NAG Library Function Document

nag_wav_3d_coeff_ins (c09fzc)

1 Purpose

nag_wav_3d_coeff_ins (c09fzc) inserts a selected set of three-dimensional discrete wavelet transform (DWT) coefficients into the full set of coefficients stored in compact form, which may be later used as input to the reconstruction functions nag_idwt_3d (c09fbc) or nag_imldwt_3d (c09fdc).

2 Specification

```
#include <nag.h>
#include <nagc09.h>
```

```
void nag_wav_3d_coeff_ins (Integer ilev, Integer cindex, Integer lenc,
     double c[], const double d[], Integer ldd, Integer sdd, Integer icomm[],
     NagError *fail)
```

3 Description

nag_wav_3d_coeff_ins (c09fzc) inserts a selected set of three-dimensional DWT coefficients into the full set of coefficients stored in compact form in a one-dimensional array **c**. It is required that nag_wav_3d_coeff_ins (c09fzc) is preceded by a call to the initialization function nag_wfilt_3d (c09acc) and either the forwards transform function nag_dwt_3d (c09fac) or multi-level forwards transform function nag_mldwt_3d (c09fcc).

Given an initial three-dimensional data set A, a prior call to nag_dwt_3d (c09fac) or nag_mldwt_3d (c09fcc) computes the approximation coefficients (at the highest requested level in the case of nag_mldwt_3d (c09fcc)) and, seven sets of detail coefficients (at all levels in the case of nag_mldwt_3d (c09fcc)) and stores these in compact form in a one-dimensional array **c**. nag_wav_3d_coeff_ext (c09fyc) can then extract either the approximation coefficients or one of the sets of detail coefficients (at one of the levels following nag_mldwt_3d (c09fcc)) as three-dimensional data into the array, **d**. Following some calculation on this set of coefficients (for example, denoising), the updated coefficients in **d** are inserted back into the full set **c** using nag_wav_3d_coeff_ins (c09fzc). Several extractions and insertions may be performed. nag_idwt_3d (c09fbc) or nag_imldwt_3d (c09fdc) can then be used to reconstruct a manipulated data set $\tilde{A}$. The dimensions of the three-dimensional data stored in **d** depend on the level extracted and are available from either: the arrays **dwtlvm**, **dwtlvn** and **dwtlvfr** as returned by nag_mldwt_3d (c09fcc) if this was called first; or, otherwise from **nwct**, **nwcn** and **nwcfr** as returned by nag_wfilt_3d (c09acc). See Section 2.1 in the c09 Chapter Introduction for a discussion of the three-dimensional DWT.

4 References

None.

5 Arguments

Note: the following notation is used in this section:

n_{cm} is the number of wavelet coefficients in the first dimension. Following a call to nag_dwt_3d (c09fac) (i.e., when $\mathbf{ilev} = 0$) this is equal to $\mathbf{nwct}/(8 \times \mathbf{nwcn} \times \mathbf{nwcfr})$ as returned by nag_wfilt_3d (c09acc). Following a call to nag_mldwt_3d (c09fcc) transforming **nwl** levels, and when inserting at level $\mathbf{ilev} > 0$, this is equal to $\mathbf{dwtlvm}[\mathbf{nwl} - \mathbf{ilev}]$.

n_{cn} is the number of wavelet coefficients in the second dimension. Following a call to nag_dwt_3d (c09fac) (i.e., when $\mathbf{ilev} = 0$) this is equal to **nwcn** as returned by nag_wfilt_3d (c09acc).

Following a call to nag_mldwt_3d (c09fcc) transforming **nwl** levels, and when inserting at level **ilev** > 0, this is equal to **dwtlvn**[**nwl** − **ilev**].

n_{cfr} is the number of wavelet coefficients in the third dimension. Following a call to nag_dwt_3d (c09fac) (i.e., when **ilev** $= 0$) this is equal to **nwcfr** as returned by nag_wfilt_3d (c09acc). Following a call to nag_mldwt_3d (c09fcc) transforming **nwl** levels, and when inserting at level **ilev** > 0, this is equal to **dwtlvfr**[**nwl** − **ilev**].

1: **ilev** – Integer *Input*

On entry: the level at which coefficients are to be inserted.

If **ilev** $= 0$, it is assumed that the coefficient array **c** was produced by a preceding call to the single level function nag_dwt_3d (c09fac).

