NAG Library Function Document

nag_zggrqf (f08ztc)

1 Purpose

nag_zggrqf (f08ztc) computes a generalized $RQ$ factorization of a complex matrix pair $(A, B)$, where $A$ is an $m$ by $n$ matrix and $B$ is a $p$ by $n$ matrix.

2 Specification

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

void nag_zggrqf (Nag_OrderType order, Integer m, Integer p, Integer n,
                 Complex a[], Integer pda, Complex taua[], Complex b[], Integer pdb,
                 Complex taub[], NagError *fail)
```

3 Description

nag_zggrqf (f08ztc) forms the generalized $RQ$ factorization of an $m$ by $n$ matrix $A$ and a $p$ by $n$ matrix $B$

$$A = RQ, \quad B = ZTQ,$$

where $Q$ is an $n$ by $n$ unitary matrix, $Z$ is a $p$ by $p$ unitary matrix and $R$ and $T$ are of the form

$$R = \begin{cases} 
  m \begin{pmatrix} n - m & m \\ 0 & R_{12} \end{pmatrix}; & \text{if } m \leq n, \\
  m - n \begin{pmatrix} R_{11} \\ n \end{pmatrix}; & \text{if } m > n,
\end{cases}$$

with $R_{12}$ or $R_{21}$ upper triangular,

$$T = \begin{cases} 
  p \begin{pmatrix} n \\ p - n \end{pmatrix}; & \text{if } p \geq n, \\
  p \begin{pmatrix} T_{11} & T_{12} \end{pmatrix}; & \text{if } p < n,
\end{cases}$$

with $T_{11}$ upper triangular.

In particular, if $B$ is square and nonsingular, the generalized $RQ$ factorization of $A$ and $B$ implicitly gives the $RQ$ factorization of $AB^{-1}$ as

$$AB^{-1} = (RT^{-1})Z^H.$$

4 References


5 Arguments

1: \texttt{order} – Nag\_OrderType \hspace{1cm} \textit{Input}

\textit{On entry}: the \texttt{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 \texttt{order} = Nag\_RowMajor. See Section 3.2.1.3 in the Essential Introduction for a more detailed explanation of the use of this argument.

\textit{Constraint}: \texttt{order} = Nag\_RowMajor or Nag\_ColMajor.

2: \texttt{m} – Integer \hspace{1cm} \textit{Input}

\textit{On entry}: \texttt{m}, the number of rows of the matrix \texttt{A}.

\textit{Constraint}: \texttt{m} \geq 0.

3: \texttt{p} – Integer \hspace{1cm} \textit{Input}

\textit{On entry}: \texttt{p}, the number of rows of the matrix \texttt{B}.

\textit{Constraint}: \texttt{p} \geq 0.

4: \texttt{n} – Integer \hspace{1cm} \textit{Input}

\textit{On entry}: \texttt{n}, the number of columns of the matrices \texttt{A} and \texttt{B}.

\textit{Constraint}: \texttt{n} \geq 0.

5: \texttt{a[dim]} – Complex \hspace{1cm} \textit{Input/Output}

\textit{Note}: the dimension, \texttt{dim}, of the array \texttt{a} must be at least

\[
\max(1, \texttt{pda} \times \texttt{n}) \text{ when } \texttt{order} = \texttt{Nag\_ColMajor}; \\
\max(1, \texttt{m} \times \texttt{pda}) \text{ when } \texttt{order} = \texttt{Nag\_RowMajor}.
\]

Where \texttt{A(i,j)} appears in this document, it refers to the array element

\[
\texttt{a}\left(\left(j - 1\right) \times \texttt{pda} + i - 1\right) \text{ when } \texttt{order} = \texttt{Nag\_ColMajor}; \\
\texttt{a}\left(\left(i - 1\right) \times \texttt{pda} + j - 1\right) \text{ when } \texttt{order} = \texttt{Nag\_RowMajor}.
\]

\textit{On entry}: the \texttt{m} by \texttt{n} matrix \texttt{A}.

\textit{On exit}: if \texttt{m} \leq \texttt{n}, the upper triangle of the subarray \texttt{A}(1 : \texttt{m}, \texttt{n} - \texttt{m} + 1 : \texttt{n}) contains the \texttt{m} by \texttt{m} upper triangular matrix \texttt{R}_{12}.

If \texttt{m} \geq \texttt{n}, the elements on and above the \texttt{(m} - \texttt{n})th subdiagonal contain the \texttt{m} by \texttt{n} upper trapezoidal matrix \texttt{R}; the remaining elements, with the array \texttt{tau}, represent the unitary matrix \texttt{Q} as a product of \texttt{min(m, n)} elementary reflectors (see Section 3.3.6 in the f08 Chapter Introduction).

6: \texttt{pda} – Integer \hspace{1cm} \textit{Input}

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

\textit{Constraints}:

\[
\text{if } \texttt{order} = \texttt{Nag\_ColMajor}, \texttt{pda} \geq \max(1, \texttt{m}); \\
\text{if } \texttt{order} = \texttt{Nag\_RowMajor}, \texttt{pda} \geq \max(1, \texttt{n}).
\]
7: \( \texttt{taua}[\min(m, n)] \) – Complex 
   \textit{Output}

   \textit{On exit:} the scalar factors of the elementary reflectors which represent the unitary matrix \( Q \).

