NAG Library Function Document

nag_zggsvp3 (f08vuc)

1 Purpose

nag_zggsvp3 (f08vuc) uses unitary 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_zggsvp3 (Nag_OrderType order, Nag_ComputeUType jobu,
     Nag_ComputeVType jobv, Nag_ComputeQType jobq, Integer m, Integer p,
     Integer n, Complex a[], Integer pda, Complex b[], Integer pdb,
     double tola, double tolb, Integer *k, Integer *l, Complex u[],
     Integer pdu, Complex v[], Integer pdv, Complex q[], Integer pdq,
     NagError *fail)
```

3 Description

nag_zggsvp3 (f08vuc) computes unitary matrices U, V and Q such that

$$U^{\mathrm{H}}AQ = \begin{cases} \begin{array}{c|ccc} & n-k-l & k & l \\ \hline k & 0 & A_{12} & A_{13} \\ l & 0 & 0 & A_{23} \\ m-k-l & 0 & 0 & 0 \end{array}, & \text{if } m-k-l \geq 0; \\[2ex] \begin{array}{c|ccc} & n-k-l & k & l \\ \hline k & 0 & A_{12} & A_{13} \\ m-k & 0 & 0 & A_{23} \end{array}, & \text{if } m-k-l < 0; \end{cases}$$

$$V^{\mathrm{H}}BQ = \begin{array}{c|ccc} & n-k-l & k & l \\ \hline l & 0 & 0 & B_{13} \\ p-l & 0 & 0 & 0 \end{array}$$

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 $\begin{pmatrix} A^{\mathrm{H}} & B^{\mathrm{H}} \end{pmatrix}^{\mathrm{H}}$.

This decomposition is usually used as the preprocessing step for computing the Generalized Singular Value Decomposition (GSVD), see function nag_ztgsja (f08ysc); the two steps are combined in nag_zggsvd3 (f08vqc).

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 unitary 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 unitary 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 unitary 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: $\mathbf{a}[\mathit{dim}]$ – Complex *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 $\mathbf{order}=\text{Nag\_ColMajor}$;
$\mathbf{a}[(i-1)\times\mathbf{pda}+j-1]$ when $\mathbf{order}=\text{Nag\_RowMajor}$.

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

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

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

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

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

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

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

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 $\begin{pmatrix}\mathbf{a}^{\mathrm{T}} & \mathbf{b}^{\mathrm{T}}\end{pmatrix}^{\mathrm{T}}$.

16: **u**[dim] – Complex *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

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

On exit: if **jobu** = Nag_AllU, **u** contains the unitary 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, $\mathbf{pdu} \geq \max(1, \mathbf{m})$;
otherwise $\mathbf{pdu} \geq 1$.

18: **v**[dim] – Complex *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

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

On exit: if **jobv** = Nag_ComputeV, **v** contains the unitary 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, $\mathbf{pdv} \geq \max(1, \mathbf{p})$;
otherwise $\mathbf{pdv} \geq 1$.

20: **q**[dim] – Complex *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

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

On exit: if **jobq** = Nag_ComputeQ, **q** contains the unitary 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** $\geq \max(1, \mathbf{n})$;
otherwise **pdq** ≥ 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 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_ENUM_INT_2

On entry, **jobq** = ⟨*value*⟩, **pdq** = ⟨*value*⟩ and **n** = ⟨*value*⟩.
Constraint: if **jobq** = Nag_ComputeQ, **pdq** $\geq \max(1, \mathbf{n})$;
otherwise **pdq** ≥ 1.

On entry, **jobu** = ⟨*value*⟩, **pdu** = ⟨*value*⟩ and **m** = ⟨*value*⟩.
Constraint: if **jobu** = Nag_AllU, **pdu** $\geq \max(1, \mathbf{m})$;
otherwise **pdu** ≥ 1.

On entry, **jobv** = ⟨*value*⟩, **pdv** = ⟨*value*⟩ and **p** = ⟨*value*⟩.
Constraint: if **jobv** = Nag_ComputeV, **pdv** $\geq \max(1, \mathbf{p})$;
otherwise **pdv** ≥ 1.

NE_INT

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

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

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

On entry, **pda** = ⟨*value*⟩.
Constraint: **pda** > 0.

On entry, **pdb** = ⟨*value*⟩.
Constraint: **pdb** > 0.

On entry, **pdq** = ⟨*value*⟩.
Constraint: **pdq** > 0.

On entry, **pdu** = ⟨*value*⟩.
Constraint: **pdu** > 0.

On entry, **pdv** = ⟨*value*⟩.
Constraint: **pdv** > 0.

NE_INT_2

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

On entry, **pda** = ⟨*value*⟩ and **n** = ⟨*value*⟩.
Constraint: **pda** $\geq \max(1, \mathbf{n})$.

On entry, **pdb** = ⟨*value*⟩ and **n** = ⟨*value*⟩.
Constraint: **pdb** $\geq \max(1, \mathbf{n})$.

On entry, **pdb** = ⟨*value*⟩ and **p** = ⟨*value*⟩.
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 3.6.6 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 3.6.5 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 ϵ is the ***machine precision***.

8 Parallelism and Performance

nag_zggsvp3 (f08vuc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_zggsvp3 (f08vuc) 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

This function replaces the deprecated function nag_zggsvp (f08vsc) which used an unblocked algorithm and therefore did not make best use of level 3 BLAS functions.

The real analogue of this function is nag_dggsvp3 (f08vgc).

10 Example

This example finds the generalized factorization

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

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

$$A = \begin{pmatrix} 0.96 - 0.81i & -0.03 + 0.96i & -0.91 + 2.06i & -0.05 + 0.41i \\ -0.98 + 1.98i & -1.20 + 0.19i & -0.66 + 0.42i & -0.81 + 0.56i \\ 0.62 - 0.46i & 1.01 + 0.02i & 0.63 - 0.17i & -1.11 + 0.60i \\ 0.37 + 0.38i & 0.19 - 0.54i & -0.98 - 0.36i & 0.22 - 0.20i \\ 0.83 + 0.51i & 0.20 + 0.01i & -0.17 - 0.46i & 1.47 + 1.59i \\ 1.08 - 0.28i & 0.20 - 0.12i & -0.07 + 1.23i & 0.26 + 0.26i \end{pmatrix}$$

and

$$B = \begin{pmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{pmatrix}.$$

10.1 Program Text

