NAG Library Function Document

nag_real_partial_svd (f02wgc)

1 Purpose

nag_real_partial_svd (f02wgc) returns leading terms in the singular value decomposition (SVD) of a real general matrix and computes the corresponding left and right singular vectors.

2 Specification

```
#include <nag.h>
#include <nagf02.h>
void nag_real_partial_svd (Nag_OrderType order, Integer m, Integer n,
     Integer k, Integer ncv,
     void (*av)(Integer *iflag, Integer m, Integer n, const double x[],
          double ax[], Nag_Comm *comm),
     Integer *nconv, double sigma[], double u[], Integer pdu, double v[],
     Integer pdv, double resid[], Nag_Comm *comm, NagError *fail)
```

3 Description

nag_real_partial_svd (f02wgc) computes a few, k, of the largest singular values and corresponding vectors of an m by n matrix A. The value of k should be small relative to m and n, for example $k \sim O(\min(m,n))$. The full singular value decomposition (SVD) of an m by n matrix A is given by

$$A = U \Sigma V^{\mathrm{T}},$$

where U and V are orthogonal and Σ is an m by n diagonal matrix with real diagonal elements, σ_i, such that

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_{\min(m,n)} \geq 0.$$

The σ_i are the *singular values* of A and the first $\min(m,n)$ columns of U and V are the *left* and *right singular vectors* of A.

If U_k, V_k denote the leading k columns of U and V respectively, and if Σ_k denotes the leading principal submatrix of Σ, then

$$A_k \equiv U_k \Sigma_k V^{\mathrm{T}}{}_k$$

is the best rank-k approximation to A in both the 2-norm and the Frobenius norm.

The singular values and singular vectors satisfy

$$Av_i = \sigma_i u_i \quad \text{and} \quad A^{\mathrm{T}} u_i = \sigma_i v_i \quad \text{so that} \quad A^{\mathrm{T}} A \nu_i = \sigma_i^2 \nu_i \text{ and } AA^{\mathrm{T}} u_i = \sigma_i^2 u_i,$$

where u_i and v_i are the ith columns of U_k and V_k respectively.

Thus, for $m \geq n$, the largest singular values and corresponding right singular vectors are computed by finding eigenvalues and eigenvectors for the symmetric matrix $A^{\mathrm{T}}A$. For $m < n$, the largest singular values and corresponding left singular vectors are computed by finding eigenvalues and eigenvectors for the symmetric matrix AA^{T}. These eigenvalues and eigenvectors are found using functions from Chapter f12. You should read the f12 Chapter Introduction for full details of the method used here.

The real matrix A is not explicitly supplied to nag_real_partial_svd (f02wgc). Instead, you are required to supply a function, **av**, that must calculate one of the requested matrix-vector products Ax or $A^{\mathrm{T}}x$ for a given real vector x (of length n or m respectively).

4 References

Wilkinson J H (1978) Singular Value Decomposition – Basic Aspects *Numerical Software – Needs and Availability* (ed D A H Jacobs) Academic Press

5 Arguments

1: **order** – Nag_OrderType *Input*

On entry: the **order** argument specifies the two-dimensional storage scheme being used, i.e., row-major ordering or column-major ordering. C language defined storage is specified by **order** = Nag_RowMajor. See Section 2.3.1.3 in How to Use the NAG Library and its Documentation for a more detailed explanation of the use of this argument.

Constraint: **order** = Nag_RowMajor or Nag_ColMajor.

2: **m** – Integer *Input*

On entry: m, the number of rows of the matrix A.

Constraint: $\mathbf{m} \geq 0$.

If $\mathbf{m} = 0$, an immediate return is effected.

3: **n** – Integer *Input*

On entry: n, the number of columns of the matrix A.

Constraint: $\mathbf{n} \geq 0$.

If $\mathbf{n} = 0$, an immediate return is effected.

4: **k** – Integer *Input*

On entry: k, the number of singular values to be computed.

Constraint: $0 < \mathbf{k} < \min(\mathbf{m}, \mathbf{n}) - 1$.

5: **ncv** – Integer *Input*

On entry: the dimension of the arrays **sigma** and **resid**. This is the number of Lanczos basis vectors to use during the computation of the largest eigenvalues of $A^{\mathrm{T}}A$ $(m \geq n)$ or AA^{T} $(m < n)$.

At present there is no *a priori* analysis to guide the selection of **ncv** relative to **k**. However, it is recommended that $\mathbf{ncv} \geq 2 \times \mathbf{k} + 1$. If many problems of the same type are to be solved, you should experiment with varying **ncv** while keeping **k** fixed for a given test problem. This will usually decrease the required number of matrix-vector operations but it also increases the internal storage required to maintain the orthogonal basis vectors. The optimal 'cross-over' with respect to CPU time is problem dependent and must be determined empirically.

