BLAS and LAPACK Subroutines of Real Skew-symmetric Matrix

Shuo Zheng

September 3, 2024

Abstract

In this note we introduce some new subroutines of real skew-symmetric matrix. These subroutines include BLAS2/3 operations, linear solver and eigensolver in LAPACK. All these subroutines are named and implemented as their corresponding real symmetric subroutines.

1 Introduction

A real square matrix A is skew-symmetric (also called anti-symmetric), if $A^T = -A$, where A^T is the transpose of A. Real skew-symmetric matrix is regular and so unitary diagonalizable. All its eigenvalues are conjugate pure imaginary value pairs, or zeros.

Real skew-symmetric matrices are widely used in physics [1, 2], engineering [9], and computer vision [8]. A survey of LAPACK [5] showed the demands of subroutines calculating such kind of matrix.

Now there are two methods to calculate real skew-symmetric matrix. They are

(1) Using general real matrix subroutines, which can't benefit from the skew-symmetric structure to reduce the computational complexity.

(2) Multiplying the matrix by i and then using Hermitian matrix subroutines, which will extend the computational complexity and storage spaces twice or more by introducing complex operation.

Since the structure of real skew-symmetric matrix is similar to real symmetric matrix, it is expected that some subroutines of real skew-symmetric matrix can be implemented, just by refering the algorithm of corresponding symmetric subroutines. What's more, their performance should be comparable with the corresponding symmetric subroutines.

In this note we introduce some subroutines of real skew-symmetric matrix. These subroutines include BLAS2/3 operations, linear solver, eigensolver and corresponding computational/auxiliary subroutines of real skew-symmetric matrix. In the following sections, we will describe their interfaces, data layouts, and algorithms in order. For brevity we omit the numerical type prefix S/D and use * instead.

2 Design principles of subroutines

The real skew-symmetric subroutines are designed to be corresponding to real symmetric subroutines. So there are following principles.

(1) Keep the function of each subroutine being the reflection of corresponding real symmetric subroutine.

(2) Keep the names resembling real symmetric subroutines. The key letter S meaning symmetric is replaced by key letter K meaning skew-symmetric. So the SY/ST in the name are replaced by KY/KT. The only exception is subroutine *LATRDK, which comes from *LATRD in real symmetric subroutines.

(3) Keep the interfaces resembling real symmetric subroutines. The order, data type, and input data layout of parameters have only few changes compared with real symmetric subroutines. But the output data layout of some real skew-symmetric subroutines are changed due to the difference of algorithms.

(4) Keep the change of algorithms being the least compared with the corresponding real symmetric subroutines. For most of real skew-symmetric subroutines, the only changes are sign of some inner operations. However, some different algorithms still can't be bypassed due to the different structure of matrices and we will describe them in details later. Generally, the storage spaces and performances of real symmetric and skew-symmetric subroutines are in similar level.

3 BLAS2/3 operations

3.1 List of subroutines

Subroutine name	Describe	Corresponding symmetric subroutine	Belonging
KYMV	real skew-symmetric matrix-vector multiply	SYMV	BLAS2
KYR2	real skew-symmetric rank-2 update	SYR2	BLAS2
KYMM	real skew-symmetric matrix-matrix multiply	SYMM	BLAS3
KYR2K	real skew-symmetric rank-2k update	SYR2K	BLAS3

3.2 Subroutine prototypes, functions, features and algotithms

*subroutine *KYMV(uplo, n, alpha, A, lda, x, incx, beta, y, incy)*

Function

The BLAS2 subroutine does real skew-symmetric matrix-vector multiplying $y = \alpha Ax + \beta y$.

Differences from symmetric case

The diagonal elements of A are implied to be 0 and not accessed.

Algorithm

Be the same as symmetric case, except for the sign of some inner operations.

*subroutine *KYR2(uplo, n, alpha, x, incx, y, incy, A, lda)*

Function

The BLAS2 subroutine does real skew-symmetric rank-2 update $A = -\alpha xy^T + \alpha yx^T + A$.

Differences from symmetric case

The diagonal elements of A are implied to be 0 and not accessed.

Algorithm

Be the same as symmetric case, except for the sign of some inner operations.

*subroutine *KYMM(side, uplo, m, n, alpha, A, lda, B, ldb, beta, C, ldc)*

Function

The BLAS3 subroutine does real skew-symmetric matrix-matrix multiplying $C = \alpha AB + \beta C$ or $C = \alpha BA + \beta C$.

Differences from symmetric case

The diagonal elements of A are implied to be 0 and not accessed.

Algorithm

Be the same as symmetric case, except for the sign of some inner operations.

*subroutine *KYR2K(uplo, trans, n, k, alpha, A, lda, B, ldb, beta, C, ldc)*

Function

The BLAS3 subroutine does real skew-symmetric rank-2k update $C = -\alpha AB^T + \alpha BA^T + \beta C$.

