NAG Library Function Document

nag_zggsvp (f08vsc)

1 Purpose

nag_zggsvp (f08vsc) 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).

2 Specification

```c
#include <nag.h>
#include <nagf08.h>
void nag_zggsvp (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_zggsvp (f08vsc) computes unitary matrices $U$, $V$ and $Q$ such that

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

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

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 $(A^H B^H)^H$.

This decomposition is usually used as the preprocessing step for computing the Generalized Singular Value Decomposition (GSVD), see function nag_zggsvd (f08vnc).
5 Arguments

1: order – Nag_OrderType  
   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_CompUType  
   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_CompVType  
   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_CompQType  
   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  
   On entry: m, the number of rows of the matrix A.  
   Constraint: m ≥ 0.

6: p – Integer  
   On entry: p, the number of rows of the matrix B.  
   Constraint: p ≥ 0.

7: n – Integer  
   On entry: n, the number of columns of the matrices A and B.  
   Constraint: n ≥ 0.

8: a[dim] – Complex  
   Note: the dimension, dim, of the array a must be at least max(1, pda × n) when order = Nag_ColMajor; max(1, m × pda) when order = Nag_RowMajor.
The \((i, j)\)th element of the matrix \(A\) is stored in
\[ a[(j - 1) \times \text{pda} + i - 1] \quad \text{when } \text{order} = \text{Nag\_ColMajor}; \]
\[ a[(i - 1) \times \text{pda} + j - 1] \quad \text{when } \text{order} = \text{Nag\_RowMajor}. \]

*On entry:* the \(m\) by \(n\) matrix \(A\).

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

9: \text{pda} – Integer \hfill Input

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

*Constraints:*

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

10: \text{b}[\text{dim}] – Complex \hfill Input/Output

*Note:* the dimension, \(\text{dim}\), of the array \(b\) must be at least
\[ \max(1, \text{pdb} \times n) \quad \text{when } \text{order} = \text{Nag\_ColMajor}; \]
\[ \max(1, p \times \text{pdb}) \quad \text{when } \text{order} = \text{Nag\_RowMajor}. \]

The \((i, j)\)th element of the matrix \(B\) is stored in
\[ b[(j - 1) \times \text{pdb} + i - 1] \quad \text{when } \text{order} = \text{Nag\_ColMajor}; \]
\[ b[(i - 1) \times \text{pdb} + j - 1] \quad \text{when } \text{order} = \text{Nag\_RowMajor}. \]

*On entry:* the \(p\) by \(n\) matrix \(B\).

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

11: \text{pdb} – Integer \hfill Input

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

*Constraints:*

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

12: \text{tola} – double \hfill Input

13: \text{tolb} – double \hfill Input

*On entry:* \(\text{tola}\) and \(\text{tolb}\) are the thresholds to determine the effective numerical rank of matrix \(B\) and a subblock of \(A\). Generally, they are set to
\[ \text{tola} = \max(m, n)\|A\|\epsilon, \]
\[ \text{tolb} = \max(p, n)\|B\|\epsilon, \]

where \(\epsilon\) is the *machine precision*.

The size of \(\text{tola}\) and \(\text{tolb}\) may affect the size of backward errors of the decomposition.

14: \text{k} – Integer \hfill Output

15: \text{l} – Integer \hfill 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 \((a^T b^T)^T\).
Note: the dimension, \( \text{dim} \), of the array \( u \) must be at least
\[
\max(1, \text{pdu} \times m)
\]when \( \text{jobu} = \text{Nag}_\text{AllU} \);
1 otherwise.

The \((i,j)\)th element of the matrix \( U \) is stored in
\[
u[(j - 1) \times \text{pdu} + i - 1] \quad \text{when } \text{order} = \text{Nag}_\text{ColMajor};
u[(i - 1) \times \text{pdu} + j - 1] \quad \text{when } \text{order} = \text{Nag}_\text{RowMajor}.
\]

On exit: if \( \text{jobu} = \text{Nag}_\text{AllU}, \) \( u \) contains the unitary matrix \( U \).

If \( \text{jobu} = \text{Nag}_\text{NotU}, \) \( u \) is not referenced.

17: \( \text{pdu} \) – Integer

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

Constraints:
\[
\begin{align*}
&\text{if } \text{jobu} = \text{Nag}_\text{AllU}, \text{ pdu} \geq \max(1, m); \\
&\text{otherwise } \text{ pdu} \geq 1.
\end{align*}
\]

18: \( v[\text{dim}] \) – Complex

Note: the dimension, \( \text{dim} \), of the array \( v \) must be at least
\[
\max(1, \text{pdv} \times p)
\]when \( \text{jobv} = \text{Nag}_\text{ComputeV} \);
1 otherwise.

