SkewLinearAlgebra.jl: Technical report

Simon Mataigne

April 2023

Minimizing-geodesic computation algorithms on Stiefel make an intensive use of exponentials of skew-symmetric matrices. This is an example where we are interested in the fast computation of the matrix exponential. In the case of *symmetric* matrices, efficient routines are available in many languages. This is the case of `Numpy` [9], `MATLAB` [10], `Julia` [2], `C/C++` and `FORTRAN`. This is inherited from the `C`/`FORTRAN` libraries `BLAS` [4] and `LAPACK` [1]. These libraries are imported in the previously cited languages/packages to perform efficient numerical linear algebra operations. However, `LAPACK` provides no specialized routines for *skew-symmetric* matrices. In consequence, the previous languages/packages provide no specialized methods for skew-symmetric matrices either. `SkewLinearAlgebra.jl` is a proposal to fill this unattended part of numerical linear algebra for eigenproblems. In 1976, [15] proposed a `FORTRAN` package with the same aims. This package was a bit forgotten due to the decline of `FORTRAN` and not updated since to the best of my knowledge. Besides, `SkewLinearAlgebra.jl` is not limited to real matrices, proposes a faster *blocked tridiagonalization* and many more options.
I developed the `SkewLinearAlgebra.jl` package at the Massachusetts Institute of Technology (MIT) under the supervision of Prof. Steven G. Johnson. It can be found at the address `https://github.com/JuliaLinearAlgebra/SkewLinearAlgebra.jl`.

1 Mimicking the symmetric problem

There are many ways to compute the matrix exponential. The paper [11] proposes at least 19 of them with various computational performances and numerical stability properties. In state-of-the-art linear algebra libraries such as `LinearAlgebra.jl` [2], the matrix exponential of *symmetric* matrices is implemented as follows. Let $A \in \mathrm{Sym}(n)$. Compute the eigendecomposition $A = V\Lambda V^T$, then return $A = V\exp(\Lambda)V^T$. We see that the symmetric exponential problem essentially reduces to the efficient computation of the eigendecomposition of A. The `LAPACK` routine to obtain the eigendecomposition is `syevr` ("symmetric eigenvalue reduction"). The first step of `syevr` is to call `sytrd`, the routine performing a tridiagonal reduction $A = QTQ^T$ where $Q \in \mathrm{O}(n)$ and T is symmetric tridiagonal. Once the tridiagonalization is performed, various very efficient algorithms exist to obtain the eigendecomposition of T. `syevr` implements a divide and conquer algorithm [5]. The praised QR algorithm [7] is another possible choice of comparable performance in single-threaded framework. It was not the selected algorithm in `LAPACK` due to its poor parallelizability. However, we will see that in the case of skew-symmetric matrices, QR iterations, or more precisely Francis's iterations [16], are particularly cheap in terms of computations.

2 The tridiagonalization: an Imitation Game©

As explained in section 1, the first step is to obtain the tridiagonalization of $A \in \mathrm{Skew}(n)$. The latter is a particular case of the so-called *Hessenberg reduction*. The Hessenberg reduction computes an almost triangular reduction of $A = QHQ^T$ by the means of an orthogonal similarity transformations $Q \in \mathrm{O}(n)$. The

first subdiagonal remains non-zero.

$$H = Q^T A Q = \begin{bmatrix} * & * & * & ... & * & * & * \\ * & * & * & ... & * & * & * \\ 0 & * & * & ... & * & * & * \\ 0 & 0 & * & ... & * & * & * \\ ... & ... & ... & ... & ... & ... & ... \\ 0 & 0 & 0 & ... & 0 & * & * \end{bmatrix}. \tag{1}$$

When A is skew-symmetric, it is clear that H is tridiagonal since the orthogonal similarity transformation Q preserves the skew-symmetry. Hence, we say $H := T$ and $A = QTQ^T$. We will now see how this similarity transformation is computed in numerical linear algebra.

