NAG Library Function Document

nag_dggsvp (f08vec)

1　Purpose

nag_dggsvp (f08vec) uses orthogonal transformations to simultaneously reduce the m by n matrix A and the p by n matrix B to upper triangular form. This factorization is usually used as a preprocessing step for computing the generalized singular value decomposition (GSVD).

2　Specification

```
#include <nag.h>
#include <nagf08.h>
```

```
void nag_dggsvp (Nag_OrderType order, Nag_ComputeUType jobu,
    Nag_ComputeVType jobv, Nag_ComputeQType jobq, Integer m, Integer p,
    Integer n, double a[], Integer pda, double b[], Integer pdb,
    double tola, double tolb, Integer *k, Integer *l, double u[],
    Integer pdu, double v[], Integer pdv, double q[], Integer pdq,
    NagError *fail)
```

3　Description

nag_dggsvp (f08vec) computes orthogonal matrices U, V and Q such that

$$
U^{\mathrm{T}}AQ = \begin{cases} \begin{array}{c} \begin{array}{ccc} n-k-l & k & l \end{array} \\ \begin{array}{c} k \\ l \\ m-k-l \end{array}\!\!\left(\begin{array}{ccc} 0 & A_{12} & A_{13} \\ 0 & 0 & A_{23} \\ 0 & 0 & 0 \end{array}\right) \end{array}, & \text{if } m-k-l \geq 0; \\[3em] \begin{array}{c} \begin{array}{ccc} n-k-l & k & l \end{array} \\ \begin{array}{c} k \\ m-k \end{array}\!\!\left(\begin{array}{ccc} 0 & A_{12} & A_{13} \\ 0 & 0 & A_{23} \end{array}\right) \end{array}, & \text{if } m-k-l < 0; \end{cases}
$$

$$
V^{\mathrm{T}}BQ = \begin{array}{c} \begin{array}{ccc} n-k-l & k & l \end{array} \\ \begin{array}{c} l \\ p-l \end{array}\!\!\left(\begin{array}{ccc} 0 & 0 & B_{13} \\ 0 & 0 & 0 \end{array}\right) \end{array}
$$

where the k by k matrix A_{12} and l by l matrix B_{13} are nonsingular upper triangular; A_{23} is l by l upper triangular if $m-k-l \geq 0$ and is $(m-k)$ by l upper trapezoidal otherwise. $(k+l)$ is the effective numerical rank of the $(m+p)$ by n matrix $\left(A^{\mathrm{T}} \quad B^{\mathrm{T}} \right)^{\mathrm{T}}$.

This decomposition is usually used as the preprocessing step for computing the Generalized Singular Value Decomposition (GSVD), see function nag_dggsvd (f08vac).

4 References

Anderson E, Bai Z, Bischof C, Blackford S, Demmel J, Dongarra J J, Du Croz J J, Greenbaum A, Hammarling S, McKenney A and Sorensen D (1999) *LAPACK Users' Guide* (3rd Edition) SIAM, Philadelphia http://www.netlib.org/lapack/lug

Golub G H and Van Loan C F (1996) *Matrix Computations* (3rd Edition) Johns Hopkins University Press, Baltimore

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 3.2.1.3 in the Essential Introduction for a more detailed explanation of the use of this argument.

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

2:     **jobu** – Nag_ComputeUType   *Input*

On entry: if **jobu** = Nag_AllU, the orthogonal matrix U is computed.

If **jobu** = Nag_NotU, U is not computed.

Constraint: **jobu** = Nag_AllU or Nag_NotU.

3:     **jobv** – Nag_ComputeVType   *Input*

On entry: if **jobv** = Nag_ComputeV, the orthogonal matrix V is computed.

If **jobv** = Nag_NotV, V is not computed.

Constraint: **jobv** = Nag_ComputeV or Nag_NotV.

4:     **jobq** – Nag_ComputeQType   *Input*

On entry: if **jobq** = Nag_ComputeQ, the orthogonal matrix Q is computed.

If **jobq** = Nag_NotQ, Q is not computed.