Constraint: $\mathbf{k} < \mathbf{ncv} \leq \min(\mathbf{m}, \mathbf{n})$.

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

av must return the vector result of the matrix-vector product Ax or $A^{\mathrm{T}}x$, as indicated by the input value of **iflag**, for the given vector x.

The specification of **av** is:

```
void av (Integer *iflag, Integer m, Integer n, const double x[],
     double ax[], Nag_Comm *comm)
```

1: **iflag** – Integer * *Input/Output*

On entry: if **iflag** $= 1$, **ax** must return the m-vector result of the matrix-vector product Ax.

If **iflag** $= 2$, **ax** must return the n-vector result of the matrix-vector product $A^{T}x$.

On exit: may be used as a flag to indicate a failure in the computation of Ax or $A^{T}x$. If **iflag** is negative on exit from **av**, nag_real_partial_svd (f02wgc) will exit immediately with **fail** set to **iflag**.

2: **m** – Integer *Input*

On entry: the number of rows of the matrix A.

3: **n** – Integer *Input*

On entry: the number of columns of the matrix A.

4: **x**[*dim*] – const double *Input*

Note: the dimension of the array **x** will be

n when **iflag** $= 1$;
m when **iflag** $= 2$.

On entry: the vector to be pre-multiplied by the matrix A or A^{T}.

5: **ax**[*dim*] – double *Output*

Note: the dimension of the array **ax** will be

m when **iflag** $= 1$;
n when **iflag** $= 2$.

On exit: if **iflag** $= 1$, contains the m-vector result of the matrix-vector product Ax.

If **iflag** $= 2$, contains the n-vector result of the matrix-vector product $A^{T}x$.

6: **comm** – Nag_Comm *

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

user – double *
iuser – Integer *
p – Pointer

The type Pointer will be `void *`. Before calling nag_real_partial_svd (f02wgc) you may allocate memory and initialize these pointers with various quantities for use by **av** when called from nag_real_partial_svd (f02wgc) (see Section 2.3.1.1 in How to Use the NAG Library and its Documentation).

7: **nconv** – Integer * *Output*

On exit: the number of converged singular values found.

8: **sigma**[**ncv**] – double *Output*

On exit: the **nconv** converged singular values are stored in the first **nconv** elements of **sigma**.

9: **u**[*dim*] – double *Output*

Note: the dimension, *dim*, of the array **u** must be at least

$\max(1, \mathbf{pdu} \times \mathbf{ncv})$ when **order** = Nag_ColMajor;
$\max(1, \mathbf{m} \times \mathbf{pdu})$ when **order** = Nag_RowMajor.

Where $\mathbf{U}(i,j)$ appears in this document, it refers to the array element

$\mathbf{u}[(j-1) \times \mathbf{pdu} + i - 1]$ when **order** = Nag_ColMajor;
$\mathbf{u}[(i-1) \times \mathbf{pdu} + j - 1]$ when **order** = Nag_RowMajor.

On exit: the left singular vectors corresponding to the singular values stored in **sigma**.

The ith element of the jth left singular vector u_j is stored in $\mathbf{U}(i,j)$, for $i = 1, 2, \ldots, m$ and $j = 1, 2, \ldots, \mathbf{nconv}$.

10: **pdu** – Integer *Input*

On entry: the stride used in the array **u**.

Constraints:

if **order** = Nag_ColMajor, $\mathbf{pdu} \geq \max(1, \mathbf{m})$;
if **order** = Nag_RowMajor, $\mathbf{pdu} \geq \mathbf{ncv}$.

11: **v**[*dim*] – double *Output*

Note: the dimension, *dim*, of the array **v** must be at least

$\max(1, \mathbf{pdv} \times \mathbf{ncv})$ when **order** = Nag_ColMajor;
$\max(1, \mathbf{n} \times \mathbf{pdv})$ when **order** = Nag_RowMajor.

Where $\mathbf{V}(i,j)$ appears in this document, it refers to the array element

$\mathbf{v}[(j-1) \times \mathbf{pdv} + i - 1]$ when **order** = Nag_ColMajor;
$\mathbf{v}[(i-1) \times \mathbf{pdv} + j - 1]$ when **order** = Nag_RowMajor.

On exit: the right singular vectors corresponding to the singular values stored in **sigma**.

The ith element of the jth right singular vector v_j is stored in $\mathbf{V}(i,j)$, for $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, \mathbf{nconv}$.

12: **pdv** – Integer *Input*

On entry: the stride used in the array **v**.

Constraints:

if **order** = Nag_ColMajor, $\mathbf{pdv} \geq \max(1, \mathbf{n})$;
if **order** = Nag_RowMajor, $\mathbf{pdv} \geq \mathbf{ncv}$.