If **ilev** > 0, it is assumed that the coefficient array **c** was produced by a preceding call to the multi-level function nag_mldwt_3d (c09fcc).

Constraints:

ilev $= 0$ (following a call to nag_dwt_3d (c09fac));
$0 \leq$ **ilev** $\leq$ **nwl**, where **nwl** is as used in a preceding call to nag_mldwt_3d (c09fcc);
if **cindex** $= 0$, **ilev** $=$ **nwl** (following a call to nag_mldwt_3d (c09fcc)).

2: **cindex** – Integer *Input*

On entry: identifies which coefficients to insert. The coefficients are identified as follows:

cindex $= 0$
The approximation coefficients, produced by application of the low pass filter over columns, rows and frames of A (LLL). After a call to the multi-level transform function nag_mldwt_3d (c09fcc) (which implies that **ilev** > 0) the approximation coefficients are present only for **ilev** $=$ **nwl**, where **nwl** is the value used in a preceding call to nag_mldwt_3d (c09fcc).

cindex $= 1$
The detail coefficients produced by applying the low pass filter over columns and rows of A and the high pass filter over frames (LLH).

cindex $= 2$
The detail coefficients produced by applying the low pass filter over columns, high pass filter over rows and low pass filter over frames of A (LHL).

cindex $= 3$
The detail coefficients produced by applying the low pass filter over columns of A and high pass filter over rows and frames (LHH).

cindex $= 4$
The detail coefficients produced by applying the high pass filter over columns of A and low pass filter over rows and frames (HLL).

cindex $= 5$
The detail coefficients produced by applying the high pass filter over columns, low pass filter over rows and high pass filter over frames of A (HLH).

cindex $= 6$
The detail coefficients produced by applying the high pass filter over columns and rows of A and the low pass filter over frames (HHL).

cindex $= 7$
The detail coefficients produced by applying the high pass filter over columns, rows and frames of A (HHH).

Constraints:

if **ilev** $= 0$, $0 \leq$ **cindex** ≤ 7;
if **ilev** $=$ **nwl**, following a call to nag_mldwt_3d (c09fcc) transforming **nwl** levels, $0 \leq$ **cindex** ≤ 7;
otherwise $1 \leq$ **cindex** ≤ 7.

3: **lenc** – Integer *Input*

On entry: the dimension of the array **c**.

Constraint: **lenc** must be unchanged from the value used in the preceding call to either nag_dwt_3d (c09fac) or nag_mldwt_3d (c09fcc)..

4: **c**[**lenc**] – double *Input/Output*

On entry: contains the DWT coefficients inserted by previous calls to nag_wav_3d_coeff_ins (c09fzc), or computed by a previous call to either nag_dwt_3d (c09fac) or nag_mldwt_3d (c09fcc).

On exit: contains the same DWT coefficients provided on entry except for those identified by **ilev** and **cindex**, which are updated with the values supplied in **d**, inserted into the correct locations as expected by one of the reconstruction functions nag_idwt_3d (c09fbc) (if nag_dwt_3d (c09fac) was called previously) or nag_imldwt_3d (c09fdc) (if nag_mldwt_3d (c09fcc) was called previously).

5: **d**[*dim*] – const double *Input*

Note: the dimension, *dim*, of the array **d** must be at least **ldd** $\times$ **sdd** $\times n_{\mathrm{cfr}}$.

On entry: the coefficients to be inserted.

If the DWT coefficients were computed by nag_dwt_3d (c09fac) then

if **cindex** $= 0$, the approximation coefficients must be stored in **d**$[(k-1) \times$ **ldd** $\times$ **sdd** $+ (j-1) \times$ **ldd** $+ i - 1]$, for $i = 1, 2, \ldots, n_{\mathrm{cm}}$, $j = 1, 2, \ldots, n_{\mathrm{cn}}$ and $k = 1, 2, \ldots, n_{\mathrm{cfr}}$;

if $1 \leq$ **cindex** ≤ 7, the detail coefficients, as indicated by **cindex**, must be stored in **d**$[(k-1) \times$ **ldd** $\times$ **sdd** $+ (j-1) \times$ **ldd** $+ i - 1]$, for $i = 1, 2, \ldots, n_{\mathrm{cm}}$, $j = 1, 2, \ldots, n_{\mathrm{cn}}$ and $k = 1, 2, \ldots, n_{\mathrm{cfr}}$.