8: \( \texttt{b}[\text{dim}] \) – Complex 
   \textit{Input/Output}

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

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

   \textit{On entry:} the \( p \) by \( n \) matrix \( B \).

   \textit{On exit:} the elements on and above the diagonal of the array contain the \( \min(p, n) \) by \( n \) upper trapezoidal matrix \( T \) (\( T \) is upper triangular if \( p \geq n \)); the elements below the diagonal, with the array \( \texttt{taub} \), represent the unitary matrix \( Z \) as a product of elementary reflectors (see Section 3.3.6 in the f08 Chapter Introduction).

9: \( \texttt{pdb} \) – Integer 
   \textit{Input}

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

   \textit{Constraints:}
   \[
   \begin{align*}
   &\text{if order = Nag\_ColMajor, } \texttt{pdb} \geq \max(1, p); \\
   &\text{if order = Nag\_RowMajor, } \texttt{pdb} \geq \max(1, n).
   \end{align*}
   \]

10: \( \texttt{taub}[\min(p, n)] \) – Complex 
    \textit{Output}

    \textit{On exit:} the scalar factors of the elementary reflectors which represent the unitary matrix \( Z \).

11: \( \texttt{fail} \) – NagError * 
   \textit{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 \( \langle \text{value} \rangle \) had an illegal value.

**NE_INT**

On entry, \( m = \langle \text{value} \rangle \).

Constraint: \( m \geq 0 \).

On entry, \( n = \langle \text{value} \rangle \).

Constraint: \( n \geq 0 \).

On entry, \( p = \langle \text{value} \rangle \).

Constraint: \( p \geq 0 \).

On entry, \( \texttt{pdb} = \langle \text{value} \rangle \).

Constraint: \( \texttt{pdb} > 0 \).
On entry, \( \text{pdb} = \langle \text{value} \rangle \).
Constraint: \( \text{pdb} > 0 \).

**NE_INT_2**

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

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

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

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

**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 generalized \( RQ \) factorization is 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**

\text{nag_zggrqf (f08ztc)} is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

\text{nag_zggrqf (f08ztc)} 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 unitary matrices \( Q \) and \( Z \) may be formed explicitly by calls to \text{nag_zungrq (f08cwc)} and \text{nag_zungqr (f08atc)} respectively. \text{nag_zunmrq (f08cxc)} may be used to multiply \( Q \) by another matrix and \text{nag_zunmqr (f08auc)} may be used to multiply \( Z \) by another matrix.

The real analogue of this function is \text{nag_dggrqf (f08zfc)}. 
10 Example

This example solves the least squares problem

\[
\minimize_x \| c - Ax \|_2 \quad \text{subject to} \quad Bx = d
\]

where

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

\[
B = \begin{pmatrix}
1 & 0 & -1 & 0 \\
0 & 1 & 0 & -1
\end{pmatrix}, \quad
C = \begin{pmatrix}
-2.54 + 0.09i \\
1.65 - 2.26i \\
-2.11 - 3.96i \\
1.82 + 3.30i \\
-6.41 + 3.77i \\
2.07 + 0.66i
\end{pmatrix}
\]

\[
d = \begin{pmatrix}
0 \\
0
\end{pmatrix}
\]

The constraints \( Bx = d \) correspond to \( x_1 = x_3 \) and \( x_2 = x_4 \).

The solution is obtained by first obtaining a generalized \( RQ \) factorization of the matrix pair \( (A, B) \). The example illustrates the general solution process, although the above data corresponds to a simple weighted least squares problem.

10.1 Program Text

/* nag_zggrqf (f08ztc) Example Program. */
/* Copyright 2014 Numerical Algorithms Group. */
/* Mark 23, 2011. */

#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <naga02.h>
#include <nagf07.h>
#include <nagf08.h>
#include <nagf16.h>

int main(void)
{
    /* Scalars */
    Complex alpha, beta;
    double rnorm;
    Integer i, j, m, n, p, pda, pdb, pdc, pdd, pdx;
    Integer y1rows, y2rows, y3rows;
    Integer exit_status = 0;

    /* Arrays */
    Complex *a = 0, *b = 0, *c = 0, *d = 0, *taua = 0, *taub = 0, *x = 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
    order = Nag_RowMajor;
    #endif

    /* other code... */
}
```c
#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_zggrqf (f08ztc) Example Program Results\n\n");