2.1 A naive skew-symmetric tridiagonalization

The earliest trace I could find of the specialized symmetric tridiagonalization is [17]. The proposed way is to use Householder reflectors. A Householder reflector is an orthogonal matrix $Q = I - \tau v v^T \in \mathrm{O}(n)$ such that for a chosen $x \in \mathbb{R}^n$, we have

$$Qx = (I - \tau v v^T)x = \begin{bmatrix} \|x\|_2 \\ 0 \\ ... \\ 0 \end{bmatrix}. \tag{2}$$

The solution to this is to take $s = x + \|x\|_2 \mathrm{sign}(x_1)e_1$, $v = \frac{s}{s_1}$ and $\tau = \sqrt{s_1}$. This transformation can be applied to tridiagonalize $A \in \mathrm{Skew}(n)$ by successive applications:

$$\hat{A}_1 = (I - \tau_1 v_1 v_1^T) A (I - \tau_1 v_1 v_1^T)^T = \begin{bmatrix} 0 & -\|a_1\|_2 & 0 & ... & 0 \\ \|a_1\|_2 & & & & \\ 0 & & & & \\ ... & & \tilde{A}_1 & & \\ 0 & & & & \end{bmatrix},$$

$$\hat{A}_2 = (I - \tau_2 v_2 v_2^T) \hat{A}_1 (I - \tau_2 v_2 v_2^T)^T = \begin{bmatrix} 0 & -\|a_1\|_2 & 0 & 0 & ... & 0 \\ \|a_1\|_2 & 0 & -\|\tilde{a}_2\|_2 & 0 & ... & 0 \\ 0 & \|\tilde{a}_2\|_2 & & & & \\ 0 & 0 & & & & \\ ... & ... & & \tilde{A}_2 & & \\ 0 & 0 & & & & \end{bmatrix}, \quad \text{etc.}$$

For the symmetric case, it is shown in [17] that a step can be conveniently written as a symmetric update of the matrix $\hat{A}_{i-1}$:

$$\begin{aligned} \hat{A}_i &= \hat{A}_{i-1} - \tau_i v_i v_i^T \hat{A}_{i-1} - \tau_i \hat{A}_{i-1} v_i v_i^T + \tau_i^2 v_i v_i^T \hat{A}_{i-1} v_i v_i^T \\ &= \hat{A}_{i-1} - v_i q_i^T - q_i v_i^T, \end{aligned}$$

where $q_i = \tau_i \hat{A}_{i-1} v_i - \frac{\tau_i^2}{2} v_i^T \hat{A}_{i-1} v_i v_i$. For the skew-symmetric case, we easily see that $v_i^T \hat{A}_{i-1} v_i = 0$. Hence, we obtain skew-symmetric updates:

$$\hat{A}_i = \hat{A}_{i-1} + v_i q_i^T - q_i v_i^T, \tag{3}$$

where $q_i = \tau_i \hat{A}_{i-1} v_i$. In terms of operation count, this procedure is the most efficient known and is very simple to implement. This was implemented under the name `TRIZD` in the previously proposed `FORTRAN` package [15]. According to [14, Lecture 26], the number of floating-point operations needed is $\frac{4n^3}{3}$. However, we can observe that it only features matrix-vector products that are called `BLAS-2` operations in numerical linear algebra. These operations are not efficient in terms of *flops* (floating-point operations per second). Only `BLAS-3` operations (matrix-matrix) reach the peak performance (highest achievable flops of the CPU). Therefore, the method we presented just above is not implemented as is in libraries like `LAPACK`, and `SkewLinearAlgebra.jl` by extension.

2.2 An efficient tridiagonalization

A major innovation for "reduction algorithms" such as tridiagonalization was the development of "blocked algorithms" that maximize the use of `BLAS-3` routines. `LAPACK`'s `sytrd` routine belongs to this family of algorithms. The scientific papers that allowed to go from `BLAS-2` to `BLAS-3` are [3] and [12]. These papers introduced a method to group successive Householder transformations called the "WV representation for products of Householder matrices". It was emphasized that it was possible to write