If the DWT coefficients were computed by nag_mldwt_3d (c09fcc) then

if **cindex** $= 0$ and **ilev** $=$ **nwl**, the approximation coefficients must be stored in **d**$[(k-1) \times$ **ldd** $\times$ **sdd** $+ (j-1) \times$ **ldd** $+ i - 1]$, for $i = 1, 2, \ldots, n_{\mathrm{cm}}$, $j = 1, 2, \ldots, n_{\mathrm{cn}}$ and $k = 1, 2, \ldots, n_{\mathrm{cfr}}$;

if $1 \leq$ **cindex** ≤ 7, the detail coefficients, as indicated by **cindex**, for level **ilev** must be stored in **d**$[(k-1) \times$ **ldd** $\times$ **sdd** $+ (j-1) \times$ **ldd** $+ i - 1]$, for $i = 1, 2, \ldots, n_{\mathrm{cm}}$, $j = 1, 2, \ldots, n_{\mathrm{cn}}$ and $k = 1, 2, \ldots, n_{\mathrm{cfr}}$.

6: **ldd** – Integer *Input*

On entry: the stride separating row elements of each of the sets of frame coefficients in the three-dimensional data stored in **d**.

Constraint: **ldd** $> n_{\mathrm{cm}}$.

7: **sdd** – Integer *Input*

On entry: the stride separating corresponding coefficients of consecutive frames in the three-dimensional data stored in **d**.

Constraint: **sdd** $> n_{\mathrm{cn}}$.

8: **icomm**[**260**] – Integer *Communication Array*

On entry: contains details of the discrete wavelet transform and the problem dimension as setup in the call to the initialization function nag_wfilt_3d (c09acc).

9: **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_INITIALIZATION

Either the initialization function has not been called first or **icomm** has been corrupted.

NE_INT

On entry, **cindex** = ⟨*value*⟩.
Constraint: **cindex** ≤ 7.

On entry, **cindex** = ⟨*value*⟩.
Constraint: **cindex** ≥ 0.

On entry, **ilev** = ⟨*value*⟩.
Constraint: **ilev** $= 0$ following a call to the single level function nag_dwt_3d (c09fac).

On entry, **ilev** = ⟨*value*⟩.
Constraint: **ilev** > 0 following a call to the multi-level function nag_mldwt_3d (c09fcc).

NE_INT_2

On entry, **ilev** = ⟨*value*⟩ and **nwl** = ⟨*value*⟩.
Constraint: **ilev** $\leq$ **nwl**, where **nwl** is the number of levels used in the call to nag_mldwt_3d (c09fcc).

On entry, **ldd** = ⟨*value*⟩ and n_{cm} = ⟨*value*⟩.
Constraint: **ldd** $\geq n_{\mathrm{cm}}$, where n_{cm} is the number of DWT coefficients in the first dimension following the single level transform.

On entry, **lenc** = ⟨*value*⟩ and n_{ct} = ⟨*value*⟩.
Constraint: **lenc** $\geq n_{\mathrm{ct}}$, where n_{ct} is the number of DWT coefficients computed in a previous call to nag_dwt_3d (c09fac).

On entry, **lenc** = ⟨*value*⟩ and n_{ct} = ⟨*value*⟩.
Constraint: **lenc** $\geq n_{\mathrm{ct}}$, where n_{ct} is the number of DWT coefficients computed in a previous call to nag_mldwt_3d (c09fcc).

On entry, **sdd** = ⟨*value*⟩ and n_{cn} = ⟨*value*⟩.
Constraint: **sdd** $\geq n_{\mathrm{cn}}$, where n_{cn} is the number of DWT coefficients in the second dimension following the single level transform.

NE_INT_3

On entry, **ilev** = ⟨*value*⟩ and **nwl** = ⟨*value*⟩, but **cindex** $= 0$.
Constraint: **cindex** > 0 when **ilev** $<$ **nwl** in the preceding call to nag_mldwt_3d (c09fcc).

On entry, **ldd** = ⟨*value*⟩ and $n_{\mathrm{cm}} = \langle value \rangle$.
Constraint: **ldd** $\geq n_{\mathrm{cm}}$, where n_{cm} is the number of DWT coefficients in the first dimension at the selected level **ilev**.