Constraint: **jobq** = Nag_ComputeQ or Nag_NotQ.

5:     **m** – Integer   *Input*

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

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

6:     **p** – Integer   *Input*

On entry: p, the number of rows of the matrix B.

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

7:     **n** – Integer   *Input*

On entry: n, the number of columns of the matrices A and B.

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

8:     **a**[*dim*] – double   *Input/Output*

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

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

The (i, j)th element of the matrix A is stored in

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

On entry: the m by n matrix A.

On exit: contains the triangular (or trapezoidal) matrix described in Section 3.

9:     **pda** – Integer   *Input*

On entry: the stride separating row or column elements (depending on the value of **order**) in the array **a**.

Constraints:

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

10:     **b**[*dim*] – double   *Input/Output*

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

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

The (i, j)th element of the matrix B is stored in

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

On entry: the p by n matrix B.

On exit: contains the triangular matrix described in Section 3.

11:     **pdb** – Integer   *Input*

On entry: the stride separating row or column elements (depending on the value of **order**) in the array **b**.

Constraints:

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

12:     **tola** – double   *Input*
13:     **tolb** – double   *Input*

On entry: **tola** and **tolb** are the thresholds to determine the effective numerical rank of matrix B and a subblock of A. Generally, they are set to

$$\mathbf{tola} = \max(\mathbf{m}, \mathbf{n})\|A\|\epsilon,$$
$$\mathbf{tolb} = \max(\mathbf{p}, \mathbf{n})\|B\|\epsilon,$$

where ϵ is the ***machine precision***.

The size of **tola** and **tolb** may affect the size of backward errors of the decomposition.

14:     **k** – Integer *   *Output*
15:     **l** – Integer *   *Output*

On exit: **k** and **l** specify the dimension of the subblocks k and l as described in Section 3; $(k + l)$ is the effective numerical rank of $\left(\mathbf{a}^{\mathrm{T}} \quad \mathbf{b}^{\mathrm{T}}\right)^{\mathrm{T}}$.

16: **u**[*dim*] – double *Output*

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

max(1, **pdu** × **m**) when **jobu** = Nag_AllU;
1 otherwise.

The (i, j)th element of the matrix U is stored in

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

On exit: if **jobu** = Nag_AllU, **u** contains the orthogonal matrix U.

If **jobu** = Nag_NotU, **u** is not referenced.

17: **pdu** – Integer *Input*

On entry: the stride separating row or column elements (depending on the value of **order**) in the array **u**.

Constraints:

if **jobu** = Nag_AllU, **pdu** $\geq$ max(1, **m**);
otherwise **pdu** ≥ 1.

18: **v**[*dim*] – double *Output*

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

max(1, **pdv** × **p**) when **jobv** = Nag_ComputeV;
1 otherwise.

The (i, j)th element of the matrix V is stored in

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

On exit: if **jobv** = Nag_ComputeV, **v** contains the orthogonal matrix V.

If **jobv** = Nag_NotV, **v** is not referenced.

19: **pdv** – Integer *Input*

On entry: the stride separating row or column elements (depending on the value of **order**) in the array **v**.

Constraints:

if **jobv** = Nag_ComputeV, **pdv** $\geq$ max(1, **p**);
otherwise **pdv** ≥ 1.

20: **q**[*dim*] – double *Output*

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

max(1, **pdq** × **n**) when **jobq** = Nag_ComputeQ;
1 otherwise.

The (i, j)th element of the matrix Q is stored in

q[$(j - 1) \times$ **pdq** $+ i - 1$] when **order** = Nag_ColMajor;
q[$(i - 1) \times$ **pdq** $+ j - 1$] when **order** = Nag_RowMajor.

On exit: if **jobq** = Nag_ComputeQ, **q** contains the orthogonal matrix Q.

If **jobq** = Nag_NotQ, **q** is not referenced.

21: **pdq** – Integer   *Input*

   On entry: the stride separating row or column elements (depending on the value of **order**) in the array **q**.