$$(I - \tau_1 v_1 v_1^T)...(I - \tau_l v_l v_l^T) = I + WV^T, \tag{4}$$

for some well build $W, V \in \mathbb{R}^{n\times l}$. We incite the reader to consult [12] for more details on this construction. The consequence of this result is that if we knew in advance the Householder transformations we wanted to apply to tridiagonalize A, we could do it using only `BLAS-3` operations thanks to Equation (4). The bottleneck is that we are only able to compute the reflector v_k to eliminate the kth column ($k < n$) once we have eliminated all the previous columns. The elimination of these columns can only be performed using `BLAS-2` operations. The trade-off between `BLAS-2` and `BLAS-3` operations gives birth to the "blocked algorithms". The method is the following. Take the $n_\mathrm{b} < n$ first columns of A (n_b is the *block size*). Reduce these n_b columns with `BLAS-2` operations but do not update any element of the $(n - n_\mathrm{b}) \times (n - n_\mathrm{b})$ bottom right submatrix. Meanwhile build a WV representation with the Householder transformations computed. Once the n_b first columns are completely reduced, update the submatrix at once using the WV representation and `BLAS-3` routines. Repeat the operation on the submatrix $A_{n_\mathrm{b}:n,n_\mathrm{b}:n}$ until the matrix is completely tridiagonalized.

The procedure described above is implemented in `sytrd` (or `hetrd` in the case of a complex Hermitian matrix). The code is memory-efficient in the sense that it maximizes cache reuse and minimizes the memory storage needed to tridiagonalize. The package `SkewLinearAlgebra.jl` implements `hessenberg!(A::SkewHermitian)`, an adapted version of `sytrd`/`hetrd` for skew-symmetric/skew-Hermitian matrices. However, an "unfair" advantage makes `sytrd` faster. As explained in [13], the blocked Hessenberg reduction features a remaining matrix-vector (m-v in short) product representing about 20% of the operations, but 70% of the running time (This highlights that `BLAS-3` performs the 80% remaining operations in 30% of the time only). `BLAS` proposes a routine called `symv` to perform a symmetric m-v product in half the time of the general m-v product `gemv`. `symv` has no equivalent skew-symmetric implementation at this day. Hence, `hessenberg!(A::SkewHermitian)` is compelled to use `gemv`. This is the only valuable difference creating a gap of performance between `LAPACK`'s symmetric tridiagonalization and the one implemented in `SkewLinearAlgebra.jl`. For the reader interested in scientific computing and detailed implementations, we encourage to visit the following web pages:

- `sytrd` implementation: `https://netlib.org/lapack/explore-html/d3/db6/group__double_s_ycomputational_gaefcd0b153f8e0c36b510af4364a12cd2.html`
- `hessenberg!(A::SkewHermitian)` implementation: `https://github.com/JuliaLinearAlgebra/SkewLinearAlgebra.jl/blob/main/src/hessenberg.jl`

3 The eigenproblem for tridiagonal matrices

We are now able to tridiagonalize $A \in \mathrm{Skew}(n)$ as $A = QTQ^T$. We want to obtain the eigenvalue decomposition of $T = Vi\Theta V^*$. Computing directly this decomposition would be very expensive as it would ask for complex arithmetic. Our aim is to produce an algorithm making only use of real arithmetic. The key here is to obtain the real Schur decomposition $T = \widetilde{Q}S\widetilde{Q}^T$. It is quite straightforward to go from the real Schur decomposition to the eigendecomposition since the real Schur form S is a block diagonal matrix with blocks $\begin{bmatrix}0\end{bmatrix}$ or $\begin{bmatrix}0 & \theta\\ -\theta & 0\end{bmatrix}$ for some $\theta \in \mathbb{R}$. In `SkewLinearAlgebra.jl`, the algorithm chosen to compute the real Schur decomposition is the double shifted QR algorithm, also called *bulge chasing algorithm* or *Francis's algorithm* as a tribute to its discoverer. The historical papers introducing the QR algorithm and then its double-shifted version are [7] and [8]. The famous 2000 paper [6] ranked *Francis's algorithm* among the 10 most important

algorithms of all times. Funny enough, according to the reviewing paper [16], Francis only learned in 2007 the importance of his contributions after Gene Golub's effort to retrieve his trace.