On entry, **sdd** = ⟨*value*⟩ and $n_{\mathrm{cn}} = \langle value \rangle$.
Constraint: **sdd** $\geq n_{\mathrm{cn}}$, where n_{cn} is the number of DWT coefficients in the second dimension at the selected level **ilev**.

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.

7 Accuracy

Not applicable.

8 Parallelism and Performance

Not applicable.

9 Further Comments

None.

10 Example

The following example demonstrates using the coefficient extraction and insertion functions in order to apply denoising using a thresholding operation. The original input data has artificial noise introduced to it, taken from a normal random number distribution. Reconstruction then takes place on both the noisy data and denoised data. The Mean Square Errors (MSE) of the two reconstructions are printed along with the reconstruction of the denoised data. The MSE of the denoised reconstruction is less than that of the noisy reconstruction.

10.1 Program Text

```
/* nag_wav_3d_coeff_ins (c09fzc) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 24, 2013.
 */
#include <stdio.h>
#include <math.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagc09.h>
#include <nagg05.h>

#define A(I,J,K) a[(K-1)*lda*sda + (J-1)*lda + I-1]
#define AN(I,J,K) an[(K-1)*lda*sda + (J-1)*lda + I-1]
#define B(I,J,K) b[(K-1)*ldb*sdb + (J-1)*ldb + I-1]
#define D(I,J,K) d[(K-1)*ldd*sdd + (J-1)*ldd + I-1]

int main(void)
```

```
{
  /* Scalars */
  Integer         exit_status = 0;
  Integer         lstate = 1, lseed = 1;
  Integer         i, j, k, lda, ldb, ldd, lenc, m, n, fr, mnfr, nf;
  Integer         nwcn, nwct, nwcfr;
  Integer         nwl, subid, genid, denoised, cindex, ilev, sda, sdb, sdd, kk;
  double          mse, thresh, var, xmu;
  /* Arrays */
  char            mode[25], wavnam[25];
  double          *a = 0,  *an = 0,  *b = 0,  *c = 0,  *d = 0,  *x = 0;
  Integer         *dwtlvm = 0,  *dwtlvn = 0,  *dwtlvfr = 0,  *state = 0;
  Integer         icomm[260], seed[1];
  /* Nag Types */
  Nag_Wavelet     wavnamenum;
  Nag_WaveletMode modenum;
  Nag_MatrixType  matrix = Nag_GeneralMatrix;
  Nag_OrderType   order = Nag_ColMajor;
  Nag_DiagType    diag = Nag_NonUnitDiag;
  NagError        fail;

  INIT_FAIL(fail);

  printf("nag_wav_3d_coeff_ins (c09fzc) Example Program Results\n\n");
  /* Skip heading in data file and read problem parameters. */
#ifdef _WIN32
  scanf_s("%*[^\n] %"NAG_IFMT "%"NAG_IFMT "%"NAG_IFMT "%*[^\n] ", &m, &n, &fr);
#else
  scanf("%*[^\n] %"NAG_IFMT "%"NAG_IFMT "%"NAG_IFMT "%*[^\n] ", &m, &n, &fr);
#endif
#ifdef _WIN32
  scanf_s("%24s%24s%*[^\n] ", wavnam, _countof(wavnam), mode, _countof(mode));
#else
  scanf("%24s%24s%*[^\n] ", wavnam, mode);
#endif

  printf("MLDWT :: Wavelet  : %s\n", wavnam);
  printf("         End mode : %s\n", mode);
  printf("         m  : %4"NAG_IFMT"\n", m);
  printf("         n  : %4"NAG_IFMT"\n", n);
  printf("         fr : %4"NAG_IFMT"\n\n", fr);

  /* Allocate arrays to hold the original data, A, original data plus noise,
   * AN, reconstruction using denoised coefficients, B, and randomly generated
   * noise, X.
   */
  lda = m;
  ldb = m;
  sda = n;
  sdb = n;
  if (!(a = NAG_ALLOC((sda)*(lda)*(fr), double)) ||
      !(an = NAG_ALLOC((sda)*(lda)*(fr), double))||
      !(b = NAG_ALLOC((sdb)*(ldb)*(fr), double))||
      !(x = NAG_ALLOC((m*n*fr), double)))
  {
    printf("Allocation failure\n");
    exit_status = 1;
    goto END;
  }

  /* nag_enum_name_to_value (x04nac).
   * Converts NAG enum member name to value.
   */
  wavnamenum = (Nag_Wavelet) nag_enum_name_to_value(wavnam);
  modenum = (Nag_WaveletMode) nag_enum_name_to_value(mode);

  /* Read in the original data. */
  for (k=1; k<=fr; k++)
  {
    for (i=1; i<=m; i++)
    {
```

```
#ifdef _WIN32
      for (j=1; j<=n; j++) scanf_s("%lf", &A(i, j, k));
#else
      for (j=1; j<=n; j++) scanf("%lf", &A(i, j, k));
#endif
#ifdef _WIN32
      scanf_s("%*[^\n]");
#else
      scanf("%*[^\n]");
#endif
    }
#ifdef _WIN32
    scanf_s("%*[^\n]");
#else
    scanf("%*[^\n]");
#endif
  }

  /* Output the original data. */
  printf("Input data :\n");
  fflush(stdout);
  for (k=1; k<=fr; k++)
  {
    nag_gen_real_mat_print_comp(order, matrix, diag, m, n, &A(1,1,k), lda,
