

# NAG Library Function Document

## nag\_dggsvp3 (f08vgc)

### 1 Purpose

nag\_dggsvp3 (f08vgc) 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). For sufficiently large problems, a blocked algorithm is used to make best use of level 3 BLAS.

### 2 Specification

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

void nag_dggsvp3 (Nag_OrderType order, Nag_ComputeUType jobv,
                  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\_dggsvp3 (f08vgc) computes orthogonal matrices  $U$ ,  $V$  and  $Q$  such that

$$U^T A Q = \begin{cases} \begin{pmatrix} n-k-l & k & l \\ k & \begin{pmatrix} 0 & A_{12} & A_{13} \\ 0 & 0 & A_{23} \end{pmatrix} \\ m-k-l & \begin{pmatrix} 0 & 0 & 0 \end{pmatrix} \end{pmatrix}, & \text{if } m-k-l \geq 0; \\ \begin{pmatrix} n-k-l & k & l \\ k & \begin{pmatrix} 0 & A_{12} & A_{13} \\ 0 & 0 & A_{23} \end{pmatrix} \\ m-k & \begin{pmatrix} 0 & 0 & 0 \end{pmatrix} \end{pmatrix}, & \text{if } m-k-l < 0; \end{cases}$$

$$V^T B Q = \begin{pmatrix} n-k-l & k & l \\ l & \begin{pmatrix} 0 & 0 & B_{13} \\ 0 & 0 & 0 \end{pmatrix} \\ p-l & \begin{pmatrix} 0 & 0 & 0 \end{pmatrix} \end{pmatrix}$$

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  $(A^T \ B^T)^T$ .

This decomposition is usually used as the preprocessing step for computing the Generalized Singular Value Decomposition (GSVD), see function nag\_dtgsja (f08yec); the two steps are combined in nag\_dggsvd3 (f08vcc).

## 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 (2012) *Matrix Computations* (4th 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 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: **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:* **m**  $\geq 0$ .
- 6: **p** – Integer *Input*  
*On entry:*  $p$ , the number of rows of the matrix  $B$ .  
*Constraint:* **p**  $\geq 0$ .
- 7: **n** – Integer *Input*  
*On entry:*  $n$ , the number of columns of the matrices  $A$  and  $B$ .  
*Constraint:* **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

$$\begin{aligned} &\mathbf{a}[(j-1) \times \mathbf{pda} + i - 1] \text{ when } \mathbf{order} = \text{Nag\_ColMajor}; \\ &\mathbf{a}[(i-1) \times \mathbf{pda} + j - 1] \text{ when } \mathbf{order} = \text{Nag\_RowMajor}. \end{aligned}$$

*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:*

$$\begin{aligned} &\text{if } \mathbf{order} = \text{Nag\_ColMajor}, \mathbf{pda} \geq \max(1, \mathbf{m}); \\ &\text{if } \mathbf{order} = \text{Nag\_RowMajor}, \mathbf{pda} \geq \max(1, \mathbf{n}). \end{aligned}$$

10: **b**[*dim*] – double

*Input/Output*

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

$$\begin{aligned} &\max(1, \mathbf{pdb} \times \mathbf{n}) \text{ when } \mathbf{order} = \text{Nag\_ColMajor}; \\ &\max(1, \mathbf{p} \times \mathbf{pdb}) \text{ when } \mathbf{order} = \text{Nag\_RowMajor}. \end{aligned}$$

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

$$\begin{aligned} &\mathbf{b}[(j-1) \times \mathbf{pdb} + i - 1] \text{ when } \mathbf{order} = \text{Nag\_ColMajor}; \\ &\mathbf{b}[(i-1) \times \mathbf{pdb} + j - 1] \text{ when } \mathbf{order} = \text{Nag\_RowMajor}. \end{aligned}$$

*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:*

$$\begin{aligned} &\text{if } \mathbf{order} = \text{Nag\_ColMajor}, \mathbf{pdb} \geq \max(1, \mathbf{p}); \\ &\text{if } \mathbf{order} = \text{Nag\_RowMajor}, \mathbf{pdb} \geq \max(1, \mathbf{n}). \end{aligned}$$

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

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

where  $\epsilon$  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  $(\mathbf{a}^T \quad \mathbf{b}^T)^T$ .