   Constraints:

   if **jobq** = Nag_ComputeQ, **pdq** $\geq$ max(1, **n**);
   otherwise **pdq** $\geq$ 1.

22: **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_ENUM_INT_2

   On entry, **jobq** = $\langle value \rangle$, **pdq** = $\langle value \rangle$ and **n** = $\langle value \rangle$.
   Constraint: if **jobq** = Nag_ComputeQ, **pdq** $\geq$ max(1, **n**);
   otherwise **pdq** $\geq$ 1.

   On entry, **jobu** = $\langle value \rangle$, **pdu** = $\langle value \rangle$ and **m** = $\langle value \rangle$.
   Constraint: if **jobu** = Nag_AllU, **pdu** $\geq$ max(1, **m**);
   otherwise **pdu** $\geq$ 1.

   On entry, **jobv** = $\langle value \rangle$, **pdv** = $\langle value \rangle$ and **p** = $\langle value \rangle$.
   Constraint: if **jobv** = Nag_ComputeV, **pdv** $\geq$ max(1, **p**);
   otherwise **pdv** $\geq$ 1.

NE_INT

   On entry, **m** = $\langle value \rangle$.
   Constraint: **m** $\geq$ 0.

   On entry, **n** = $\langle value \rangle$.
   Constraint: **n** $\geq$ 0.

   On entry, **p** = $\langle value \rangle$.
   Constraint: **p** $\geq$ 0.

   On entry, **pda** = $\langle value \rangle$.
   Constraint: **pda** $>$ 0.

   On entry, **pdb** = $\langle value \rangle$.
   Constraint: **pdb** $>$ 0.

   On entry, **pdq** = $\langle value \rangle$.
   Constraint: **pdq** $>$ 0.

   On entry, **pdu** = $\langle value \rangle$.
   Constraint: **pdu** $>$ 0.

   On entry, **pdv** = $\langle value \rangle$.
   Constraint: **pdv** $>$ 0.

NE_INT_2

> On entry, $\mathbf{pda} = \langle value \rangle$ and $\mathbf{m} = \langle value \rangle$.
> Constraint: $\mathbf{pda} \geq \max(1, \mathbf{m})$.

> On entry, $\mathbf{pda} = \langle value \rangle$ and $\mathbf{n} = \langle value \rangle$.
> Constraint: $\mathbf{pda} \geq \max(1, \mathbf{n})$.

> On entry, $\mathbf{pdb} = \langle value \rangle$ and $\mathbf{n} = \langle value \rangle$.
> Constraint: $\mathbf{pdb} \geq \max(1, \mathbf{n})$.

> On entry, $\mathbf{pdb} = \langle value \rangle$ and $\mathbf{p} = \langle value \rangle$.
> Constraint: $\mathbf{pdb} \geq \max(1, \mathbf{p})$.

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

The computed factorization is nearly the exact factorization for nearby matrices $(A + E)$ and $(B + F)$, where

$$\|E\|_2 = O(\epsilon)\|A\|_2 \quad \text{and} \quad \|F\|_2 = O(\epsilon)\|B\|_2,$$

and ϵ is the *machine precision*.

8 Parallelism and Performance

nag_dggsvp (f08vec) is not threaded by NAG in any implementation.

nag_dggsvp (f08vec) 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' Note for your implementation for any additional implementation-specific information.

9 Further Comments

The complex analogue of this function is nag_zggsvp (f08vsc).

10 Example

This example finds the generalized factorization

$$A = U\Sigma_1 \begin{pmatrix} 0 & S \end{pmatrix} Q^{\mathrm{T}}, \quad B = V\Sigma_2 \begin{pmatrix} 0 & T \end{pmatrix} Q^{\mathrm{T}},$$

of the matrix pair $\begin{pmatrix} A & B \end{pmatrix}$, where

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \\ 4 & 5 & 6 \\ 7 & 8 & 8 \end{pmatrix} \quad \text{and} \quad B = \begin{pmatrix} -2 & -3 & 3 \\ 4 & 6 & 5 \end{pmatrix}.$$