                                "%11.4e", " ",  Nag_NoLabels, 0,
                                Nag_NoLabels, 0, 80, 0, 0, &fail);
    if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_gen_real_mat_print_comp (x04cbc).\n%s\n",
             fail.message);
      exit_status = 2;
      goto END;
    }
    printf("\n");
    fflush(stdout);
  }

  /* Set up call to nag_rand_normal (g05skc) in order to create some random
   * noise from a normal distribution to add to the original data.
   * Initial call to RNG initialiser to get size of STATE array.
   */
  seed[0] = 642521;
  genid = Nag_MersenneTwister;
  subid = 0;
  if ( !(state = NAG_ALLOC((lstate), Integer)))
  {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }

  /* nag_rand_init_repeatable (g05kfc).
   * Query the size of state.
   */
  lstate = 0;
  nag_rand_init_repeatable(genid, subid, seed, lseed, state, &lstate, &fail);
  if (fail.code != NE_NOERROR)
  {
    printf("Error from nag_rand_init_repeatable (g05kfc).\n%s\n",
           fail.message);
    exit_status = 3;
    goto END;
  }

  /* Reallocate STATE. */
  NAG_FREE(state);
  if (!(state = NAG_ALLOC((lstate), Integer)))
  {
    printf("Allocation failure\n");
    exit_status = 4;
    goto END;
  }

  /* nag_rand_init_repeatable (g05kfc).
   * Initialize the generator to a repeatable sequence.
   */
  nag_rand_init_repeatable(genid, subid, seed, lseed, state, &lstate, &fail);
  if (fail.code != NE_NOERROR)
  {
    printf("Error from nag_rand_init_repeatable (g05kfc).\n%s\n",
           fail.message);
    exit_status = 5;
    goto END;
  }

  /* Set the distribution parameters for the random noise. */
  xmu = 0.0;
  var = 0.1E-3;

  /* Generate the noise variates. */

  /* nag_rand_normal (g05skc).
   * Generates a vector of pseudorandom numbers from a Normal distribution.
   */
  mnfr = n * m * fr;
  nag_rand_normal(mnfr, xmu, var, state, x, &fail);
  if (fail.code != NE_NOERROR)
  {
    printf("Error from nag_rand_normal (g05skc).\n%s\n",
           fail.message);
    exit_status = 6;
    goto END;
  }

  /* Add the noise to the original input and save in AN */
  kk = 0;
  for (k=1; k<=n; k++)
  {
    for (j=1; j<=n; j++)
    {
      for (i=1; i<=m; i++)
      {
        AN(i, j, k) = A(i, j, k) + x[kk];
        kk = kk + 1;
      }
    }
  }

  /* Output the noisy data*/
  printf("Input data plus noise :\n");
  fflush(stdout);
  for (k=1; k<=fr; k++)
  {
    nag_gen_real_mat_print_comp(order, matrix, diag, m, n, &AN(1,1,k), lda,
                                "%11.4e", " ",
                                Nag_NoLabels, 0, Nag_NoLabels, 0, 80, 0, 0,
                                &fail);
    if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_gen_real_mat_print_comp (x04cbc).\n%s\n",
             fail.message);
      exit_status = 7;
      goto END;
    }
    printf("\n");
    fflush(stdout);
  }

  /* nag_wfilt_3d (c09acc).
   * Three-dimensional wavelet filter initialization.
   */
  nag_wfilt_3d(wavnamenum, Nag_MultiLevel, modenum, m, n, fr, &nwl, &nf, &nwct,
              &nwcn, &nwcfr, icomm, &fail);
  if (fail.code != NE_NOERROR)
  {
    printf("Error from nag_wfilt_3d (c09acc).\n%s\n",
           fail.message);
    exit_status = 8;
    goto END;
  }

  /* Allocate arrays to hold the coefficients, c, and the dimensions
   * of the coefficients at each level, dwtlvm, dwtlvn.
   */
  lenc = nwct;
  if (!(c = NAG_ALLOC((lenc), double)) ||
      !(dwtlvm = NAG_ALLOC((nwl), Integer)) ||
      !(dwtlvn = NAG_ALLOC((nwl), Integer)) ||
      !(dwtlvfr = NAG_ALLOC((nwl), Integer)))
  {
    printf("Allocation failure\n");
    exit_status = 9;
    goto END;
  }

  /* Perform a forwards multi-level transform on the noisy data. */

  /* nag_mldwt_3d (c09fcc).
   * Two-dimensional multi-level discrete wavelet transform.
   */
  nag_mldwt_3d(m, n, fr, an, lda, sda, lenc, c, nwl, dwtlvm, dwtlvn,
               dwtlvfr, icomm, &fail);
  if (fail.code != NE_NOERROR)
  {
    printf("Error from nag_mldwt_3d (c09fcc).\n%s\n",
           fail.message);
    exit_status = 10;
    goto END;
  }

  /* Reconstruct without thresholding of detail coefficients. */

  /* nag_imldwt_3d (c09fdc).
   * Two-dimensional inverse multi-level discrete wavelet transform.