Differences from symmetric case

The diagonal elements of C are implied to be 0 and not accessed.

Algorithm

Be the same as symmetric case, except for the sign of some inner operations.

4 linear solver subroutines in LAPACK

4.1 List of subroutines

Subroutine name	Describe	Corresponding symmetric subroutine
KYSV	solving real skew-symmetric linear system	SYSV
KYTRF	real skew-symmetric factorization into LDL^T/UDU^T form	SYTRF
LAKYF	real skew-symmetric factorization into LDL^T/UDU^T form, blocked	LASYF
KYTF2	real skew-symmetric factorization into LDL^T/UDU^T form, unblocked	SYTF2
KYTRS2	solving factorized real skew-symmetric linear system, blocked	SYTRS2
KYTRS	solving factorized real skew-symmetric linear system, unblocked	SYTRS
KYTRI2	inverting factorized real skew-symmetric matrix	SYTRI2
KYTRI2X	inverting factorized real skew-symmetric matrix, blocked	SYTRI2X
KYTRI	inverting factorized real skew-symmetric matrix, unblocked	SYTRI
KYCONV	permutating the interchanges of factorized real skew-symmetric matrix	SYCONV

4.2 Subroutine prototypes, functions, features and algotithms

*subroutine *KYSV(uplo, n, nrhs, A, lda, ipiv, B, ldb, work, lwork, info)*

Function

The subroutine computes the solution of a real linear system $AX = B$, where A is a real skew-symmetric matrix.

Differences from symmetric case

(1) The diagonal elements of A are implied to be 0 and not accessed.

(2) The interchanges of pivoting stored in ipiv have a different layout. Refer to the *KYTRF.

Algorithm

The subroutine calls *KYTRF to do real skew-symmetric LDL^T/UDU^T factorization and then solves the factorized triangular linear system by calling *KYTRS2 and *KYTRS.

*subroutine *KYTRF(uplo, n, A, lda, ipiv, work, lwork, info)*

Function

The subroutine computes the factorization of a real skew-symmetric matrix A using the Bunch partial pivoting method [3]. The form of the factorization is LDL^T/UDU^T. It is a shell and calls *LAKYF for the main body of matrix, then calls *KYTF2 for the tail of matrix.

Differences from symmetric case

(1) The diagonal elements of A are implied to be 0 and not accessed.

(2) The interchanges of pivoting stored in ipiv have a different layout.

In this subroutine, due to the adoption of Bunch partial pivoting method, the step of factorization is always 2 (until the program end or stop with singularity), instead of 1 or 2 in symmetric case. So in ipiv, the interchanges are recorded at every other elements and the rest of elements are not referenced. That means the elements of ipiv are always organized in pair. In case of uplo = 'L', The interchanges is applied from column 1 to N, and the former element in a pair is effective. In case of uplo = 'U', The interchanges is applied from column N to 1, and the latter element in a pair is effective.

The sign of recording of interchange decides which kind of interchange is done. Positive integer P at the K-th element means the interchange is between row and column P and K+1 (if uplo = 'L'), or K-1 (if uplo = 'U'). Negative integer P at the K-th element means the interchange is between row and column K and K+1, then -P and K+1 (if uplo = 'L'), or K and K-1, then -P and K-1 (if uplo = 'U'). Zero element means no interchange.

For example, for a 6-by-6 dimension matrix, if:

a. uplo = 'L', and ipiv = (3, *, -6, *, 0, *), then the interchanges are applied by the following 3 steps.

Step 1, paraphrase (3, *), interchange row and column 2 and 3.

Step 2, paraphrase (-6, *), interchange row and column 3 and 4, then 4 and 6.

Step 3, paraphrase (0, *), no interchange.

b. uplo = 'U', and ipiv = (*, 0, *, -1, *, 3), then the interchanges are applied by the following 3 steps.

Step 1, paraphrase (*, 3), interchange row and column 5 and 3.

Step 2, paraphrase (*, -1), interchange row and column 4 and 3, then 3 and 1.

Step 3, paraphrase (*, 0), no interchange.

* means unreferenced elements here. If n is odd, then the last (if uplo = 'L') or first (if uplo = 'U') element in ipiv is unreferenced.

Algorithm

(1) For the case of uplo = 'L', we partition the n-by-n real skew-symmetric matrix A into the following pattern

$$A = \begin{bmatrix} S & -C^T \\ C & B \end{bmatrix}$$

where S is a 2-by-2 real skew-symmetric matrix, B is a n-2-by-n-2 real skew-symmetric matrix and C is a n-2-by-2 real matrix.