```
/* nag_zggsvp3 (f08vuc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

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

int main(void)
{
  /* Scalars */
  double         norm, eps, 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 */
  Complex        *a = 0, *b = 0, *q = 0, *u = 0, *v = 0;
  double         *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       intlab = Nag_IntegerLabels;
  Nag_ComplexFormType brac = Nag_BracketForm;

#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_zggsvp3 (f08vuc) 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]",
        &printu, &printv, &printq, &printr);

  /* Allocate memory */
  if (!(a = NAG_ALLOC(m * n, Complex)) ||
      !(b = NAG_ALLOC(p * n, Complex)) ||
      !(q = NAG_ALLOC(n * n, Complex)) ||
      !(u = NAG_ALLOC(m * m, Complex)) ||
      !(v = NAG_ALLOC(p * m, Complex)) ||
      !(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)
    for (j = 1; j <= n; ++j)
#ifdef _WIN32
      scanf_s(" ( %lf , %lf )", &A(i, j).re, &A(i, j).im);
#else
      scanf(" ( %lf , %lf )", &A(i, j).re, &A(i, j).im);
#endif
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif
  for (i = 1; i <= p; ++i)
    for (j = 1; j <= n; ++j)
#ifdef _WIN32
      scanf_s(" ( %lf , %lf )", &B(i, j).re, &B(i, j).im);
#else
      scanf(" ( %lf , %lf )", &B(i, j).re, &B(i, j).im);
#endif
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
```