16: **u**[*dim*] – double *Output*

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

$\max(1, \mathbf{pdu} \times \mathbf{m})$  when **jobu** = Nag\_AllU;  
1 otherwise.

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

**u**[(*j* – 1) × **pdu** + *i* – 1] when **order** = Nag\_ColMajor;  
**u**[(*i* – 1) × **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** ≥ max(1, **m**);  
otherwise **pdu** ≥ 1.

18: **v**[*dim*] – double *Output*

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

$\max(1, \mathbf{pdv} \times \mathbf{p})$  when **jobv** = Nag\_ComputeV;  
1 otherwise.

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

**v**[(*j* – 1) × **pdv** + *i* – 1] when **order** = Nag\_ColMajor;  
**v**[(*i* – 1) × **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** ≥ max(1, **p**);  
otherwise **pdv** ≥ 1.

20: **q**[*dim*] – double *Output*

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

$\max(1, \mathbf{pdq} \times \mathbf{n})$  when **jobq** = Nag\_ComputeQ;  
1 otherwise.

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

**q**[(*j* – 1) × **pdq** + *i* – 1] when **order** = Nag\_ColMajor;  
**q**[(*i* – 1) × **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 2.7 in How to Use the NAG Library and its Documentation).

## 6 Error Indicators and Warnings

### NE\_ALLOC\_FAIL

Dynamic memory allocation failed.  
 See Section 2.3.1.2 in How to Use the NAG Library and its Documentation 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, **pda** =  $\langle value \rangle$  and **m** =  $\langle value \rangle$ .

Constraint: **pda**  $\geq \max(1, \mathbf{m})$ .

On entry, **pda** =  $\langle value \rangle$  and **n** =  $\langle value \rangle$ .

Constraint: **pda**  $\geq \max(1, \mathbf{n})$ .

On entry, **pdb** =  $\langle value \rangle$  and **n** =  $\langle value \rangle$ .

Constraint: **pdb**  $\geq \max(1, \mathbf{n})$ .

On entry, **pdb** =  $\langle value \rangle$  and **p** =  $\langle value \rangle$ .

Constraint: **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 2.7.6 in How to Use the NAG Library and its Documentation for further information.

**NE\_NO\_LICENCE**

Your licence key may have expired or may not have been installed correctly.

See Section 2.7.5 in How to Use the NAG Library and its Documentation 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  $\epsilon$  is the *machine precision*.

**8 Parallelism and Performance**

nag\_dggsvp3 (f08vgc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag\_dggsvp3 (f08vgc) 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**

This function replaces the deprecated function nag\_dggsvp (f08vec) which used an unblocked algorithm and therefore did not make best use of level 3 BLAS functions.

The complex analogue of this function is nag\_zggsvp3 (f08vuc).

**10 Example**

This example finds the generalized factorization

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

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_dggsvp3 (f08vgc) Example Program.
*
* NAGPRODCODE Version.
*
* Copyright 2016 Numerical Algorithms Group.
*
* Mark 26, 2016.
*/

#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, ncycle, p, pda, pdb, pdq,
                pdu, pdv;
    Integer      printq, printr, printu, printv;
    Integer      exit_status = 0;
    /* Arrays */
    double      *a = 0, *b = 0, *q = 0, *u = 0, *v = 0, *alpha = 0, *beta = 0;

    /* Nag Types */
    NagError fail;
    Nag_OrderType order;
    Nag_DiagType  diag = Nag_NonUnitDiag;
    Nag_MatrixType genmat = Nag_GeneralMatrix, upmat = Nag_UpperMatrix;
    Nag_LabelType  intl = Nag_IntegerLabels;

#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_dggsvp3 (f08vgc) 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;

    /* Read in 0s or 1s to determine whether matrices U, V, Q or R are to be
     * printed.
     */
    scanf("%" NAG_IFMT "%" NAG_IFMT "%" NAG_IFMT "%" NAG_IFMT "%*[\n]",
          &prntu, &prntv, &prntq, &prntr);