For the factorization, we should make the following pivoting

a. Find the absolute maximum element of C and assume its row index is k.

b. If the maximum is less than or equal to $|S_{21}|$, no pivoting is needed.

c. If the maximum is in the first column of C, and is greater than $|S_{21}|$, interchange the row and column 2 and k. This can permute the maximum to S_{21}. k is recorded in the former element of ipiv.

d. If the maximum is in the second column of C, and is greater than $|S_{21}|$, interchange the row and column 1 and 2, then row and column 2 and k. This can permute the maximum to S_{21}. -k is recorded in the former element of ipiv.

e. If both the maximum and S_{21} are 0, the matrix is singular and the process should be terminated.

Then factorize the matrix into the following pattern

$$\begin{bmatrix} S & -C^T \\ C & B \end{bmatrix} = \begin{bmatrix} I & 0 \\ CS^{-1} & I \end{bmatrix} \begin{bmatrix} S & 0 \\ 0 & B + CS^{-1}C^T \end{bmatrix} \begin{bmatrix} I & -S^{-1}C^T \\ 0 & I \end{bmatrix}$$

We store S_{21} and CS^{-1} in A below the diagonal. There is no need to refer to the leading upper triangular part of A.

Since $B + CS^{-1}C^T$ is also a real skew-symmetric matrix, we continue to do the factorization for it until we get to the tail, or no pivoting can be done.

(2) For the case of uplo = 'U', we partition the n-by-n real skew-symmetric matrix A into the following pattern

$$A = \begin{bmatrix} B & C \\ -C^T & S \end{bmatrix}$$

where S is a 2-by-2 real skew-symmetric matrix, B is a n-2-by-n-2 real skew-symmetric matrix and C is a n-2-by-2 real matrix.

For the factorization, we should make the following pivoting

a. Find the absolute maximum element of C and assume its row index is k.

b. If the maximum is less than or equal to $|S_{12}|$, no pivoting is needed.

c. If the maximum is in the second column of C, and is greater than $|S_{12}|$, interchange the row and column n-1 and k. This can permute the maximum to S_{12}. k is recorded in the latter element of ipiv.

d. If the maximum is in the first column of C, and is greater than $|S_{12}|$, interchange the row and column n and n-1, then row and column n-1 and k. This can permute the maximum to S_{12}. -k is recorded in the latter element of ipiv.

e. If both the maximum and S_{12} are 0, the matrix is singular and the process should be terminated.

Then factorize the matrix into the following pattern

$$\begin{bmatrix} B & C \\ -C^T & S \end{bmatrix} = \begin{bmatrix} I & CS^{-1} \\ 0 & I \end{bmatrix} \begin{bmatrix} B + CS^{-1}C^T & 0 \\ 0 & S \end{bmatrix} \begin{bmatrix} I & 0 \\ -S^{-1}C^T & I \end{bmatrix}$$

We store the S_{12} and CS^{-1} in A above the diagonal. There is no need to refer to the leading lower triangular part of A.

Since $B + CS^{-1}C^T$ is also a real skew-symmetric matrix, we continue to do the factorization for it until we get to the head, or no pivoting can be done.

*subroutine *LAKYF(uplo, n, nb, kb, A, lda, ipiv, w, ldw, info)*

Function

The subroutine computes factorization of a real skew-symmetric matrix A using the Bunch partial pivoting method. It is blocked version called by *KYTRF.

Differences from symmetric case

(1) The diagonal elements of A are implied to be 0 and not accessed.

(2) The interchanges of pivoting stored in ipiv have a different layout. Refer to the *KYTRF.

Algorithm

Refer to the *KYTRF.

*subroutine *KYTF2(uplo, n, A, lda, ipiv, info)*

Function

The subroutine computes factorization of a real skew-symmetric matrix A using the Bunch partial pivoting method. It is unblocked version called by *KYTRF.

Differences from symmetric case

(1) The diagonal elements of A are implied to be 0 and not accessed.

(2) The interchanges of pivoting stored in ipiv have a different layout. Refer to the *KYTRF.

Algorithm

Refer to the *KYTRF.

*subroutine *KYTRS2(uplo, n, nrhs, A, lda, ipiv, B, ldb, work, info)*

Function

The subroutine solves triangular real skew-symmetric linear system factorized by *KYTRF. It is blocked version and called by *KYSV if the size of work is equal to or greater than N.

Differences from symmetric case

(1) The diagonal elements of A are implied to be 0 and not accessed.

(2) The interchanges of pivoting stored in ipiv have a different layout. Refer to the *KYTRF.

Algorithm

Be the same as 2-by-2 blocked case of symmetric subroutine except that the interchanges stored in ipiv are in skew-symmetric layout.

*subroutine *KYTRS(uplo, n, nrhs, A, lda, ipiv, B, ldb, info)*

Function

The subroutine solves triangular real skew-symmetric linear system factorized by *KYTRF. It is unblocked version and called by *KYSV if the size of work is less than N.