   */
  nag_imldwt_3d(nwl, lenc, c, m, n, fr, b, ldb, sdb, icomm, &fail);
  if (fail.code != NE_NOERROR)
  {
    printf("Error from nag_imldwt_3d (c09fdc).\n%s\n",
           fail.message);
    exit_status = 11;
    goto END;
  }

  /* Calculate the Mean Square Error of the noisy reconstruction. */
  mse = 0.0;
  for (k=1; k<=fr; k++)
    for (j=1; j<=n; j++)
      for (i=1; i<=m; i++)
        mse = mse + pow((A(i, j, k) - B(i, j, k)), 2);
  mse = mse/(double)(m * n * fr);
  printf("Without denoising Mean Square Error is %11.4e\n\n", mse);

  /* Now perform the denoising by extracting each of the detail
   * coefficients at each level and applying hard thresholding
   * Allocate a 2D array to hold the detail coefficients
   */
  ldd = dwtlvm[nwl-1];
  sdd = dwtlvn[nwl-1];
  if (!(d = NAG_ALLOC((ldd)*(sdd)*(dwtlvfr[nwl-1]), double)))
  {
    printf("Allocation failure\n");
    exit_status = 12;
    goto END;
  }

  /* Calculate the threshold based on VisuShrink denoising. */
  thresh = sqrt(var) * sqrt(2. * log((double)(m * n * fr)));
  denoised = 0;
  /* For each level */
  for (ilev=nwl; ilev>=1; ilev-=1)
  {
    /* Select detail coefficients */
    for (cindex=1; cindex<=7; cindex++)
    {
      /* Extract coefficients into the 2D array d*/

      /* nag_wav_3d_coeff_ext (c09fyc).
       * Three-dimensional discrete wavelet transform coefficient extraction.
       */
      nag_wav_3d_coeff_ext(ilev, cindex, lenc, c, d, ldd, sdd, icomm, &fail);
      if (fail.code != NE_NOERROR)
      {
        printf("Error from nag_wav_3d_coeff_ext (c09fyc).\n%s\n",
               fail.message);
        exit_status = 13;
        goto END;
      }

      /* Perform the hard thresholding operation*/
      for (k=1; k<=dwtlvfr[nwl - ilev]; k++)
        for (j=1; j<=dwtlvn[nwl - ilev]; j++)
          for (i=1; i<=dwtlvm[nwl - ilev]; i++)
            if (fabs(D(i, j, k))< thresh)
            {
              D(i, j, k) = 0.0;
              denoised = denoised + 1;
            }

      /* Insert the denoised coefficients back into c. */

      /* nag_wav_3d_coeff_ins (c09fzc).
       * Three-dimensional discrete wavelet transform coefficient insertion.
       */
      nag_wav_3d_coeff_ins(ilev, cindex, lenc, c, d, ldd, sdd, icomm, &fail);
      if (fail.code != NE_NOERROR)
      {
        printf("Error from nag_wav_3d_coeff_ins (c09fzc).\n%s\n",
               fail.message);
        exit_status = 14;
        goto END;
      }

    }
  }

  /* Output the number of coefficients that were set to zero*/
  printf("Number of coefficients denoised is %4"NAG_IFMT" out of %4"NAG_IFMT
         "\n\n", denoised, nwct - dwtlvm[0]*dwtlvn[0]*dwtlvfr[0]);
  fflush(stdout);

  /* Reconstruct original data following thresholding of detail coefficients */

  /* nag_imldwt_3d (c09fdc).
   * Three-dimensional inverse multi-level discrete wavelet transform.
   */
  nag_imldwt_3d(nwl, lenc, c, m, n, fr, b, ldb, sdb, icomm, &fail);
  if (fail.code != NE_NOERROR)
  {
    printf("Error from nag_imldwt_3d (c09fdc).\n%s\n",
           fail.message);
    exit_status = 15;
    goto END;
  }
  /* Calculate the Mean Square Error of the denoised reconstruction. */
  mse = 0.0;
  for (k=1; k<=n; k++)
    for (j=1; j<=n; j++)
      for (i=1; i<=m; i++)
        mse = mse + pow((A(i, j, k) - B(i, j, k)), 2);
  mse = mse/(double)(m * n * fr);
  printf("With denoising Mean Square Error is %11.4e \n\n", mse);

  /* Output the denoised reconstruction. */
  printf("Reconstruction of denoised input :\n");
  fflush(stdout);
  for (k=1; k<=fr; k++)
  {
    nag_gen_real_mat_print_comp(order, matrix, diag, m, n, &B(1,1,k), ldb,
                                "%11.4e", " ",
                                Nag_NoLabels, 0, Nag_NoLabels, 0, 80, 0, 0,
                                &fail);
    if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_gen_real_mat_print_comp (x04cbc).\n%s\n",
             fail.message);
      exit_status = 16;
      goto END;
    }
    if (k<fr) printf("\n");
    fflush(stdout);
  }

  END:
  NAG_FREE(a);
  NAG_FREE(an);
  NAG_FREE(b);
  NAG_FREE(c);
  NAG_FREE(d);
  NAG_FREE(x);
  NAG_FREE(dwtlvm);
  NAG_FREE(dwtlvn);
  NAG_FREE(dwtlvfr);
  NAG_FREE(state);
  return exit_status;
}
```