3.1 Francis's algorithm

In this section, we describe Francis's algorithm for general matrices. We explain further the significant advantage that skew-symmetry provides in its implementation. The first step is to present the QR algorithm with shifts, Algorithm 1 originally presented in [7].

Algorithm 1 QR algorithm with shifts

$H_0 = Q_0^* A Q_0$
for $k = 1, 2, ...$ **do**
  $Q_k R_k = H_k - \mu_k I$
  $H_{k+1} = R_k Q_k + \mu_k I$
end for

[7, Theorem 3] provides the original result about the convergence of H_k to the Schur form. It was then proved that proper shifts μ_k lead to the quadratic convergence of the diagonal elements of H_k to the eigenvalues of A. The problem with Algorithm 1 is that it must deal with complex arithmetic to make appear the complex eigenvalues on the diagonal. The solution to this problem proposed in [8] is to group iterations two by two using conjugated shifts. Doing so, we converge to the real Schur form (real two by two diagonal blocks on the main diagonal), not the complex Schur form. This trick allows to keep real arithmetic. The new iteration takes the form

$$\begin{aligned} Q_k R_k &= (H_k - \mu_k I)(H_k + \bar{\mu}_k I) =: p(H_k) \\ H_{k+2} &= Q_k^T H_k Q_k \end{aligned}$$

The use of conjugated shifts has for consequence that $(H_k - \mu_k I)(H_k + \bar{\mu}_k I)$ is real. Indeed,

$$p(H_k) := (H_k - \mu_k I)(H_k + \bar{\mu}_k I) = H_k^2 - 2\mathrm{Re}(\mu_k) H_k + |\mu_k|^2 I \in \mathbb{R}^{n \times n}. \tag{5}$$

Used as is, the trick would be expensive as it would ask to compute H_k^2. [8, Theorem 11], today known as the *Implicit Q theorem*, provides an astonishing result avoiding to compute H_k^2. This theorem says that if H_k is taken such that its subdiagonal elements are positive, then Q_k is *uniquely* defined by the first column of $p(H_k)$. Hence, it is only needed to compute the first column of $p(H_k)$ and the Householder reflection eliminating this column, that we denote by U_1^T. Applying the similarity transformation U_1 to H_k leads to

$$U_1^T H_k U_1 = \begin{bmatrix} * & * & * & * & * & ... & * & * & * \\ * & * & * & * & * & ... & * & * & * \\ * & * & * & * & * & ... & * & * & * \\ * & * & * & * & * & ... & * & * & * \\ 0 & 0 & 0 & * & * & ... & * & * & * \\ 0 & 0 & 0 & 0 & * & ... & * & * & * \\ ... & ... & ... & ... & ... & ... & ... & ... & ... \\ 0 & 0 & 0 & 0 & 0 & ... & 0 & * & * \end{bmatrix}. \tag{6}$$

Non-zero elements were introduced under the subdiagonal. Indeed, U_1^T was designed to eliminate the first column of $p(H_k)$, not of H_k. We call these subdiagonal non-zero elements the *bulge*. As Q_k is uniquely defined by U_1 (Implicit Q theorem), all we have to do is to eliminate the bulge to retrieve a Hessenberg form. We call U_2^T the transformation that eliminates the first column of the bulge. Applying the similarity

transformation yields

$$U_2^T U_1^T H_k U_1 U_2 = \begin{bmatrix} * & * & * & * & * & ... & * & * & * \\ * & * & * & * & * & ... & * & * & * \\ 0 & * & * & * & * & ... & * & * & * \\ 0 & * & * & * & * & ... & * & * & * \\ 0 & * & * & * & * & ... & * & * & * \\ 0 & 0 & 0 & 0 & * & ... & * & * & * \\ ... & ... & ... & ... & ... & ... & ... & ... & ... \\ 0 & 0 & 0 & 0 & 0 & ... & 0 & * & * \end{bmatrix}. \tag{7}$$