Differences from symmetric case

(1) The diagonal elements of A are implied to be 0 and not accessed.

(2) The interchanges of pivoting stored in ipiv have a different layout. Refer to the *KYTRF.

Algorithm

Be the same as 2-by-2 blocked case of symmetric subroutine except that the interchanges stored in ipiv are in skew-symmetric layout.

*subroutine *KYTRI2(uplo, n, A, lda, ipiv, work, lwork, info)*

Function

The subroutine inverts triangular real skew-symmetric matrix factorized by *KYTRF. It is a shell and calls *KYTRI2X or *KYTRI depending on the block size. If the block size is less than N, *KYTRI2X is called, or *KYTRI is called.

Differences from symmetric case

(1) The diagonal elements of A are implied to be 0 and not accessed.

(2) The interchanges of pivoting stored in ipiv have a different layout. Refer to the *KYTRF.

Algorithm

Be the same as 2-by-2 blocked case of symmetric subroutine except that the interchanges stored in ipiv are in skew-symmetric layout.

*subroutine *KYTRI2X(uplo, n, A, lda, ipiv, work, nb, info)*

Function

The subroutine inverts triangular real skew-symmetric matrix factorized by *KYTRF. It is blocked version.

Differences from symmetric case

(1) The diagonal elements of A are implied to be 0 and not accessed.

(2) The interchanges of pivoting stored in ipiv have a different layout. Refer to the *KYTRF.

Algorithm

Be the same as 2-by-2 blocked case of symmetric subroutine except that the interchanges stored in ipiv are in skew-symmetric layout.

*subroutine *KYTRI(uplo, n, A, lda, ipiv, work, info)*

Function

The subroutine inverts triangular real skew-symmetric matrix factorized by *KYTRF. It is unblocked version.

Differences from symmetric case

(1) The diagonal elements of A are implied to be 0 and not accessed.
(2) The interchanges of pivoting stored in ipiv have a different layout. Refer to the *KYTRF.

Algorithm
Be the same as 2-by-2 blocked case of symmetric subroutine except that the interchanges stored in ipiv are in skew-symmetric layout.

*subroutine *KYCONV(uplo, way, n, A, lda, ipiv, e, info)*

Function
The subroutine permutes interchanges of real skew-symmetric matrix factorized by *KYTRF.

Differences from symmetric case
(1) The diagonal elements of A are implied to be 0 and not accessed.
(2) The interchanges of pivoting stored in ipiv have a different layout. Refer to the *KYTRF.

Algorithm
Be the same as 2-by-2 blocked case of symmetric subroutine except that the interchanges stored in ipiv are in skew-symmetric layout.

5 eigensolver subroutines in LAPACK

5.1 List of subroutines

Subroutine name	Describe	Corresponding symmetric subroutine
KYEV	solving eigenproblem of real skew-symmetric matrix	SYEV
KYTRD	orthogonally reducing real skew-symmetric matrix to tridiagonal form	SYTRD
LATRDK	orthogonally reducing real skew-symmetric matrix to tridiagonal form, blocked	LATRD
KYTD2	orthogonally reducing real skew-symmetric matrix to tridiagonal form, unblocked	SYTD2
KTEV	solving eigenproblem of real skew-symmetric tridiagonal matrix	STEV
KTEQR	solving eigenproblem of real skew-symmetric tridiagonal matrix, core QR method	STEQR
KYGV	solving generalized eigenproblem of real skew-symmetric matrix	SYGV
KYGST	reducing real skew-symmetric generalized eigenproblem to standard form, blocked	SYGST
KYGS2	reducing real skew-symmetric generalized eigenproblem to standard form, unblocked	SYGS2
KYSWAPR	applying elementary permutation on real skew-symmetric matrix	SYSWAPR

5.2 Subroutine prototypes, functions, features and algotithms

*subroutine *KYEV(jobz, uplo, n, A, lda, w, work, lwork, info)*

Function

The subroutine solves eigenproblem of real skew-symmetric matrix matrix A.

Differences from symmetric case

(1) The diagonal elements of A are implied to be 0 and not accessed.

(2) w has a different layout compared with *SYEV.

It should be avoided to calculate and store complex values in real subroutines. So the layout of eigenvalues in w is different with subroutine *SYEV in symmetric case, where all the eigenvalues are stored in w. In this subroutine, the original matrix is reduced to real skew-symmetric diagonal block form. The size of blocks are 1 (indicate eigenvalue 0) or 2 (indicate a pair of conjugate pure imaginary eigenvalues). The subdiagonal elements of the reduced diagonal block matrix is stored at the front of w. The last element in w is always 0. Non-zero elements in w are always positive and sorted in decending order.