13: **resid**[**ncv**] – double *Output*

On exit: the residual $\|Av_j - \sigma_j u_j\|$, for $m \geq n$, or $\|A^{\mathrm{T}} u_j - \sigma_j v_j\|$, for $m < n$, for each of the converged singular values σ_j and corresponding left and right singular vectors u_j and v_j.

14: **comm** – Nag_Comm *

The NAG communication argument (see Section 2.3.1.1 in How to Use the NAG Library and its Documentation).

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

The NAG error argument (see Section 2.7 in How to Use the NAG Library and its Documentation).

nag_real_partial_svd (f02wgc) returns with **fail.code** = NE_NOERROR if at least k singular values have converged and the corresponding left and right singular vectors have been computed.

6 Error Indicators and Warnings

NE_ALLOC_FAIL

Dynamic memory allocation failed.
See Section 3.2.1.2 in How to Use the NAG Library and its Documentation for further information.

NE_BAD_PARAM

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

NE_EIGENVALUES

The number of eigenvalues found to sufficient accuracy is zero.

NE_INT

On entry, $\mathbf{k} = \langle value \rangle$.
Constraint: $\mathbf{k} > 0$.

On entry, $\mathbf{m} = \langle value \rangle$.
Constraint: $\mathbf{m} \geq 0$.

On entry, $\mathbf{n} = \langle value \rangle$.
Constraint: $\mathbf{n} \geq 0$.

On entry, $\mathbf{pdu} = \langle value \rangle$.
Constraint: $\mathbf{pdu} > 0$.

On entry, $\mathbf{pdv} = \langle value \rangle$.
Constraint: $\mathbf{pdv} > 0$.

NE_INT_2

On entry, $\mathbf{pdu} = \langle value \rangle$ and $\mathbf{m} = \langle value \rangle$.
Constraint: $\mathbf{pdu} \geq \mathbf{m}$.

On entry, $\mathbf{pdu} = \langle value \rangle$ and $\mathbf{ncv} = \langle value \rangle$.
Constraint: $\mathbf{pdu} \geq \mathbf{ncv}$.

On entry, $\mathbf{pdv} = \langle value \rangle$ and $\mathbf{n} = \langle value \rangle$.
Constraint: $\mathbf{pdv} \geq \mathbf{n}$.

On entry, $\mathbf{pdv} = \langle value \rangle$ and $\mathbf{ncv} = \langle value \rangle$.
Constraint: $\mathbf{pdv} \geq \mathbf{ncv}$.

NE_INT_3

On entry, $\mathbf{m} = \langle value \rangle$, $\mathbf{n} = \langle value \rangle$ and $\mathbf{k} = \langle value \rangle$.
Constraint: $0 < \mathbf{k} < \min(\mathbf{m}, \mathbf{n}) - 1$.

NE_INT_4

On entry, $\mathbf{k} = \langle value \rangle$, $\mathbf{ncv} = \langle value \rangle$, $\mathbf{m} = \langle value \rangle$ and $\mathbf{n} = \langle value \rangle$.
Constraint: $\mathbf{k} < \mathbf{ncv} \leq \min(\mathbf{m}, \mathbf{n})$.

NE_INTERNAL_ERROR

An error occurred during an internal call. Consider increasing the size of **ncv** relative to **k**.

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 How to Use the NAG Library and its Documentation for further information.

NE_LANCZOS_ITERATION

No shifts could be applied during a cycle of the implicitly restarted Lanczos iteration.

NE_MAX_ITER

The maximum number of iterations has been reached. The maximum number of iterations $= \langle value \rangle$. The number of converged eigenvalues $= \langle value \rangle$.

NE_NO_LANCZOS_FAC

Could not build a full Lanczos factorization.

NE_NO_LICENCE

Your licence key may have expired or may not have been installed correctly.
See Section 3.6.5 in How to Use the NAG Library and its Documentation for further information.

NE_USER_STOP

On output from user-defined function **av**, **iflag** was set to a negative value, **iflag** $= \langle value \rangle$.

7 Accuracy

See Section 2.14.2 in the f08 Chapter Introduction.

8 Parallelism and Performance

nag_real_partial_svd (f02wgc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_real_partial_svd (f02wgc) 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' Notefor your implementation for any additional implementation-specific information.

9 Further Comments

None.