/* Skip heading in data file */
#endif _WIN32
scanf_s("%*[\n");
#else
scanf("%*[\n");
#endif _WIN32
scanf("%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;
pdc = m;
pdd = p;
pdx = n;
#else
pda = n;
pdb = n;
pdc = 1;
pdd = 1;
pdx = 1;
#endif

/* Allocate memory */
if (!(a = NAG_ALLOC(m*n, Complex)) ||
    !(b = NAG_ALLOC(p*n, Complex)) ||
    !(c = NAG_ALLOC(m, Complex)) ||
    !(d = NAG_ALLOC(p, Complex)) ||
    !(taua = NAG_ALLOC(MIN(m, n), Complex)) ||
    !(taub = NAG_ALLOC(MIN(p, n), Complex)) ||
    !(x = NAG_ALLOC(n, Complex)))
{
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
}

/* Read A, B, c and d from data file for the problem
* min||c-Ax||_2, x in R^n and Bx=d
*/
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 _WIN32
#endif
```

---

Additional information about the code:

The above code snippet is from the NAG Library, specifically for the function `f08ztc`. This function is used for computing the generalized Schur decomposition of a complex matrix pair. The code includes the necessary definitions, initialization, file reading, and memory allocation steps. The comments within the code provide a high-level understanding of the operations being performed, such as reading data from a file and allocating memory for matrices. The code is designed to handle various scenarios, including column-major and row-major order, and includes error handling for invalid inputs.
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
#endif _WIN32

for (i = 0; i < m; ++i) scanf_s(" ( %lf , %lf )", &c[i].re, &c[i].im);
#endif _WIN32
#endif _WIN32

for (i = 0; i < p; ++i) scanf_s(" ( %lf , %lf )", &d[i].re, &d[i].im);
#endif _WIN32
#endif _WIN32

/* First compute the generalized RQ factorization of (B,A) as
B = (0 R12)*Q, A = Z*(T11 T12 T13)*Q = T*Q,
where R12, T11 and T22 are upper triangular,
using nag_zggrqf (f08ztc).
*/
nag_zggrqf(order, p, m, n, b, pdb, taub, a, pda, taua, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_zggrqf (f08ztc).
%s
", fail.message);
    exit_status = 1;
    goto END;
}

/* Now, Z^H * (c-Ax) = Z^H * c - T*Q*x, and
let f = (f1) = Z^H * (c1) => minimize ||f - T*Q*x||
  (f2) (c2)
* Compute f using nag_zunmqr (f08auc), storing result in c
*/
nag_zunmqr(order, Nag_LeftSide, Nag_ConjTrans, m, 1, MIN(m, n), a, pda, taua,
            c, pdc, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_zunmqr (f08auc).
%s
", fail.message);
    exit_status = 1;
    goto END;
}