The \((i,j)\)th element of the matrix \( V \) is stored in
\[
v[(j - 1) \times \text{pdv} + i - 1] \quad \text{when } \text{order} = \text{Nag}_\text{ColMajor};
v[(i - 1) \times \text{pdv} + j - 1] \quad \text{when } \text{order} = \text{Nag}_\text{RowMajor}.
\]

On exit: if \( \text{jobv} = \text{Nag}_\text{ComputeV}, \) \( v \) contains the unitary matrix \( V \).

If \( \text{jobv} = \text{Nag}_\text{NotV}, \) \( v \) is not referenced.

19: \( \text{pdv} \) – Integer

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

Constraints:
\[
\begin{align*}
&\text{if } \text{jobv} = \text{Nag}_\text{ComputeV}, \text{ pdv} \geq \max(1, p); \\
&\text{otherwise } \text{ pdv} \geq 1.
\end{align*}
\]

20: \( q[\text{dim}] \) – Complex

Note: the dimension, \( \text{dim} \), of the array \( q \) must be at least
\[
\max(1, \text{pdq} \times n)
\]when \( \text{jobq} = \text{Nag}_\text{ComputeQ} \);
1 otherwise.

The \((i,j)\)th element of the matrix \( Q \) is stored in
\[
q[(j - 1) \times \text{pdq} + i - 1] \quad \text{when } \text{order} = \text{Nag}_\text{ColMajor};
q[(i - 1) \times \text{pdq} + j - 1] \quad \text{when } \text{order} = \text{Nag}_\text{RowMajor}.
\]

On exit: if \( \text{jobq} = \text{Nag}_\text{ComputeQ}, \) \( q \) contains the unitary matrix \( Q \).

If \( \text{jobq} = \text{Nag}_\text{NotQ}, \) \( q \) 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 ≥ max(1, n);
   otherwise pdq ≥ 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 <value> had an illegal value.

NE_ENUM_INT_2

   On entry, jobq = <value>, pdq = <value> and n = <value>.
   Constraint: if jobq = Nag_ComputeQ, pdq ≥ max(1, n);
   otherwise pdq ≥ 1.

   On entry, jobu = <value>, pdu = <value> and m = <value>.
   Constraint: if jobu = Nag_AllU, pdu ≥ max(1, m);
   otherwise pdu ≥ 1.

   On entry, jobv = <value>, pdv = <value> and p = <value>.
   Constraint: if jobv = Nag_ComputeV, pdv ≥ max(1, 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, \( pda = \langle \text{value} \rangle \) and \( m = \langle \text{value} \rangle \).
Constraint: \( pda \geq \max(1, m) \).

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

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

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

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 \( \epsilon \) is the machine precision.

8 Parallelism and Performance

\texttt{nag_zggsvp (f08vsc)} is not threaded by NAG in any implementation.

\texttt{nag_zggsvp (f08vsc)} 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 real analogue of this function is \texttt{nag_dggsvp (f08vec)}.

10 Example

This example finds the generalized factorization
\[
A = U \Sigma_1 (0 \quad S) Q^H, \quad B = V \Sigma_2 (0 \quad T) Q^H,
\]
of the matrix pair \( (A \quad B) \), 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

$$B = \begin{pmatrix}
1 & 0 & -1 & 0 \\
0 & 1 & 0 & -1
\end{pmatrix}.$$
#endif
if (n < 0 || m < 0 || p < 0)
{
    printf("Invalid n, m or p\n");
    exit_status = 1;
    goto END;
}
#else
    pda = m;
pdb = p;
    pdv = P;
#endif
pda = n;
pdb = n;
    pdv = m;
#endif
pdq = n;
    pdu = m;

/* 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))))
{
    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)
#endif _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 _WIN32
    scanf_s(\%*[\'\n");
#else
    scanf(\%*[\'\n");
#endif _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 _WIN32
    scanf_s(\%*[\'\n");
#else
    scanf("\%*[\'\n");
#endif _WIN32
    scanf_s(" %lf , %lf ", &A(i, j).re, &A(i, j).im);
#else
    scanf(" %lf , %lf ", &B(i, j).re, &B(i, j).im);
#endif _WIN32
    scanf_s(" %lf , %lf ", &A(i, j).re, &A(i, j).im);
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 = 1;
    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_zggsvp (f08vsc).