Observe that eliminating the first column of the bulge shifted this bulge to the right. Repeating this shifting operation $n-3$ more times leads to total elimination of the bulge, hence the name of *bulge chasing algorithm*. This process of chasing the bulge completely is known as *Francis's iteration*. Q_k, however never computed, was implicitly applied on H_k by the iteration, hence the name of *Implicit Q theorem*. Finally, we obtain

$$Q_k = U_1 U_2 ... U_{n-1} \quad \text{and} \quad H_{k+2} = Q_k^T H_k Q_k. \tag{8}$$

Algorithm 2 Francis's algorithm

$H_0 = Q_0^T A Q_0$
for $k = 1, 2, ...$ **do**
  Compute $p(H_k)\mathbf{e}_1$.
  Compute U_1^T to eliminate $p(H_k)\mathbf{e}_1$.
  Initiate the bulge by performing $U_1^T H_k U_1$.
  Compute and apply successively $U_2, ..., U_{n-1}$ to H_k to chase the bulge.
  The result $H_{k+1} = Q_k^T H_k Q_k$ was implicitly obtained.
end for

3.2 The skew-symmetric case

In this section, we highlight the significant advantage that skew-symmetry provides to Francis's iteration. First, we recall that the Hessenberg form is here tridiagonal, we denote $H_k := T_k$. We take the "Wilkinson shifts". In the case of a tridiagonal skew-symmetric matrices, this shift is the last subdiagonal element of T_k times $\pm i$. The result on $p(T_k)$ is

$$p(T_k) = T_k^2 + |\mu_k|^2 I \tag{9}$$

The efficiency of Francis's iterations resides in three key observations that allow to create very cheap iterations. I discovered that these observations were already made in 1976 independently of me in the `FORTRAN` package [15].

3.2.1 Observation 1: the initialized bulge is a scalar

We define a 3-Givens rotation as a matrix $G = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \in \mathrm{SO}(3)$ It is easy to verify that $p(T_k)$ is a band-symmetric matrix with a band of two. Moreover, the first sub- and superdiagonal are zero. In consequence, the first column of $p(T_k)$ features only two non-zero elements. The easiest orthogonal matrix

to initialize the bulge is thus a 3-Givens rotations. Let us observe the result on T_0:

$$\begin{bmatrix} c & . & s & . \\ . & 1 & . & . \\ -s & . & c & . \\ . & . & . & 1 \end{bmatrix} \begin{bmatrix} . & -t_1 & . & . \\ t_1 & . & -t_2 & . \\ . & t_2 & . & -t_3 \\ . & . & t_3 & . \end{bmatrix} \begin{bmatrix} c & . & -s & . \\ . & 1 & . & . \\ s & . & c & . \\ . & . & . & 1 \end{bmatrix} =$$
$$\begin{bmatrix} . & -ct_1 + st_2 & . & -st_3 \\ ct_1 - st_2 & . & -st_1 - ct_2 & . \\ . & st_1 + ct_2 & . & -ct_3 \\ st_3 & . & ct_3 & . \end{bmatrix}$$

The bulge, that we denote by β, is reduced to a single number: $\beta = st_3$.

3.2.2 Observation 2: a chased scalar bulge remains scalar

As the bulge β is a scalar, we only need a simple 3-Givens rotation to chase it. Let us observe the effect of this 3-Givens rotation on T_k:

$$\begin{bmatrix} 1 & . & . & . & . \\ . & c & . & s & . \\ . & . & 1 & . & . \\ . & -s & . & c & . \\ . & . & . & . & 1 \end{bmatrix} \begin{bmatrix} . & -t_1 & . & -\beta & . \\ t_1 & . & -t_2 & . & . \\ . & t_2 & . & -t_3 & . \\ \beta & . & t_3 & . & -t_4 \\ . & . & . & t_4 & . \end{bmatrix} \begin{bmatrix} 1 & . & . & . & . \\ . & c & . & -s & . \\ . & . & 1 & . & . \\ . & s & . & c & . \\ . & . & . & . & 1 \end{bmatrix} =$$
$$\begin{bmatrix} . & -ct_1 - s\beta & . & . & . \\ ct_1 + s\beta & . & -ct_2 + st_3 & . & -st_4 \\ . & ct_2 - st_3 & . & -st_2 - ct_3 & . \\ . & . & st_2 + ct_3 & . & -ct_4 \\ . & st_4 & . & ct_4 & . \end{bmatrix}$$

The scalar bulge remains scalar! The two previous observations are enough to obtain the eigenvalues of any skew-symmetric tridiagonal matrix at very low cost. T_k can be stored as a vector and β as a scalar. Performing Francis's iteration reduces to chase the bulge using the formulas obtained above.