For example, A real skew-symmetric matrix will be reduced to the following diagonal block form

$$\begin{bmatrix} 0 & -5 & & & \\ 5 & 0 & & & \\ & & 0 & -2 & \\ & & 2 & 0 & \\ & & & & 0 \end{bmatrix}$$

and elements in w is (5, 0, 2, 0, 0). This shows the eigenvalues of original matrix is $5i$, $-5i$, $2i$, $-2i$, 0.

If jobz = 'V', A is overwritten by the orthogonal transformation matrix which transforms the original matrix to diagonal block form.

Algorithm

Be the same as symmetric case, except for calling *KYTRD instead of *SYTRD and *KTEQR instead of *STEQR. Whether the orthogonal transformation matrix is needed or not, we call *KTEQR. *STERF has no corresponding skew-symmetric subroutine because of the difference of QR algorithm.

*subroutine *KYTRD(uplo, n, A, lda, e, tau, work, lwork, info)*

Function

The subroutine reduces matrix A to real skew-symmetric tridiagonal matrix by orthogonal transformation. It is a shell and calls *LATRDK and *KYR2K for the main body of matrix, then calls *KYTD2 for the tail of matrix.

Differences from symmetric case

(1) The diagonal elements of A are implied to be 0 and not accessed.

(2) The diagonal elements array d as output is deleted, for its implicit values 0.

Algorithm

The algorithm is similar as symmetric case, but it has some simplifications due to the property of skew-symmetric structure. Both unblocked and blocked case are described here.

In general, by applying a series of Householder transformation $H_i = I - \tau_i v_i v_i^T$, we reduce a skew-symmetric matrix A to skew-symmetric tridiagonal form T.

$T = H_n \dots H_1 A H_1 \dots H_n$

For the initial transformation, we have

$$\begin{aligned} A_1 &= H_1 A H_1 \\ &= (I - \tau_1 v_1 v_1^T) A (I - \tau_1 v_1 v_1^T) \\ &= A - \tau_1 v_1 v_1^T A - \tau_1 A v_1 v_1^T + \tau_1^2 v_1 v_1^T A v_1 v_1^T \end{aligned}$$

The term $\tau_1^2 v_1 v_1^T A v_1 v_1^T = 0$ for the property of skew-symmetric structure. By denoting $x_i = \tau_i A v_i$, we have $A_1 = A + v_1 x_1^T - x_1 v_1^T$, which is a *KYR2 operation. In the unblock subroutine *KYTD2, we apply this formula.

For the blocked subroutine, we represent the formula into block form. To do this, we exert induction.

In the case $i = 1$, let $V_1 = (v_1)$ and $X_1 = (x_1)$, we have $A_1 = A + VX^T - XV^T$.

Suppose that when $i = k$, A_k can be represented as $A_k = H_k \dots H_1 A H_1 \dots H_k = A + V_k X_k^T - X_k V_k^T$, where $V_k = (v_1, \dots, v_k)$ and $X_k = (x_1, \dots, x_k)$.

To proof the case $i = k + 1$, we need to find v_{k+1} and x_{k+1}, and show A_{k+1} still has the form $A_{k+1} = H_{k+1} \dots H_1 A H_1 \dots H_{k+1} = A + V_{k+1} X_{k+1}^T - X_{k+1} V_{k+1}^T$. There are 3 steps.

(1) v_{k+1} can be evaluated by calling *DLARFG with the input u_{k+1}, where

$$u_{k+1} = A(:, k+1) + V_k X_k^T(:, k+1) - X_k V_k^T(:, k+1)$$

and $A(:, k+1)$ is the column k + 1 of A.

(2) x_{k+1} can be evaluated by

$$\begin{aligned} x_{k+1} &= \tau_{k+1} A_k v_{k+1} \\ &= \tau_{k+1}(A + V_k X_k^T - X_k V_k^T) v_{k+1} \\ &= \tau_{k+1}(A v_{k+1} + V_k X_k^T v_{k+1} - X_k V_k^T v_{k+1}) \end{aligned}$$

(3) Apply the Householder transformation to A_k, we have

$$\begin{aligned} A_{k+1} &= H_{k+1} A_k H_{k+1} \\ &= (I - \tau_{k+1} v_{k+1} v_{k+1}^T) A_k (I - \tau_{k+1} v_{k+1} v_{k+1}^T) \\ &= A_k - \tau_{k+1} v_{k+1} v_{k+1}^T A_k - \tau_{k+1} A_k v_{k+1} v_{k+1}^T \\ &= A + V_k X_k^T - X_k V_k^T + v_{k+1} x_{k+1}^T - x_{k+1} v_{k+1}^T \\ &= A + V_{k+1} X_{k+1}^T - X_{k+1} V_{k+1}^T \end{aligned}$$

In blocked case, we construct V and X in the leading block with *LATRDK, and update the rest of matrix by *KYR2K.