    /* 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(n * n, double)) ||
        !(alpha = NAG_ALLOC(n, double)) ||
        !(beta = NAG_ALLOC(n, 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_dggsvp3 (f08vgc).
 */
nag_dggsvp3(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_dggsvp3 (f08vgc).\n%s\n", fail.message);
    exit_status = 2;
    goto END;
}

/* Compute the generalized singular value decomposition of preprocessed (A,B)
 * (A = U*D1*(O R)*(Q^T), B = V*D2*(O R)*(Q^T))
 * using nag_dtgsja (f08yec).
 */
nag_dtgsja(order, Nag_AllU, Nag_ComputeV, Nag_ComputeQ, m, p, n, k, l, a,
            pda, b, pdb, tola, tolb, alpha, beta, u, pdu, v, pdv, q, pdq,
            &ncycle, &fail);
if (fail.code != NE_NOERROR) {
    printf("Error from nag_dtgsja (f08yec).\n%s\n", fail.message);
    exit_status = 3;
    goto END;
}

/* Print the generalized singular value pairs alpha, beta */
irank = MIN(k + 1, m);
printf("Number of infinite generalized singular values (k): %5" NAG_IFMT
      "\n", k);
printf("Number of finite generalized singular values (l): %5" NAG_IFMT
      "\n", l);
printf("Effective Numerical rank of (A^T B^T)^T (k+1): %5" NAG_IFMT
      "\n", irank);
printf("\nFinite generalized singular values:\n");

for (j = k; j < irank; ++j)
    printf("%45s%12.4e\n", "", alpha[j] / beta[j]);

printf("\nNumber of cycles of the Kogbetliantz method: %12" NAG_IFMT "\n\n",
      ncycle);

if (printu) {
    fflush(stdout);
    nag_gen_real_mat_print_comp(order, genmat, diag, m, m, u, pdu, "%13.4e",
                                "Orthogonal matrix U", intlab, NULL, intlab,
                                NULL, 80, 0, NULL, &fail);
    if (fail.code != NE_NOERROR)
        goto PRINTERR;
    printf("\n");
}
if (printv) {
    fflush(stdout);
    nag_gen_real_mat_print_comp(order, genmat, diag, p, p, v, pdv, "%13.4e",
                                "Orthogonal matrix V", intlab, NULL, intlab,
                                NULL, 80, 0, NULL, &fail);
    if (fail.code != NE_NOERROR)
        goto PRINTERR;
    printf("\n");
}
if (printq) {
    fflush(stdout);
    nag_gen_real_mat_print_comp(order, genmat, diag, n, n, q, pdq, "%13.4e",
                                "Orthogonal matrix Q", intlab, NULL, intlab,

```

```

                                NULL, 80, 0, NULL, &fail);
    if (fail.code != NE_NOERROR)
        goto PRINTERR;
    printf("\n");
}
if (printr) {
    fflush(stdout);
    nag_gen_real_mat_print_comp(order, upmat, diag, irank, irank,
                                &A(1, n - irank + 1), pda, "%13.4e",
                                "Nonsingular upper triangular matrix R",
                                intlab, NULL, intlab, NULL, 80, 0, NULL,
                                &fail);
}
PRINTERR:
if (fail.code != NE_NOERROR) {
    printf("Error from nag_gen_real_mat_print_comp (x04cbc).\n%s\n",
          fail.message);
    exit_status = 4;
}
}

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

return exit_status;
}

```

## 10.2 Program Data

nag\_dggsvp3 (f08vgc) Example Program Data

```

4      3      2      : m, n and p

0      0      0      0 : printing u, v, q, r?

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\_dggsvp3 (f08vgc) Example Program Results

```

Number of infinite generalized singular values (k):      1
Number of finite generalized singular values (l):      2
Effective Numerical rank of (AT BT)T (k+l):      3

Finite generalized singular values:
                                1.3151e+00
                                8.0185e-02

Number of cycles of the Kogbetliantz method:      2

```

---