer *Input*

On entry: the size, M, 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: **xmax** – double *Input*

On entry: the upper bound for the interval over which the fractional Brownian motion is to be simulated, as input to nag_rand_field_1d_user_setup (g05zmc) or nag_rand_field_1d_predef_setup (g05znc).

Constraint: $\mathbf{xmax} > 0.0$.

5: **h** – double *Input*

On entry: the Hurst parameter, H, for the fractional Brownian motion. This must be the same value as supplied to nag_rand_field_1d_predef_setup (g05znc) in **params**[0], when the eigenvalues of the embedding matrix were calculated.

Constraint: $0.0 < \mathbf{h} < 1.0$.

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

On entry: contains 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: $\mathbf{lam}[i-1] \geq 0$, for $i = 1, 2, \ldots, \mathbf{m}$.

7: **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: $0.0 < \mathbf{rho} \leq 1.0$.

8: **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.

9: **z**[$(\mathbf{ns} + \mathbf{1}) \times \mathbf{s}$] – double *Output*

On exit: contains the realizations of the fractional Brownian motion, Z. The jth realization, for the ith point $\mathbf{xx}[i-1]$, is stored in $\mathbf{z}[(j-1) \times (\mathbf{ns}+1) + i - 1]$, for $j = 1, 2, \ldots, \mathbf{s}$ and $i = 1, 2, \ldots, \mathbf{ns} + 1$.

10: **xx**[**ns** + **1**] – double *Output*

On exit: the points at which values of the fractional Brownian motion are output. The first point is always zero, and the subsequent **ns** points represent the equispaced steps towards the last point, **xmax**. Note that in nag_rand_field_1d_user_setup (g05zmc) and nag_rand_field_1d_predef_setup (g05znc), the returned **ns** sample points are the mid-points of the grid returned in **xx** here.

11: **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_INT

On entry, **ns** = ⟨*value*⟩.
Constraint: **ns** ≥ 1.

On entry, **s** = ⟨*value*⟩.
Constraint: **s** ≥ 1.

NE_INT_2

On entry, **m** = ⟨*value*⟩, and **ns** = ⟨*value*⟩.
Constraint: **m** $\geq \max(1, 2(\mathbf{ns} - 1))$.

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, **h** = ⟨*value*⟩.
Constraint: $0.0 < \mathbf{h} < 1.0$.

On entry, **rho** = ⟨*value*⟩.
Constraint: $0.0 < \mathbf{rho} \leq 1.0$.

On entry, **xmax** = ⟨*value*⟩.
Constraint: **xmax** > 0.0.

7 Accuracy

Not applicable.

8 Parallelism and Performance

nag_rand_field_fracbm_generate (g05ztc) 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

None.

10 Example

This example calls nag_rand_field_fracbm_generate (g05ztc) to generate 5 realizations of a fractional Brownian motion over 10 steps from $x = 0.0$ to $x = 2.0$ using eigenvalues generated by nag_rand_field_1d_predef_setup (g05znc) with **cov** = Nag_VgmBrownian.