Refer [6] for more details.

*subroutine *LATRDK(uplo, n, nb, A, lda, e, tau, w, ldw)*

Function

The subroutine reduces leading block of matrix A to real skew-symmetric tridiagonal form by orthogonal transformation. Then its caller *KYTRD can update the rest of matrix by calling BLAS3 subroutines *KYR2K.

Differences from symmetric case

(1) The diagonal elements of A are implied to be 0 and not accessed.

(2) The diagonal elements array d as output is deleted, for its implicit values 0.

Algorithm

Refer to the *KYTRD.

*subroutine *KYTD2(uplo, n, A, lda, e, tau, info)*

Function

The subroutine reduces matrix A to real skew-symmetric tridiagonal form by orthogonal transformation. It is unblocked version called by *KYTRD.

Differences from symmetric case

(1) The diagonal elements of A are implied to be 0 and not accessed.

(2) The diagonal elements array d as output is deleted, for its implicit values 0.

Algorithm

Refer to the *KYTRD.

*subroutine *KTEV(jobz, n, d, e, z, ldz, work, info)*

Function

The subroutine solves eigenproblem of real skew-symmetric tridiagonal matrix.

Differences from symmetric case

The subdiagonal elements of original matrix as input is stored in e and the subdiagonal elements of reduced diagonal block matrix as output is stored in d with the same layout as *KYEV. d is not referenced as input, for its implicit values 0.

If jobz = 'V', ldz stores the orthogonal transformation matrix which transforms the original tridiagonal matrix to diagonal block form.

Algorithm

Be the same as symmetric case, except for calling *KTEQR instead of *STEQR.

*subroutine *KTEQR(compz, n, e, z, ldz, work, info)*

The subroutine solves eigenproblem of real skew-symmetric tridiagonal matrix. It's the core double shift QR/QL subroutine, called by *KYEV and *KTEV.

Differences from symmetric case

The diagonal elements array d is deleted, for its implicit values 0. The subdiagonal elements of original matrix is stored in e and the subdiagonal elements of reduced diagonal block matrix will overwrite e with the same layout as *KYEV, except for the last element, for the dimension of e being N-1.

Algorithm

We adopt standard Francis double shift QR/QL iteration to reduce the real skew-symmetric tridiagonal matrix to 2-by-2 diagonal block form. What's more, for the property of skew-symmetric structure, the Householder reflection can be simplified. Denote the real skew-symmetric tridiagonal matrix T, and suppose none of its subdiagonal elements is zero (or we can split it into submatrices). We make a shift at the initial step, and then chase and annihilate the nonzero bulgy elements under the subdiagonal.

The iteration can be done by QR and QL method, corresponding to the value of first and last elements of subdiagonal. For brevity, we show the QR iteration only. One loop of iteration has 2 steps.

(1) Adopt the shift and do the initial step. Apply the following transformation to T.

$$T_1 = \begin{bmatrix} A_0 & 0 \\ 0 & I \end{bmatrix} T \begin{bmatrix} A_0 & 0 \\ 0 & I \end{bmatrix}$$

where A_0 is a 3-by-3 symmetric matrix representing the Householder reflection

$$A_0 = \begin{bmatrix} -\frac{p_0}{s_0} & 0 & -\frac{r_0}{s_0} \\ 0 & 1 & 0 \\ -\frac{r_0}{s_0} & 0 & \frac{p_0}{s_0} \end{bmatrix}$$

Here β is the shift, $p_0 = -t_1^2 + \beta$, $r_0 = t_1 t_2$, $s_0 = sign(p_0)\sqrt{p_0^2 + r_0^2}$ and t_k is the k-th element of subdiagonal. Due to the skew-symmetric structure, different from a full 3-by-3 Householder reflection matrix, A_0 has 4 zeros, which make it degenerating to a combination of a plane rotation and an axis reflection.

Usually the Francis shift $\beta = t_{n-1}^2$ perform well in convergence. But there are some matrices with special structure that may lead to invariant iteration. For example, the following two matrices will fail to converge under single precision subroutine

$$T = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \text{ or } \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & -10^{-4} & 0 \\ 0 & 10^{-4} & 0 & -1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

This is because in both cases the orthogonal matrix in QR decomposition of $T_s = T^2 + t_{n-1}^2 I$ is close to identity matrix under limited precision and make the iteration invariant.

To resolve this problem, we adopt an exceptional shift every 10 steps of iteration. Different with the exceptional shift in [7], we use a more straightforward shift to force the mass scattering in T_s under certain precision.