10.1 Program Text

```
/* nag_dggsvp (f08vec) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 23, 2011.
 */

#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagf08.h>
#include <nagf16.h>
#include <nagx02.h>
#include <nagx04.h>

int main(void)
{
  /* Scalars */
  double       eps, norm, tola, tolb;
  Integer      i, irank, j, k, l, m, n, nrows, p, pda, pdb, pdq, pdu, pdv;
  Integer      exit_status = 0;
  /* Arrays */
  double       *a = 0, *b = 0, *q = 0, *u = 0, *v = 0;

  /* Nag Types */
  NagError     fail;
  Nag_OrderType order;

#ifdef NAG_COLUMN_MAJOR
#define A(I, J) a[(J-1)*pda + I - 1]
#define B(I, J) b[(J-1)*pdb + I - 1]
  order = Nag_ColMajor;
#else
#define A(I, J) a[(I-1)*pda + J - 1]
#define B(I, J) b[(I-1)*pdb + J - 1]
  order = Nag_RowMajor;
#endif

  INIT_FAIL(fail);

  printf("nag_dggsvp (f08vec) Example Program Results\n\n");

  /* Skip heading in data file */
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif
#ifdef _WIN32
  scanf_s("%"NAG_IFMT"%"NAG_IFMT"%"NAG_IFMT"%*[^\n]", &m, &n, &p);
#else
  scanf("%"NAG_IFMT"%"NAG_IFMT"%"NAG_IFMT"%*[^\n]", &m, &n, &p);
#endif
  if (n < 0 || m < 0 || p < 0)
    {
      printf("Invalid n, m or p\n");
      exit_status = 1;
      goto END;
    }

#ifdef NAG_COLUMN_MAJOR
  pda = m;
  pdb = p;
  pdv = p;
#else
  pda = n;
  pdb = n;
  pdv = m;
#endif
```

```
    pdq = n;
    pdu = m;

  /* Allocate memory */
  if (!(a = NAG_ALLOC(m*n, double)) ||
      !(b = NAG_ALLOC(p*n, double)) ||
      !(q = NAG_ALLOC(n*n, double)) ||
      !(u = NAG_ALLOC(m*m, double)) ||
      !(v = NAG_ALLOC(p*m, double)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }

  /* Read the m by n matrix A and p by n matrix B from data file */
  for (i = 1; i <= m; ++i)
#ifdef _WIN32
    for (j = 1; j <= n; ++j) scanf_s("%lf", &A(i, j));
#else
    for (j = 1; j <= n; ++j) scanf("%lf", &A(i, j));
#endif
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif
  for (i = 1; i <= p; ++i)
#ifdef _WIN32
    for (j = 1; j <= n; ++j) scanf_s("%lf", &B(i, j));
#else
    for (j = 1; j <= n; ++j) scanf("%lf", &B(i, j));
#endif
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif

  /* get norms of A and B using nag_dge_norm (f16rac). */
  nag_dge_norm(order, Nag_OneNorm, m, n, a, pda, &norm, &fail);
  nag_dge_norm(order, Nag_OneNorm, p, n, b, pdb, &norm, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_dge_norm (f16rac).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* Get the machine precision, using nag_machine_precision (x02ajc) */
  eps = nag_machine_precision;

  tola = MAX(m, n) * norm  * eps;
  tolb = MAX(p, n) * norm  * eps;

  /* Compute the factorization of (A, B) (A = U*S*(Q**T), B = V*T*(Q**T))
   * using nag_dggsvp (f08vec).
   */
  nag_dggsvp(order, Nag_AllU, Nag_ComputeV, Nag_ComputeQ, m, p, n, a, pda, b,
             pdb, tola, tolb, &k, &l, u, pdu, v, pdv, q, pdq, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_dggsvp (f08vec).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* Print details of the generalizeed SVD */
  irank = k + l;
  printf("Numerical rank of (A**T B**T)**T (K+L)\n%5"NAG_IFMT"\n\n", irank);
  nrows = MIN(m,irank);
```

```
      fflush(stdout);
      nag_gen_real_mat_print_comp(order, Nag_UpperMatrix, Nag_NonUnitDiag, nrows,
                                  irank, &A(1, n - irank + 1), pda, "%13.4e",
                                  "Matrix S", Nag_IntegerLabels, 0,
                                  Nag_IntegerLabels, 0, 80, 0, 0, &fail);
      if (fail.code != NE_NOERROR) goto FAIL;
      printf("\n");
      fflush(stdout);
      nag_gen_real_mat_print_comp(order, Nag_UpperMatrix, Nag_NonUnitDiag, l, l,
                                  &B(1, n - l + 1), pdb, "%13.4e",
                                  "Upper triangular matrix T", Nag_IntegerLabels,
                                  0, Nag_IntegerLabels, 0, 80, 0, 0, &fail);
      if (fail.code != NE_NOERROR) goto FAIL;
      printf("\n");
      fflush(stdout);
      nag_gen_real_mat_print_comp(order, Nag_GeneralMatrix, Nag_NonUnitDiag, m,
                                  m, u, pdu, "%13.4e", "Orthogonal matrix U",
                                  Nag_IntegerLabels, 0, Nag_IntegerLabels, 0,
                                  80, 0, 0, &fail);
      if (fail.code != NE_NOERROR) goto FAIL;
      printf("\n");
      fflush(stdout);
      nag_gen_real_mat_print_comp(order, Nag_GeneralMatrix, Nag_NonUnitDiag, p, p,
                                  v, pdv, "%13.4e", "Orthogonal matrix V",
                                  Nag_IntegerLabels, 0, Nag_IntegerLabels, 0, 80,
                                  0, 0, &fail);
      if (fail.code != NE_NOERROR) goto FAIL;
      printf("\n");
      fflush(stdout);
      nag_gen_real_mat_print_comp(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, n,
                                  q, pdq, "%13.4e", "Orthogonal matrix Q",
                                  Nag_IntegerLabels, 0, Nag_IntegerLabels, 0, 80,
                                  0, 0, &fail);
 FAIL:
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_gen_real_mat_print_comp (x04cbc).\n%s\n",
             fail.message);
      exit_status = 1;
      goto END;
    }

 END:
  NAG_FREE(a);
  NAG_FREE(b);
  NAG_FREE(q);
  NAG_FREE(u);
  NAG_FREE(v);

  return exit_status;
}
```