```
  scanf("%*[^\n]");
#endif

  /* Get the machine precision, using nag_machine_precision (x02ajc) */
  eps = nag_machine_precision;
  /* Compute one-norm of A nad B using nag_zge_norm (f16uac). */
  nag_zge_norm(order, Nag_OneNorm, m, n, a, pda, &norm, &fail);
  nag_zge_norm(order, Nag_OneNorm, p, n, b, pdb, &norm, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_zge_norm (f16uac).\n%s\n", fail.message);
    exit_status = 2;
    goto END;
  }
  tola = MAX(m, n) * norm * eps;
  tolb = MAX(p, n) * norm * eps;

  /* Compute the factorization of (A, B) A = U*S*(Q^H), B = V*T*(Q^H))
   * using using nag_zggsvp3 (f08vuc).
   */
  nag_zggsvp3(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_zggsvp3 (f08vuc).\n%s\n", fail.message);
    exit_status = 3;
    goto END;
  }

  /* Compute the generalized singular value decomposition of preprocessed (A, B)
   * (A = U*D1*(0 R)*(Q^H), B = V*D2*(0 R)*(Q^H))
   * using nag_ztgsja (f08ysc).
   */
  nag_ztgsja(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_ztgsja (f08ysc).\n%s\n", fail.message);
    exit_status = 4;
    goto END;
  }

  /* Print the generalized singular value pairs alpha, beta */
  irank = MIN(k + l, 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+l): %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_complx_mat_print_comp(order, genmat, diag, m, m, u, pdu, brac,
                                  "%13.4e", "Unitary matrix U", intlab,
                                  NULL, intlab, NULL, 80, 0, NULL, &fail);
    if (fail.code != NE_NOERROR)
      goto PRINTERR;
  }
  if (printv) {
    printf("\n");
    fflush(stdout);
    nag_gen_complx_mat_print_comp(order, genmat, diag, p, p, v, pdv, brac,
                                  "%13.4e", "Unitary matrix V", intlab,
                                  NULL, intlab, NULL, 80, 0, NULL, &fail);
    if (fail.code != NE_NOERROR)
```

```
      goto PRINTERR;
  }
  if (printq) {
    printf("\n");
    fflush(stdout);
    nag_gen_complx_mat_print_comp(order, genmat, diag, n, n, q, pdq, brac,
                                  "%13.4e", "Unitary matrix Q", intlab,
                                  NULL, intlab, NULL, 80, 0, NULL, &fail);
    if (fail.code != NE_NOERROR)
      goto PRINTERR;
  }
  if (printr) {
    printf("\n");
    fflush(stdout);
    nag_gen_complx_mat_print_comp(order, upmat, diag, irank, irank,
                                  &A(1, n - irank + 1), pda, brac, "%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 = 5;
  }

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_zggsvp3 (f08vuc) Example Program Data

  6              4              2                          : m, n and p

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

( 0.96,-0.81) (-0.03, 0.96) (-0.91, 2.06) (-0.05, 0.41)
(-0.98, 1.98) (-1.20, 0.19) (-0.66, 0.42) (-0.81, 0.56)
( 0.62,-0.46) ( 1.01, 0.02) ( 0.63,-0.17) (-1.11, 0.60)
( 0.37, 0.38) ( 0.19,-0.54) (-0.98,-0.36) ( 0.22,-0.20)
( 0.83, 0.51) ( 0.20, 0.01) (-0.17,-0.46) ( 1.47, 1.59)
( 1.08,-0.28) ( 0.20,-0.12) (-0.07, 1.23) ( 0.26, 0.26) : matrix A

( 1.00, 0.00) ( 0.00, 0.00) (-1.00, 0.00) ( 0.00, 0.00)
( 0.00, 0.00) ( 1.00, 0.00) ( 0.00, 0.00) (-1.00, 0.00) : matrix B
```

10.3 Program Results

```
nag_zggsvp3 (f08vuc) Example Program Results

Number of infinite generalized singular values (k):     2
Number of   finite generalized singular values (l):     2
Effective  Numerical  rank  of (A^T B^T)^T  (k+l):     4

Finite generalized singular values:
```

```
                                               2.0720e+00
                                               1.1058e+00

Number of cycles of the Kogbetliantz method:            2
```