

NAG Library Function Document

nag_zhpr (f16sqc)

1 Purpose

nag_zhpr (f16sqc) performs a Hermitian rank-1 update on a complex Hermitian matrix stored in packed form.

2 Specification

```
#include <nag.h>
#include <nagf16.h>

void nag_zhpr (Nag_OrderType order, Nag_UploType uplo, Integer n,
              double alpha, const Complex x[], Integer incx, double beta,
              Complex ap[], NagError *fail)
```

3 Description

nag_zhpr (f16sqc) performs the Hermitian rank-1 update operation

$$A \leftarrow \alpha x x^H + \beta A,$$

where A is an n by n complex Hermitian matrix, stored in packed form, x is an n -element complex vector, while α and β are real scalars.

4 References

Basic Linear Algebra Subprograms Technical (BLAST) Forum (2001) *Basic Linear Algebra Subprograms Technical (BLAST) Forum Standard* University of Tennessee, Knoxville, Tennessee <http://www.netlib.org/blas/blast-forum/blas-report.pdf>

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: **uplo** – Nag_UploType *Input*
On entry: specifies whether the upper or lower triangular part of A is stored.
uplo = Nag_Upper
The upper triangular part of A is stored.
uplo = Nag_Lower
The lower triangular part of A is stored.
Constraint: **uplo** = Nag_Upper or Nag_Lower.
- 3: **n** – Integer *Input*
On entry: n , the order of the matrix A .
Constraint: $n \geq 0$.

- 4: **alpha** – double *Input*
On entry: the scalar α .
- 5: **x**[*dim*] – const Complex *Input*
Note: the dimension, *dim*, of the array **x** must be at least $\max(1, 1 + (\mathbf{n} - 1)|\mathbf{incx}|)$.
On entry: the vector x .
- 6: **incx** – Integer *Input*
On entry: the increment in the subscripts of **x** between successive elements of x .
Constraint: **incx** $\neq 0$.
- 7: **beta** – double *Input*
On entry: the scalar β .
- 8: **ap**[*dim*] – Complex *Input/Output*
Note: the dimension, *dim*, of the array **ap** must be at least $\max(1, \mathbf{n} \times (\mathbf{n} + 1)/2)$.
On entry: the n by n Hermitian matrix A , packed by rows or columns.
The storage of elements A_{ij} depends on the **order** and **uplo** arguments as follows:
if **order** = 'Nag_ColMajor' and **uplo** = 'Nag_Upper',
 A_{ij} is stored in **ap**[($j - 1$) \times $j/2 + i - 1$], for $i \leq j$;
if **order** = 'Nag_ColMajor' and **uplo** = 'Nag_Lower',
 A_{ij} is stored in **ap**[($2n - j$) \times ($j - 1$)/2 + $i - 1$], for $i \geq j$;
if **order** = 'Nag_RowMajor' and **uplo** = 'Nag_Upper',
 A_{ij} is stored in **ap**[($2n - i$) \times ($i - 1$)/2 + $j - 1$], for $i \leq j$;
if **order** = 'Nag_RowMajor' and **uplo** = 'Nag_Lower',
 A_{ij} is stored in **ap**[($i - 1$) \times $i/2 + j - 1$], for $i \geq j$.
On exit: the updated matrix A . The imaginary parts of the diagonal elements are set to zero.
- 9: **fail** – NagError * *Input/Output*
The NAG error argument (see Section 3.6 in the Essential Introduction).

6 Error Indicators and Warnings

NE_BAD_PARAM

On entry, argument $\langle value \rangle$ had an illegal value.

NE_INT

On entry, **incx** = $\langle value \rangle$.

Constraint: **incx** $\neq 0$.

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

Constraint: **n** ≥ 0 .

7 Accuracy

The BLAS standard requires accurate implementations which avoid unnecessary over/underflow (see Section 2.7 of Basic Linear Algebra Subprograms Technical (BLAST) Forum (2001)).

8 Parallelism and Performance

Not applicable.

9 Further Comments

None.

10 Example

Perform rank-1 update of complex Hermitian matrix A , stored in packed storage format, using vector x :

$$A \leftarrow A - xx^H,$$

where A is the 4 by 4 Hermitian matrix given by

$$A = \begin{pmatrix} 4.0 + 0.0i & 7.0 - 4.0i & -0.6 + 2.2i & -4.0 + 3.0i \\ 7.0 + 4.0i & 14.0 + 0.0i & 0.3 + 1.2i & -4.7 - 2.1i \\ -0.6 - 2.2i & 0.3 - 1.2i & 2.04 + 0.0i & -5.9 - 0.1i \\ -4.0 - 3.0i & -4.7 + 2.1i & -5.9 + 0.1i & 6.0 + 0.0i \end{pmatrix},$$

and

$$x = \begin{pmatrix} 2.0 + 1.0i \\ 2.0 + 3.0i \\ 0.2 - 1.0i \\ -1.0 - 2.0i \end{pmatrix}.$$