10.2 Program Data

```
nag_dggsvp (f08vec) Example Program Data

  4    3    2   : m, n and p

  1.0  2.0  3.0
  3.0  2.0  1.0
  4.0  5.0  6.0
  7.0  8.0  8.0 : matrix A

 -2.0 -3.0  3.0
  4.0  6.0  5.0 : matrix B
```

10.3 Program Results

```
nag_dggsvp (f08vec) Example Program Results

Numerical rank of (A**T B**T)**T (K+L)
    3

 Matrix S
                1              2              3
 1   -2.0569e+00    1.0771e+01   -7.2814e+00
 2                  7.1947e+00   -7.5262e+00
 3                                5.8129e-01

 Upper triangular matrix T
                1              2
 1    8.0623e+00   -3.1305e+00
 2                 -4.9193e+00

 Orthogonal matrix U
                1              2              3              4
 1   -1.3484e-01    5.1025e-01   -2.4351e-01    8.1373e-01
 2    6.7420e-01   -5.4670e-01   -3.5349e-01    3.4874e-01
 3    2.6968e-01    4.8292e-01   -6.9127e-01   -4.6499e-01
 4    6.7420e-01    4.5558e-01    5.8129e-01    1.5127e-15

 Orthogonal matrix V
                1              2
 1   -4.4721e-01    8.9443e-01
 2    8.9443e-01    4.4721e-01

 Orthogonal matrix Q
                1              2              3
 1   -8.3205e-01    5.5470e-01    0.0000e+00
 2    5.5470e-01    8.3205e-01    0.0000e+00
 3    0.0000e+00    0.0000e+00   -1.0000e+00
```