/* Putting Q*x = (y1), B * x = d becomes (O R12) (y1) = d;
  (w ) (w )
* => R12 * w = d. 
* Solve for w using nag_dtrtrs (f07tec), storing result in d;
* R12 is (p by p) triangular submatrix starting at B(1,n-p+1).
*/
nag_ztrtrs(order, Nag_Upper, Nag_NoTrans, Nag_NonUnitDiag, p, 1,
            &B(1, n - p + 1), pdb, d, pdd, &fail);
if (fail.code != NE_NOERROR)
printf("Error from nag_ztrtrs (f07tsc).\n\n", fail.message);
exit_status = 1;
goto END;
}

/* The problem now reduces to finding the minimum norm of
* g = (g1) = (f1) - T11*y1 - (T12 T13)*w
* (g2) (f2) - (T22 T23)*w.
* Form c1 = f1 - (T12 T13)*w using nag_zgemv (f16sac).
*/
alpha = nag_complex(-1.0,0.0);
beta = nag_complex(1.0,0.0);
y1rows = n - p;
nag_zgemv(order, Nag_NoTrans, y1rows, p, alpha, &A(1, n-p+1), pda, d, 1,
beta, c, l, &fail);
if (fail.code != NE_NOERROR)
{
printf("Error from nag_zgemv (f16sac).\n\n", fail.message);
exit_status = 1;
goto END;
}

/* => now (g1) = c - T11*y1 and ||g1|| = 0 when T11 * y1 = c1.
* Solve this for y1 using nag_ztrtrs (f07tsc) storing result in c1.
*/
nag_ztrtrs(order, Nag_Upper, Nag_NoTrans, Nag_NonUnitDiag, y1rows, 1, a, pda,
c, pdc, &fail);
if (fail.code != NE_NOERROR)
{
printf("Error from nag_ztrtrs (f07tsc).\n\n", fail.message);
exit_status = 1;
goto END;
}

/* So now Q*x = (y1) is stored in (c1), which is now copied to x.
* (w ) (d )
* for (i = 0; i < y1rows; ++i) x[i] = nag_complex(c[i].re,c[i].im);
* for (i = 0; i < n-y1rows; ++i) x[y1rows+i] = nag_complex(d[i].re,d[i].im);
*/

/* Compute x by applying Q inverse using nag_zunmrq (f08cxc).
*/
nag_zunmrq(order, Nag_LeftSide, Nag_ConjTrans, n, 1, p, b, pdb, taub, x, pdx,
&fail);
if (fail.code != NE_NOERROR)
{
printf("Error from nag_zunmrq (f08cxc).\n\n", fail.message);
exit_status = 1;
goto END;
}

/* It remains to minimize ||g2||, g2 = f2 - (T22 T23)*w.
* Putting w = (y2), gives g2 = f2 - T22*y2 - T23*y3
* [y2 stored in d1, first y2rows of d; y3 stored in d2, next n-m rows of d.]
* First form T22*y2 using nag_ztrmv (f16sfc) where y2 is held in d.
*/
y2rows = MIN(m, n) - y1rows;
alpha = nag_complex(1.0,0.0);
nag_ztrmv(order, Nag_Upper, Nag_NoTrans, Nag_NonUnitDiag, y2rows, alpha,
&A(n-p+1, n-p+1), pda, d, 1, &fail);
if (fail.code != NE_NOERROR)
{
printf("Error from nag_ztrmv (f16sfc).\n\n", fail.message);
exit_status = 1;
goto END;
}

/* Then, f2 - T22*y2 (c2 = c2 - d) */
for (i = 0; i < y2rows; ++i)
c[y2rows + i] = nag_complex_subtract(c[y2rows+i], d[i]);
y2rows = m - y1rows;
if (m < n)