10.1 Program Text

```

/* nag_zhpr (f16sqc) Example Program.
 *
 * Copyright 2005 Numerical Algorithms Group.
 *
 * Mark 8, 2005.
 */

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

int main(void)
{
    /* Scalars */
    double      alpha, beta;
    Integer     exit_status, i, incx, j, n, ap_len, xlen;

    /* Arrays */
    Complex     *ap = 0, *x = 0;
    char        nag_enum_arg[40];

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

#ifdef NAG_COLUMN_MAJOR
#define A_UPPER(I, J) ap[J*(J-1)/2 + I - 1]
#define A_LOWER(I, J) ap[(2*n-J)*(J-1)/2 + I - 1]
    order = Nag_ColMajor;
#else
#define A_UPPER(I, J) ap[I*(I-1)/2 + J - 1]
#define A_LOWER(I, J) ap[(2*n-I)*(I-1)/2 + J - 1]
    order = Nag_RowMajor;
#endif

```

```

#endif

exit_status = 0;
INIT_FAIL(fail);

printf("nag_zhpr (f16sqc) Example Program Results\n\n");

/* Skip heading in data file */
scanf("%*[\n] ");

/* Read the problem dimension */
scanf("%ld%[\n] ", &n);

/* Read the uplo storage parameter */
scanf("%39s%[\n] ", nag_enum_arg);
/* nag_enum_name_to_value (x04nac).
 * Converts NAG enum member name to value
 */
uplo = (Nag_UploType) nag_enum_name_to_value(nag_enum_arg);

/* Read scalar parameters */
scanf("%lf%lf%[\n] ", &alpha, &beta);
/* Read increment parameter */
scanf("%ld%[\n] ", &incx);

xlen = MAX(1, 1 + (n - 1)*ABS(incx));
ap_len = n * (n + 1)/2;

if (n > 0)
{
/* Allocate memory */
if (!(ap = NAG_ALLOC(ap_len, Complex)) ||
!(x = NAG_ALLOC(xlen, Complex)))
{
printf("Allocation failure\n");
exit_status = -1;
goto END;
}
}
else
{
printf("Invalid n\n");
exit_status = 1;
return exit_status;
}

/* Input matrix A and vector x */
if (uplo == Nag_Upper)
{
for (i = 1; i <= n; ++i)
{
for (j = i; j <= n; ++j)
scanf(" ( %lf , %lf )", &A_UPPER(i, j).re,
&A_UPPER(i, j).im);
scanf("%*[\n] ");
}
}
else
{
for (i = 1; i <= n; ++i)
{
for (j = 1; j <= i; ++j)
scanf(" ( %lf , %lf )", &A_LOWER(i, j).re,
&A_LOWER(i, j).im);
scanf("%*[\n] ");
}
}
for (i = 0; i < xlen; ++i)
scanf(" ( %lf , %lf )%*[\n] ", &x[i].re, &x[i].im);

```

```

/* nag_zhpr (f16sqc).
 * Rank one update of complex Hermitian matrix,
 * packed storage.
 */
nag_zhpr(order, uplo, n, alpha, x, incx, beta, ap, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_zhpr.\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}
/* Print updated matrix A */
/* nag_pack_complx_mat_print_comp (x04ddc).
 * Print complex packed triangular matrix (comprehensive)
 */
fflush(stdout);
nag_pack_complx_mat_print_comp(order, uplo, Nag_NonUnitDiag, n, ap,
                               Nag_BracketForm, "%7.4f",
                               "Updated Matrix A", Nag_IntegerLabels,
                               0, Nag_IntegerLabels, 0, 80, 0, 0,
                               &fail);

if (fail.code != NE_NOERROR)
{
    printf("Error from nag_pack_complx_mat_print_comp (x04ddc).\n%s"
           "\n", fail.message);
    exit_status = 1;
    goto END;
}
END:
    NAG_FREE(ap);
    NAG_FREE(x);

    return exit_status;
}

```

10.2 Program Data

```

nag_zhpr (f16sqc) Example Program Data
  4                               :Value of n
  Nag_Lower                       :Storage of A
-1.0 1.0                          :Values of alpha and beta
  1                               :Value of incx
( 4.0, 0.0)
( 7.0, 4.0) (14.0, 0.0)
(-0.6,-2.2) ( 0.3,-1.2) ( 2.04,0.0)
(-4.0,-3.0) (-4.7, 2.1) (-5.9, 0.1) ( 6.0, 0.0) :End of matrix A
( 2.0, 1.0)
( 2.0, 3.0)
( 0.2,-1.0)
(-1.0,-2.0)                       :End of vector x

```

10.3 Program Results

nag_zhpr (f16sqc) Example Program Results

```

Updated Matrix A
      1                2                3                4
1  (-1.0000, 0.0000)
2  ( 0.0000, 0.0000) ( 1.0000, 0.0000)
3  ( 0.0000, 0.0000) ( 2.9000, 1.4000) ( 1.0000, 0.0000)
4  ( 0.0000, 0.0000) ( 3.3000, 3.1000) (-7.7000, 1.5000) ( 1.0000, 0.0000)

```