3.2.3 Observation 3: The resulting orthogonal transformation is a chessboard matrix

This last observation is only interesting if we want to retrieve the eigenvectors as well. We will observe that the eigenvectors can be assembled at "low" cost (we will precise what we mean by "low"). A chessboard matrix $M \in \mathbb{R}^{n \times n}$ is a matrix such that $M_{i,j} = 0$ if $i + j$ is odd. If we want to build the eigenvectors we have to store the 3-Givens rotations applied on T_k. The first operation is to apply the initialization of the bulge on the identity I_n. Let us notice that the complete Francis's iteration will produce 3-Givens rotations each shifted from one index to the right compared to the previous one in order to "follow" the bulge. Here are the important observations:

- Applying a 3-Givens rotation on a chessboard matrix preserves the chessboard structure.
- Let us number the 3-Givens rotations by the index k. Given a initial chessboard matrix M, we denote G_k these rotations augmented by appropriate identities such that $G_k \in \mathrm{SO}(n)$. Francis's iteration creates the new transformation $MG_1G_2G_3...G_{n-2}$. If k is odd, G_k only modifies the odd columns, and conversely if k is even. Let us define $M^{\mathrm{o}} \in \mathrm{SO}(\lceil \frac{n}{2} \rceil)$ and $M^{\mathrm{e}} \in \mathrm{SO}(\lfloor \frac{n}{2} \rfloor)$ such that for all i, j with $i + j$ even, we have

$$M_{i,j} = \begin{cases} M^{\mathrm{o}}_{\lceil \frac{i}{2} \rceil, \lceil \frac{j}{2} \rceil} & \text{if } i, j \text{ odd} \\ M^{\mathrm{e}}_{\frac{i}{2}, \frac{j}{2}} & \text{if } i, j \text{ even} \end{cases}$$

The nice property is that odd- and even-indexed rotations G_k are decoupled:

$$[MG_1G_2G_3...G_{n-2}]_{i,j} = \begin{cases} [M^{\mathrm{o}}G_1^{\mathrm{o}}G_3^{\mathrm{o}}G_5^{\mathrm{o}}...]_{\lceil \frac{i}{2} \rceil, \lceil \frac{j}{2} \rceil} & \text{if } i,j \text{ odd} \\ [M^{\mathrm{e}}G_2^{\mathrm{e}}G_4^{\mathrm{e}}G_6^{\mathrm{e}}...]_{\frac{i}{2},\frac{j}{2}} & \text{if } i,j \text{ even} \end{cases}$$

All the operations to retrieve the eigenvectors can thus can performed on 2 $\frac{n}{2} \times \frac{n}{2}$ matrices instead of a $n \times n$ matrix. The memory needed during the iteration is divided by two. The number of operations to perform is also divided by two. A non-negligible advantage is that the spatial locality of the operations is increased, dividing also the number of access to the main memory by a factor 2. Finally, recall that a Given rotation only require $\mathcal{O}(6 \cdot \frac{n}{2})$ flop.

Once the real Schur form of the tridiagonal matrix T is obtained, the eigenvalue decomposition is easily obtained. The algorithms `eigen(A)` and `eigvals(A)` applied on `A::SkewHermTridiagonal` or `A::SkewHermitian` available through `SkewLinearAlgebra.jl` implement Francis's algorithm enhanced with the 3 previous observations.