{
    y3rows = n - m;
    /* Then g2 = f2 - T22*y2 - T23*y3 (c2 = c2 - T23*d2) */
    alpha = nag_complex(-1.0,0.0);
    nag_zgemv(order, Nag_NoTrans, y2rows, y3rows, alpha, &A(n-p+1, m+1), pda,
    &d[y2rows], 1, beta, &c[y1rows], 1, &fail);
    if (fail.code != NE_NOERROR)
    {
        printf("Error from nag_zgemv (f16sac).\n%s\n", fail.message);
        exit_status = 1;
        goto END;
    }

    /* Compute ||g|| = ||g2|| = norm(f2 - T22*y2 - T23*y3)
     * using nag_zge_norm (f16uac).
     */
    nag_zge_norm(Nag_ColMajor, Nag_FrobeniusNorm, y2rows, 1, &c[y1rows], y2rows,
    &rnorm, &fail);
    if (fail.code != NE_NOERROR)
    {
        printf("Error from nag_zge_norm (f16uac).\n%s\n", fail.message);
        exit_status = 1;
        goto END;
    }

    /* Print least squares solution x */
    printf("Constrained least squares solution\n");
    for (i = 0; i < n; ++i)
        printf(" (%7.4f, %7.4f)%s", x[i].re, x[i].im, i%4 == 3?"\n":"");

    /* Print the square root of the residual sum of squares */
    printf("\nSquare root of the residual sum of squares\n");
    printf("%11.2e\n", rnorm);
}

END:
NAG_FREE(a);
NAG_FREE(b);
NAG_FREE(c);
NAG_FREE(d);
NAG_FREE(taua);
NAG_FREE(taub);
NAG_FREE(x);

return exit_status;
}
10.2 Program Data

nag_zggrqf (f08ztc) Example Program Data

\[
\begin{array}{ccc}
6 & 4 & 2 \\
(0.96, -0.81) & (-0.03, 0.96) & (-0.91, 2.06) & (-0.05, 0.41) \\
(-0.98, 1.98) & (-1.20, 0.19) & (-0.66, 0.42) & (-0.81, 0.56) \\
(0.62, -0.46) & (1.01, 0.02) & (0.63, -0.17) & (-1.11, 0.60) \\
(0.37, 0.38) & (0.19, -0.54) & (-0.98, -0.36) & (0.22, -0.20) \\
(0.83, 0.51) & (0.20, 0.01) & (-0.17, -0.46) & (1.47, 1.59) \\
(1.08, -0.28) & (0.20, -0.12) & (-0.07, 1.23) & (0.26, 0.26) \\
\end{array}
\]

\text{matrix A}

\[
\begin{array}{cccc}
1.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & -1.00 & 0.00 \\
0.00 & 0.00 & 0.00 & -1.00 \\
0.00 & 0.00 & 0.00 & 0.00 \\
\end{array}
\]

\text{matrix B}

\[
\begin{array}{c}
(-2.54, 0.09) \\
(1.65, -2.26) \\
(-2.11, -3.96) \\
(1.82, 3.30) \\
(-6.41, 3.77) \\
(2.07, 0.66) \\
\end{array}
\]

\text{vector c}

\[
\begin{array}{c}
(0.00, 0.00) \\
(0.00, 0.00) \\
\end{array}
\]

\text{vector d}

10.3 Program Results

nag_zggrqf (f08ztc) Example Program Results

Constrained least squares solution

\[
\begin{array}{cccc}
1.0874, -1.9621 & -0.7409, 3.7297 & 1.0874, -1.9621 & -0.7409, 3.7297 \\
\end{array}
\]

Square root of the residual sum of squares

\[1.59e-01\]