10.2 Program Data

```
nag_wav_3d_coeff_ins (c09fzc) Example Program Data
 4  4  4              : m, n, fr
 Nag_Haar Nag_Periodic : wavnam, mode
0.1000E-01 0.1000E-01 0.1000E-01 0.1000E-01
0.1000E+01 0.1000E+01 0.1000E+01 0.1000E+01
0.1000E-01 0.1000E-01 0.1000E-01 0.1000E-01
0.1000E+01 0.1000E+01 0.1000E+01 0.1000E+01

0.1000E+01 0.1000E+01 0.1000E+01 0.1000E+01
0.1000E-01 0.1000E-01 0.1000E-01 0.1000E-01
0.1000E+01 0.1000E+01 0.1000E+01 0.1000E+01
0.1000E-01 0.1000E-01 0.1000E-01 0.1000E-01

0.1000E-01 0.1000E-01 0.1000E-01 0.1000E-01
0.1000E+01 0.1000E+01 0.1000E+01 0.1000E+01
0.1000E-01 0.1000E-01 0.1000E-01 0.1000E-01
0.1000E+01 0.1000E+01 0.1000E+01 0.1000E+01

0.1000E+01 0.1000E+01 0.1000E+01 0.1000E+01
0.1000E-01 0.1000E-01 0.1000E-01 0.1000E-01
0.1000E+01 0.1000E+01 0.1000E+01 0.1000E+01
0.1000E-01 0.1000E-01 0.1000E-01 0.1000E-01
```