4 The matrix exponential

In this section, we derive an efficient formula to compute the matrix exponential of a skew-symmetric matrix A given the eigenvalue decomposition $A = Vi\Theta V^*$. We pose $V = V_{\mathrm{Re}} + iV_{\mathrm{Im}}$ with $V_{\mathrm{Re}}, V_{\mathrm{Im}} \in \mathbb{R}^{n \times n}$. Computing directly $Ve^{i\Theta}V^*$ would involve costly complex arithmetic. However, we know that e^A is real by definition. Therefore, we must have $\mathrm{Im}\left(Ve^{i\Theta}V^*\right) = 0$ and $e^A = \mathrm{Re}\left(Ve^{i\Theta}V^*\right)$. This yields

$$\begin{aligned} e^A &= \mathrm{Re}\left(Ve^{i\Theta}V^*\right) \\ &= \mathrm{Re}\Big((V_{\mathrm{Re}} + iV_{\mathrm{Im}})[\cos(\Theta) + i\sin(\Theta)](V_{\mathrm{Re}} + iV_{\mathrm{Im}})^*\Big) \\ &= \big[V_{\mathrm{Re}}\cos(\Theta) - V_{\mathrm{Im}}\sin(\Theta)\big]V_{\mathrm{Re}}^T + \big[V_{\mathrm{Re}}\sin(\Theta) + V_{\mathrm{Im}}\cos(\Theta)\big]V_{\mathrm{Im}}^T \end{aligned}$$

This trick allows to obtain the exponential from the eigendecomposition with only two $n \times n$ real matrix products instead of the four needed initially. This formula is implemented in `exp(A::SkewHermitian)`. In the same fashion, `SkewLinearAlgebra.jl` provides the set of efficiently computed trigonometric functions `sin`, `cos`, `cis`, `tan`, `cot`, `sinh`, `cosh`, `tanh`, `coth` and `log`.

5 Performance analysis

In this section, we compare the implementation of the `SkewLinearAlgebra.jl` package against the `LinearAlgebra.jl` package. The latter package implements direct calls to `LAPACK` for the benchmarked functions. Therefore, we obtain a comparison of performance with `LAPACK`. We compare the algorithms in single threaded mode. The matrices used in the test are generated using normal random variables through the function `randn`. The tests are performed in double precision (64 bits) on a Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz 2.21 GHz processor.

We provide a little guidance to the reader to go through Tables 2 and 1. The difference between the "General" and "Skew-symmetric" columns represents the improvement of performance provided by the `SkewLinearAlgebra.jl` package. The "Symmetric" column provides a standard reference to compare the performance a skew-symmetric eigensolver.

- "General": Type `A::Matrix`, `A = randn(n, n)`.
- "Symmetric": Type `A::Symmetric`, `A = Symmetric(randn(n, n))`.
- "Skew-symmetric":
  Type `A::SkewHermitian` (new type), `A = skewhermitian(randn(n, n))`.

- "Symmetric tridiagonal":
  Type `A::SymTridiagonal`, `A = SymTridiagonal(randn(n), randn(n-1))`[1].
- "Skew-symmetric tridiagonal":
  Type `A::SkewHermTridiagonal` (new type), `A = SkewHermTridiagonal(randn(n-1))`.

In `Julia`, the `!` symbol after a function specifies that the method is performed in-place.

Size n	10		100		500		1000	
Scale	$\cdot 10^{-6}s$		$\cdot 10^{-4}s$		$\cdot 10^{-2}s$		$\cdot 10^{-1}s$	
Type	Sym	Skew	Sym	Skew	Sym	Skew	Sym	Skew
`eigvals!`	0.232	0.258	0.176	0.0045	0.371	0.506	0.149	0.199
`eigen!`	11.2	0.520	7.87	2.97	2.35	1.67	0.984	1.50
`exp`	/	4.81	/	4.72	/	3.17	/	3.13

Table 1: Benchmark on tridiagonal matrices

Size n	10			100			500			1000		
Scale	$\cdot 10^{-5}s$			$\cdot 10^{-4}s$			$\cdot 10^{-2}s$			$\cdot 10^{-1}s$		
Type	Gen	Sym	Skew	Gen	Sym	Skew	Gen	Sym	Skew	Gen	Sym	Skew
`hessenberg!`	0.258	0.233	0.421	2.94	1.81	1.89	1.93	0.833	1.18	1.73	0.583	1.18
`eigvals!`	1.42	0.524	0.632	29.1	3.41	3.66	13.1	1.13	1.55	7.02	0.683	1.19
`eigen!`	1.17	1.45	1.01	54.6	9.34	7.23	19.5	3.73	4.66	10.8	2.30	4.15
`exp`	0.732	1.52	1.12	8.73	10.2	8.67	8.29	4.56	6.07	7.49	2.82	5.05