Suppose that

$$T = \begin{bmatrix} 0 & -a & 0 \\ a & 0 & -b \\ 0 & b & 0 \end{bmatrix} \text{ or } \begin{bmatrix} 0 & -a & 0 & 0 \\ a & 0 & -b & 0 \\ 0 & b & 0 & -c \\ 0 & 0 & c & 0 \end{bmatrix}$$

and accordingly

$$T_s = \begin{bmatrix} -a^2+b^2 & 0 & ab \\ 0 & -a^2 & 0 \\ ab & 0 & 0 \end{bmatrix} \text{ or } \begin{bmatrix} -a^2+c^2 & 0 & ab & 0 \\ 0 & -a^2-b^2+c^2 & 0 & bc \\ ab & 0 & -b^2 & 0 \\ 0 & bc & 0 & 0 \end{bmatrix}$$

In former case, when $|a|$ is close to $|b|$, T_s is close to a permuted identity matrix. The same problem occur for the latter case when $|a|$ is close to $|c|$, and $|b|$ is much smaller than them.

To keep T_s being far away from such invariant case, we try to adjust the strength of shift to avoid the first column of T_s being close to axis. Denoting $T_{s,adj} = T^2 + (1-k)t_{n-1}^2 I$ where k is a coefficient between 0 and 1, the first column of it is $[-a^2+(1-k)b^2, 0, ab]^T$ or $[-a^2+(1-k)c^2, 0, ab, 0]^T$. In these two cases, balance the vector by $-a^2+(1-k)b^2 = \pm ab$ or $-a^2+(1-k)c^2 = \pm ab$, we get $k = (-a^2+b^2 \mp ab)/b^2$ or $k = (-a^2+c^2 \mp ab)/c^2$ accordingly. This means that in above two invarient cases, $k = |b/a|$ will perform well under the asumption that $|a| \approx |b|$ or $|a| \approx |c|, |b| \ll |a|$.

So the shift is adjusted every 10 steps of iteration by multiplying $1-k$, where $k = |t_{n-2}/t_{n-1}|$ in QR method. To avoid the case of $|t_{n-2}| \gg |t_{n-1}|$, we limit $k \leq 1$.

(2) After the step 1, the real skew-symmetric matrix has a bulge. The leading 4-by-4 blocks has the following structure

$$T_1 = \begin{bmatrix} 0 & & & \\ p_1 & 0 & & \\ 0 & * & 0 & \\ r_1 & 0 & * & 0 \end{bmatrix}$$

Although * may represent some non-zero values, we only use p_1 and r_1 to form the next transformation, which is a Householder reflection to annihilate the bulge r_1. The transformation has the very similar structure.

$$T_2 = \begin{bmatrix} 1 & & \\ & A_1 & \\ & & I \end{bmatrix} T_1 \begin{bmatrix} 1 & & \\ & A_1 & \\ & & I \end{bmatrix}$$

where A_1 is a 3-by-3 symmetric matrix

$$A_1 = \begin{bmatrix} -\frac{p_1}{s_1} & 0 & -\frac{r_1}{s_1} \\ 0 & 1 & 0 \\ -\frac{r_1}{s_1} & 0 & \frac{p_1}{s_1} \end{bmatrix}$$

Here $s_1 = sign(p_1)\sqrt{p_1^2 + r_1^2}$ and the matrix also has 4 zeros. After transformation, T_2 also has a bulge with the same structure at next lower-right position. We use p_2 and r_2 in T_2 to form next Householder reflection.

Continue to chase and annihilate the bulge and get to the end of T, we complete a loop of iteration. Near the end, the rest of elements of T may not be enough to form matrix larger than 3-by-3, so the transformation matrix should correspondingly degenerate.

The iteration is terminated if no adjacent elements at subdiagonal are both nonzero, which means T has been decomposed into 2-by-2 skew-symmetric blocks, and zeros. To define the zero velue t_k numerically, we set the following criterion

$$t_k < \epsilon(t_{k-1}t_{k+1})^{\frac{1}{2}}$$

where ϵ is a small enough value defined in LAPACK.

Refer section 4.4.8 of [4] or [7] for more details.

*subroutine *KYGV(itype, jobz, uplo, n, A, lda, B, ldb, w, work, lwork, info)*

Function

The subroutine solves generalized eigenproblem of matrix A and B, where A is real skew-symmetric matrix and B is positive definite matrix.

Differences from symmetric case

(1) The diagonal elements of A are implied to be 0 and not accessed.

(2) The subdiagonal elements of reduced diagonal block matrix is stored in W with the same layout as *KYEV.

Algorithm

Be the same as symmetric case, except for calling *KYEV instead of *SYEV and *KYGST instead of *SYGST.

*subroutine *KYGST(itype, uplo, n, A, lda, B, ldb, info)*

Function

The subroutine reduces real skew-symmetric-definite generalized eigenproblem to standard form. This subroutine is called by *KYGV.