10.3 Program Results

```
nag_wav_3d_coeff_ins (c09fzc) Example Program Results

MLDWT :: Wavelet  : Nag_Haar
         End mode : Nag_Periodic
         m  :    4
         n  :    4
         fr :    4

Input data :
   1.0000e-02  1.0000e-02  1.0000e-02  1.0000e-02
   1.0000e+00  1.0000e+00  1.0000e+00  1.0000e+00
   1.0000e-02  1.0000e-02  1.0000e-02  1.0000e-02
   1.0000e+00  1.0000e+00  1.0000e+00  1.0000e+00

   1.0000e+00  1.0000e+00  1.0000e+00  1.0000e+00
   1.0000e-02  1.0000e-02  1.0000e-02  1.0000e-02
   1.0000e+00  1.0000e+00  1.0000e+00  1.0000e+00
   1.0000e-02  1.0000e-02  1.0000e-02  1.0000e-02

   1.0000e-02  1.0000e-02  1.0000e-02  1.0000e-02
   1.0000e+00  1.0000e+00  1.0000e+00  1.0000e+00
   1.0000e-02  1.0000e-02  1.0000e-02  1.0000e-02
   1.0000e+00  1.0000e+00  1.0000e+00  1.0000e+00

   1.0000e+00  1.0000e+00  1.0000e+00  1.0000e+00
   1.0000e-02  1.0000e-02  1.0000e-02  1.0000e-02
   1.0000e+00  1.0000e+00  1.0000e+00  1.0000e+00
   1.0000e-02  1.0000e-02  1.0000e-02  1.0000e-02

Input data plus noise :
   1.3549e-02 -9.3452e-03 -3.7110e-04  3.7765e-02
   1.0015e+00  9.8416e-01  1.0007e+00  9.8894e-01
  -1.7004e-03  1.3876e-02  1.3834e-02 -4.9390e-03
   9.8993e-01  1.0070e+00  1.0049e+00  9.9833e-01

   1.0094e+00  1.0080e+00  9.9208e-01  9.9019e-01
   1.0467e-02 -8.5383e-04  1.5987e-02  1.9701e-02
   9.9938e-01  1.0044e+00  9.9562e-01  1.0014e+00
   9.0719e-03 -8.3748e-03  1.8721e-02  2.3127e-03

   5.8176e-03 -5.3098e-03  1.1169e-03  1.5873e-02
   1.0113e+00  9.8944e-01  1.0018e+00  9.9924e-01
   1.0583e-02  8.2037e-03  9.2665e-03  1.5297e-02
   1.0023e+00  1.0157e+00  1.0084e+00  9.8344e-01

   9.9692e-01  1.0010e+00  9.9042e-01  9.9677e-01
   2.2743e-02  2.1923e-03  6.2213e-03  2.1367e-02
   9.9480e-01  9.9810e-01  9.9514e-01  9.9681e-01
   1.2116e-02  1.0294e-02  1.1353e-02  2.0623e-02

Without denoising Mean Square Error is  8.0946e-05

Number of coefficients denoised is   55 out of   63

With denoising Mean Square Error is  1.4878e-05

Reconstruction of denoised input :
   5.3377e-03  5.3377e-03  1.6635e-02  1.6635e-02
   1.0026e+00  1.0026e+00  9.9133e-01  9.9133e-01
   5.5007e-03  5.5007e-03  7.6994e-03  7.6994e-03
   1.0025e+00  1.0025e+00  1.0003e+00  1.0003e+00

   1.0026e+00  1.0026e+00  9.9133e-01  9.9133e-01
   5.3377e-03  5.3377e-03  1.6635e-02  1.6635e-02
   1.0025e+00  1.0025e+00  1.0003e+00  1.0003e+00
   5.5007e-03  5.5007e-03  7.6994e-03  7.6994e-03

   7.3252e-03  7.3252e-03  1.1020e-02  1.1020e-02
   1.0006e+00  1.0006e+00  9.9694e-01  9.9694e-01
   7.7713e-03  7.7713e-03  1.3076e-02  1.3076e-02
   1.0002e+00  1.0002e+00  9.9489e-01  9.9489e-01

   1.0006e+00  1.0006e+00  9.9694e-01  9.9694e-01
   7.3252e-03  7.3252e-03  1.1020e-02  1.1020e-02
   1.0002e+00  1.0002e+00  9.9489e-01  9.9489e-01
   7.7713e-03  7.7713e-03  1.3076e-02  1.3076e-02
```