Table 2: Benchmark on dense matrices

Table 1 shows that the tridiagonal skew-symmetric eigensolver features the same performances as the symmetric one. In particular, we can notice that it is faster to obtain the complete eigendecomposition when the matrices are of size n less than 500. This is a result of the "chessboard" structure of the Schur decomposition we observed. Table 2 shows the result on dense matrices. In all the cases, the skew-symmetric algorithms are faster than the general ones, asserting the utility of `SkewLinearAlgebra.jl`. The difference with the symmetric case essentially resides in the gap created during the tridiagonal reduction. For n of the order of 100, we even observe that the skew-symmetric algorithms are faster because the tridiagonal reduction is not yet dominating.

References

[1] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. SIAM, Philadelphia, Pennsylvania, USA, third edition, 1999.

[2] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B Shah. Julia: A fresh approach to numerical computing. *SIAM review*, 59(1):65–98, 2017.

[3] Christian Bischof and Charles Van Loan. The WY Representation for Products of Householder Matrices. *Society for Industrial and Applied Mathematics.SIAM Journal on Scientific and Statistical Computing*, 8(1):1, 01 1987. Copyright - Copyright] © 1987 Society for Industrial and Applied Mathematics.

[4] L Susan Blackford, Antoine Petitet, Roldan Pozo, Karin Remington, R Clint Whaley, James Demmel, Jack Dongarra, Iain Duff, Sven Hammarling, Greg Henry, et al. An updated set of Basic Linear Algebra Subprograms (BLAS). *ACM Transactions on Mathematical Software*, 28(2):135–151, 2002.

[1] It would have been preferable for the comparison to initialize `A = SymTridiagonal(zeros(n), randn(n-1))`. However, we highlighted that such matrices leaded to the failure of `LAPACK`'s `stegr` routine through `LAPACKException (22)`.

[5] Inderjit S. Dhillon and Beresford N. Parlett. Multiple representations to compute orthogonal eigenvectors of symmetric tridiagonal matrices. *Linear Algebra and its Applications*, 387:1–28, 2004.

[6] J. Dongarra and F. Sullivan. Guest editors introduction to the top 10 algorithms. *Computing in Science Engineering*, 2(1):22–23, 2000.

[7] J. G. F. Francis. The QR Transformation a Unitary Analogue to the LR Transformation - part 1. *Comput. J.*, 4:265–271, 1961.

[8] J. G. F. Francis. The QR transformation—part 2. *The Computer Journal*, 4(4):332–345, 01 1962.

[9] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[10] The MathWorks Inc. Matlab version: 9.13.0 (r2022b), 2022.

[11] Cleve Moler and Charles Loan. Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later. *Society for Industrial and Applied Mathematics*, 45:3–49, 03 2003.

[12] Robert Schreiber and Charles Van Loan. A Storage-Efficient WY Representation for Products of Householder Transformations. *SIAM Journal on Scientific and Statistical Computing*, 10, 02 1989.

[13] Stanimire Tomov, Rajib Nath, and Jack Dongarra. Accelerating the reduction to upper Hessenberg, tridiagonal, and bidiagonal forms through hybrid GPU-based computing. *Parallel Computing*, 36(12):645–654, 2010.

[14] Lloyd N. Trefethen and David Bau. *Numerical Linear Algebra*. SIAM, 1997.

[15] R C Ward and L J Gray. Eigensystem computation for skew-symmetric matrices and a class of symmetric matrices. [subroutines trizd, imzd, and tbakzd in fortran for ibm 360/91 computer].

[16] David S. Watkins. Francis's algorithm. *The American Mathematical Monthly*, 118(5):pp. 387–403, 2011.

[17] J.H. Wilkinson. Householder's method for symmetric matrices. (4):354–361, 1962.