Differences from symmetric case

The diagonal elements of A are implied to be 0 and not accessed.

Algorithm

Be the same as symmetric case, except for the sign of some inner operations.

*subroutine *KYGS2(itype, uplo, n, A, lda, B, ldb, info)*

Function

The subroutine reduces real skew-symmetric-definite generalized eigenproblem to standard form. This subroutine is called by *KYGST to process the tail part using unblock code.

Differences from symmetric case
The diagonal elements of A are implied to be 0 and not accessed.

Algorithm
Be the same as symmetric case, except for the sign of some inner operations.

*subroutine *KYSWAPR(uplo, n, A, lda, i1, i2)*

Function
The subroutine applies elementary permutation on the rows and columns of a real skew-symmetric matrix.

Differences from symmetric case
The diagonal elements of A are implied to be 0 and not accessed.

Algorithm
Be the same as symmetric case.

6 Matrix norm subroutines in LAPACK

6.1 List of subroutines

Subroutine name	Describe	Corresponding symmetric subroutine
LANKY	norm of real skew-symmetric matrix	LANSY
LANKT	norm of real skew-symmetric tridiagonal matrix	LANST

6.2 Subroutine prototypes, functions, features and algotithms

*subroutine *LANKY(norm, uplo, n, A, lda, work)*

Function
The subroutine computes the norm of a real skew-symmetric matrix.

Differences from symmetric case
The diagonal elements of A are implied to be 0 and not accessed.

Algorithm
Be the same as symmetric case.

*subroutine *LANKT(norm, n, e)*

Function

The subroutine computes the norm of a real skew-symmetric tridiagonal matrix.

Differences from symmetric case

The diagonal elements array d as input is deleted, for its implicit values 0.

Algorithm

Be the same as symmetric case.

7 Performance

We test the performance of two subroutines *KYTRF and *KTEQR, which have major algorithm modifications between symmetric and skew-symmetric cases. The dimention of matrix in testcases are 1000, 2000 and 4000. The performance is evaluated with the average elapsed time (in millisecond) of running 5 times. We use gcc and gfortran with CFLAGS = -O3 and FFLAGS = -O2 -frecursive. The test run with single thread on Intel Xeon E5-1660 v3 at 3.0 GHz.

Dimention	SKYTRF	SSYTRF	SKTEQR	SSTEQR
1000	106.4	99.8	484.4	928.2
2000	794.0	793.6	2606.6	6462.6
4000	6247.0	6075.0	18031.4	47478.0

References

[1] Arvind, B Dutta, N Mukunda and R Simon. “The real symplectic groups in quantum mechanics and optics”. *Pramana* 45.6 (1995), pp. 471–497.

[2] M Bajdich and L Mitas. “Electronic Structure Quantum Monte Carlo”. *Acta Physica Slovaca* 59.2 (2009), pp. 81–168.

[3] J Bunch. “A Note on the Stable Decomposition of Skew-Symmetric Matrices”. *Mathematics of Computation* 38.158 (1982), pp. 475–479.

[4] James W. Demmel. *Applied Numerical Linear Algebra*. Society for Industrial and Applied Mathematics, 1997. URL: `https://epubs.siam.org/doi/abs/10.1137/1.9781611971446`.

[5] Jack Dongarra et al. *2016 Dense Linear Algebra Software Packages Survey*. Tech. rep. 290. LAPACK Working Note, Sept. 2016. URL: `http://www.netlib.org/lapack/lawnspdf/lawn290.pdf`.

[6] Jack J. Dongarra, Sven Hammarling, and Danny C. Sorensen. *Block Reduction of Matrices to Condensed Forms for Eigenvalue Computations*. Tech. rep. 2. LAPACK Working Note, Sept. 1987. URL: `http://www.netlib.org/lapack/lawnspdf/lawn02.pdf`.

[7] R. S. Martin, G. Peters, and J. H. Wilkinson. “The QR Algorithm for Real Hessenberg Matrices”. In: *Handbook for Automatic Computation: Volume II: Linear Algebra*. Ed. by F. L. Bauer et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 1971, pp. 359–371. ISBN: 978-3-642-86940-2. DOI: `10.1007/978-3-642-86940-2_25`. URL: `https://doi.org/10.1007/978-3-642-86940-2_25`.

[8] K Yousif, A Bab-Hadiashar, and R Hoseinnezhad. “An Overview to Visual Odometry and Visual SLAM: Applications to Mobile Robotics”. *Intelligent Industrial Systems* 1.4 (2015), pp. 289–311.

[9] W Zhong, J Lin, and J Zhu. “Computation of gyroscopic systems and symplectic eigensolutions of skew-symmetric matrices”. *Computers & Structures* 52.5 (1994), pp. 999–1009.