10.1 Program Text

```
/* nag_rand_field_fracbm_generate (g05ztc) 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(double *params, 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);
static void initialize_state(Integer *state);
static void display_realizations(Integer ns, Integer s, double *yy, double *z,
                                 Integer *exit_status);

int main(void)
{
  /*  Scalars */
  Integer        exit_status = 0;
  double         h, rho, var = 1.0, xmax, xmin = 0.0;
  Integer        approx, icount, m, maxm, np = 2, ns, s;
  /*  Arrays */
  double         eig[3], params[NPMAX];
  double         *lam = 0, *xx = 0, *yy = 0, *z = 0;
  Integer        state[LENST];
  /* Nag types */
  Nag_Variogram  cov = Nag_VgmBrownian;
  Nag_EmbedPad   pad;
  Nag_EmbedScale corr;
  NagError       fail;

  INIT_FAIL(fail);

  printf("nag_rand_field_fracbm_generate (g05ztc) Example Program Results\n\n");
  /* Get problem specifications from data file*/
  read_input_data(params, &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;
    }

  display_embedding_results(approx, m, rho, eig, icount);
  /* Initialize state array*/
  initialize_state(state);
  if (!(yy = NAG_ALLOC(ns+1, double)) ||
      !(z = NAG_ALLOC((ns+1)*s, double)))
    {
      printf("Allocation failure\n");
      exit_status = -2;
      goto END;
    }
  /* Compute s random field realizations.
   * nag_rand_field_fracbm_generate (g05ztc).
   * Generates s realizations of a one-dimensional random field by the
   * circulant embedding method.
   */
  h = params[0];
  nag_rand_field_fracbm_generate(ns, s, m, xmax, h, lam, rho, state, z, yy,
                                 &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_rand_field_fracbm_generate (g05ztc).\n%s\n",
             fail.message);
      exit_status = 2;
      goto END;
    }
  display_realizations(ns, s, yy, z, &exit_status);
 END:
  NAG_FREE(lam);
  NAG_FREE(xx);
  NAG_FREE(yy);
  NAG_FREE(z);
  return exit_status;
}

void read_input_data(double *params, double *xmax, Integer *ns,
                     Integer *maxm, Nag_EmbedScale *corr,
                     Nag_EmbedPad *pad, Integer *s)
{
  char nag_enum_arg[40];

#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif
  /* Read in Hurst parameter H. */
#ifdef _WIN32
  scanf_s("%lf%*[^\n]", &params[0]);
#else
  scanf("%lf%*[^\n]", &params[0]);
#endif
  /* Read in domain endpoint. */
#ifdef _WIN32
  scanf_s("%lf%*[^\n]", xmax);
#else
  scanf("%lf%*[^\n]", xmax);
#endif
  /* Read in number of sample points. */
#ifdef _WIN32
  scanf_s("%"NAG_IFMT"%*[^\n]", ns);
#else
  scanf("%"NAG_IFMT"%*[^\n]", ns);
#endif
  params[1] = *xmax/((double) (*ns));
  /* 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)
{
  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");
    }
}

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 *yy, double *z,
                          Integer *exit_status)
{
  /*  Scalars */
  Integer  indent = 0, ncols = 80;
  Integer  i, nsp1;
  /*  Arrays */
  char     **rlabs = 0;
  /* Nag types */
  NagError fail;

  INIT_FAIL(fail);

  nsp1 = ns+1;
  if (!(rlabs = NAG_ALLOC(nsp1, 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 < nsp1; i++)
    {
      if (!(rlabs[i] = NAG_ALLOC(7, char)))
        {
          printf("Allocation failure\n");
          *exit_status = -4;
          goto END;
        }
#ifdef _WIN32
      sprintf_s(rlabs[i], 7, "%6.1f", yy[i]);
#else
      sprintf(rlabs[i], "%6.1f", yy[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,
    nsp1, s, z, nsp1, "%10.5f",
    "Fractional Brownian"
    " motion realizations (x coordinate first):",
    Nag_CharacterLabels, (const char **) rlabs,
    Nag_IntegerLabels, NULL, ncols, indent, 0, &fail);
 END:
  for (i = 0; i < nsp1; i++)
    {
      NAG_FREE(rlabs[i]);
    }
  NAG_FREE(rlabs);
}
```

10.2 Program Data

```
nag_rand_field_fracbm_generate (g05ztc) Example Program Data
    0.35             : h (params[0])
    2                : xmax
   10                : ns
 2048                : maxm
 Nag_EmbedScaleOne   : corr
 Nag_EmbedPadValues  : pad
 5                   : s
```

10.3 Program Results

```
nag_rand_field_fracbm_generate (g05ztc) Example Program Results


Size of embedding matrix = 32

Approximation not required

 Fractional Brownian motion realizations (x coordinate first):
             1           2           3           4           5
 0.0    0.00000    0.00000    0.00000    0.00000    0.00000
 0.2   -0.52650   -0.16159   -0.96224   -0.40096    0.65803
 0.4   -1.81085   -0.85811   -1.43661    0.03947    0.99671
 0.6   -1.65690   -0.74802   -0.61733   -0.34685    0.05141
 0.8   -1.72240   -0.14958    0.14996    0.18134    0.26567
 1.0   -2.20349    0.46219    0.70982    0.66405    0.40706
 1.2   -2.38542    0.52085    0.36330    0.31831    0.81515
 1.4   -3.13939    0.68433    0.79826   -0.35408    1.12296
 1.6   -3.54602    0.64413    0.85751   -0.39303    1.14220
 1.8   -4.09082    1.67048    0.06038    0.30181    1.30350
 2.0   -2.97487    1.72275   -0.67253   -0.07439    1.57169
```