10 Example

This example finds the four largest singular values (σ) and corresponding right and left singular vectors for the matrix A, where A is the m by n real matrix derived from the simplest finite difference discretization of the two-dimensional kernel $k(s,t)dt$ where

$$k(s,t) = \begin{cases} s(t-1) & \text{if } 0 \leq s \leq t \leq 1 \\ t(s-1) & \text{if } 0 \leq t < s \leq 1 \end{cases}.$$

10.1 Program Text

```
/* nag_real_partial_svd (f02wgc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */
/* Pre-processor includes */
```

```
#include <stdio.h>
#include <math.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagf02.h>

#ifdef __cplusplus
extern "C"
{
#endif
  static void NAG_CALL av(Integer *iflag, Integer m, Integer n,
                          const double x[], double ax[], Nag_Comm *comm);
#ifdef __cplusplus
}
#endif
int main(void)
{
  /*Integer scalar and array declarations */
  Integer exit_status = 0;
  Integer i, m, n, nconv, ncv, nev;
  Integer pdu, pdv;
  Nag_Comm comm;
  NagError fail;
  /*Double scalar and array declarations */
  static double ruser[1] = { -1.0 };
  double *resid = 0, *sigma = 0, *u = 0, *v = 0;
  Nag_OrderType order;

  INIT_FAIL(fail);

  printf("nag_real_partial_svd (f02wgc) Example Program Results\n\n");

  /* For communication with user-supplied functions: */
  comm.user = ruser;

  /*     Skip heading in data file */
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif
#ifdef _WIN32
  scanf_s("%" NAG_IFMT "%" NAG_IFMT "%" NAG_IFMT "%" NAG_IFMT "%*[^\n]", &m,
          &n, &nev, &ncv);
#else
  scanf("%" NAG_IFMT "%" NAG_IFMT "%" NAG_IFMT "%" NAG_IFMT "%*[^\n]", &m, &n,
        &nev, &ncv);
#endif

#ifdef NAG_COLUMN_MAJOR
  order = Nag_ColMajor;
  pdu = m;
  pdv = n;
#else
  order = Nag_RowMajor;
  pdu = ncv;
  pdv = ncv;
#endif

  if (!(resid = NAG_ALLOC(m, double)) ||
      !(sigma = NAG_ALLOC(ncv, double)) ||
      !(u = NAG_ALLOC(m * ncv, double)) || !(v = NAG_ALLOC(n * ncv, double)))
  {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }
  /*
   * nag_real_partial_svd (f02wgc)
   * Computes leading terms in the singular value decomposition of
   * a real general matrix; also computes corresponding left and right
```

```
   * singular vectors.
   */
  nag_real_partial_svd(order, m, n, nev, ncv, av, &nconv, sigma, u, pdu,
                       v, pdv, resid, &comm, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_real_partial_svd (f02wgc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }
  /* Print computed residuals */
  printf("%s\n", "  Singular Value    Residual");
  for (i = 0; i < nconv; i++)
    printf("%10.5f %16.2g\n", sigma[i], resid[i]);
  printf("\n");

END:
  NAG_FREE(resid);
  NAG_FREE(sigma);
  NAG_FREE(u);
  NAG_FREE(v);

  return exit_status;
}

static void NAG_CALL av(Integer *iflag, Integer m, Integer n,
                        const double x[], double ax[], Nag_Comm *comm)
{
  Integer i, j;
  double one = 1.0, zero = 0.0;
  double h, k, s, t;

  /* Matrix vector multiply: w <- A*x or w <- Trans(A)*x. */
  if (comm->user[0] == -1.0) {
    printf("(User-supplied callback av, first invocation.)\n");
    comm->user[0] = 0.0;
  }
  h = one / (double) (m + 1);
  k = one / (double) (n + 1);
  if (*iflag == 1) {
    for (i = 0; i < m; i++)
      ax[i] = zero;
    t = zero;
    for (j = 0; j < n; j++) {
      t = t + k;
      s = zero;
      for (i = 0; i < MIN(j + 1, m); i++) {
        s = s + h;
        ax[i] = ax[i] + k * s * (t - one) * x[j];
      }
      for (i = j + 1; i < m; i++) {
        s = s + h;
        ax[i] = ax[i] + k * t * (s - one) * x[j];
      }
    }
  }
  else {
    for (i = 0; i < n; i++)
      ax[i] = zero;
    t = zero;
    for (j = 0; j < n; j++) {
      t = t + k;
      s = zero;
      for (i = 0; i < MIN(j + 1, m); i++) {
        s = s + h;
        ax[j] = ax[j] + k * s * (t - one) * x[i];
      }
      for (i = j + 1; i < m; i++) {
        s = s + h;
        ax[j] = ax[j] + k * t * (s - one) * x[i];
```

```
      }
    }
  }
  return;
}
```

10.2 Program Data

```
nag_real_partial_svd (f02wgc) Example Program Data
 100 500 4 10 : Values for m n nev and ncv
```

10.3 Program Results

```
nag_real_partial_svd (f02wgc) Example Program Results

(User-supplied callback av, first invocation.)
  Singular Value    Residual
   0.00830          2.7e-19
   0.01223          5.9e-18
   0.02381          1.2e-17
   0.11274          7.8e-17
```