*/

nag_zggsvp(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_zggsvp (f08vsc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

/* Print details of the generalized SVD */
irank = k + l;
printf("Numerical rank of (A**H B**H)**H (K+L)\n%5"NAG_IFMT"\n
", irank);
nrows = MIN(m,irank);
fflush(stdout);
nag_gen_complx_mat_print_comp(order, Nag_UpperMatrix, Nag_NonUnitDiag, nrows,
                               &A(1, n - irank + 1), pda,
                               Nag_BracketForm, "%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_complx_mat_print_comp(order, Nag_UpperMatrix, Nag_NonUnitDiag, l, l,
                               &B(1, n - l + 1), pdb, Nag_BracketForm,
                               "%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_complx_mat_print_comp(order, Nag_GeneralMatrix, Nag_NonUnitDiag,
                               m, m, u, pdu, Nag_BracketForm, "%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_complx_mat_print_comp(order, Nag_GeneralMatrix, Nag_NonUnitDiag,
                               p, p, v, pdv, Nag_BracketForm, "%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_complx_mat_print_comp(order, Nag_GeneralMatrix, Nag_NonUnitDiag,
                               n, n, q, pdq, Nag_BracketForm, "%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_complx_mat_print_comp (x04dbc).\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_zggsvp (f08vsc) Example Program Data**

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>4</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>( 0.96, -0.81)</td>
<td>( -0.03, 0.96)</td>
<td>( -0.91, 2.06)</td>
<td>( -0.05, 0.41)</td>
</tr>
<tr>
<td>( -0.98, 1.98)</td>
<td>( -1.20, 0.19)</td>
<td>( -0.66, 0.42)</td>
<td>( -0.81, 0.56)</td>
</tr>
<tr>
<td>( 0.62, -0.46)</td>
<td>( 1.01, 0.02)</td>
<td>( 0.63, -0.17)</td>
<td>( -1.11, 0.60)</td>
</tr>
<tr>
<td>( 0.37, 0.38)</td>
<td>( 0.19, -0.54)</td>
<td>( -0.98, -0.36)</td>
<td>( 0.22, -0.20)</td>
</tr>
<tr>
<td>( 0.83, 0.51)</td>
<td>( 0.20, 0.01)</td>
<td>( -0.17, -0.46)</td>
<td>( 1.47, 1.59)</td>
</tr>
<tr>
<td>( 1.08, -0.28)</td>
<td>( 0.20, -0.12)</td>
<td>( -0.07, 1.23)</td>
<td>( 0.26, 0.26)</td>
</tr>
</tbody>
</table>

: matrix A

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00, 0.00)</td>
<td>( 0.00, 0.00)</td>
<td>( -1.00, 0.00)</td>
<td>( 0.00, 0.00)</td>
</tr>
<tr>
<td>( 0.00, 0.00)</td>
<td>( 1.00, 0.00)</td>
<td>( 0.00, 0.00)</td>
<td>( -1.00, 0.00)</td>
</tr>
</tbody>
</table>

: matrix B

### 10.3 Program Results

**nag_zggsvp (f08vsc) Example Program Results**

Numerical rank of \((A^H B^H)^H (K+L)\)

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Matrix S**

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( -2.7118e+00, 0.0000e+00)</td>
<td>( -1.4390e+00, -1.0315e+00)</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>( -1.8583e+00, 0.0000e+00)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Upper triangular matrix \(T\)**

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( 1.4142e+00, 0.0000e+00)</td>
<td>( 0.0000e+00, 0.0000e+00)</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>( 1.4142e+00, 0.0000e+00)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Orthogonal matrix \(U\)**

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( 1.0000e+00, 0.0000e+00)</td>
<td>( 0.0000e+00, -1.3015e+00)</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>( 0.0000e+00, -1.3015e+00)</td>
<td>( -1.3038e-02, -3.2595e-01)</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>( 4.2764e-01, -6.2582e-01)</td>
<td>( 8.6298e-02, -3.8174e-02)</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>( 1.5906e-01, -5.2151e-03)</td>
<td>( 1.4142e+00, 0.0000e+00)</td>
<td></td>
</tr>
</tbody>
</table>

**Orthogonal matrix \(V\)**

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( 1.0000e+00, 0.0000e+00)</td>
<td>( 0.0000e+00, 0.0000e+00)</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>( 0.0000e+00, 0.0000e+00)</td>
<td>( 1.0000e+00, 0.0000e+00)</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Orthogonal matrix $Q$

1 ( 7.0711e-01, 0.0000e+00) ( 0.0000e+00, 0.0000e+00)
2 ( 0.0000e+00, 0.0000e+00) ( 7.0711e-01, 0.0000e+00)
3 ( 7.0711e-01, 0.0000e+00) ( 0.0000e+00, 0.0000e+00)
4 ( 0.0000e+00, 0.0000e+00) ( 7.0711e-01, 0.0000e+00)

3 ( 7.0711e-01, 0.0000e+00) ( 0.0000e+00, 0.0000e+00)
2 ( 0.0000e+00, 0.0000e+00) ( 7.0711e-01, 0.0000e+00)
4 ( -7.0711e-01, 0.0000e+00) ( 0.0000e+00, 0.0000e+00)
4 ( 0.0000e+00, 0.0000e+00) ( -7.0711e-01, 0.0000e+00)

---