MATRIX EXPONENTIATION TIMES A VECTOR USING SCALING AND SQUARING METHOD

HEM N. CHAUDHARY

Contents

1. Introduction

One of the most popular method to exponentiate a matrix is the scaling and squaring method. For given $A \in \mathbb{C}^{n\times n}$ and $B \in \mathbb{C}^{n\times 1}$, $e^A B$ is computed by

$$e^A B = (e^{s^{-1}A})^s B = \underbrace{e^{s^{-1}A} e^{s^{-1}A} \cdots e^{s^{-1}A}}_{\text{s times}} B \tag{1}$$

where $e^{s^{-1}A}$ is calculated by truncated Taylor series:

$$e^{s^{-1}A} = T_{m_*}(s^{-1}A) = \sum_{j=0}^{m_*} \frac{(s^{-1}A)^j}{j\,!} \tag{2}$$

as this allows to determine an optimal choice of s and m_* by exploiting the backward error analysis of Higham [1][2], as refined by Al-Mohy and Higham [3]. To develop the the algorithm to exponentiate a matrix and multiply it by a vector the following three ideas will be used:

- Careful choice of the parameters m and s, exploiting estimates of $\|A^p\|^{1/p}$ for several p, in order to keep the backward error suitably bounded while minimizing the computational cost.
- Shifting the matrix A to reduce the norm of A.
- Premature termination of the truncated Taylor series evaluation based on tolerance.

The approach is similar to applying Runge-Kutta or Taylor series method with fixed stepsize to the underlying ODE $y'(t) = Ay(t),\ y(0) = B$, for which $y(t) = e^{tA}B$, which is the sixth Mler and van Loan's "19 dubious ways"[[5], sect4, [6]]. This algorithm based on scaling and squaring is especially useful for large sparse A, as it could be expensive to naively exponentiate large sparse A.

Acknowledgments to Prof. Steven Johnson for guidance and helping with debugging.

2. Analysis of Algorithm 1

After looking at (1) and (2), we see that the two parameters that are crucial for calculating $e^A B$ are $\boldsymbol{m_*}$ and $\boldsymbol{s}$.

To find $\boldsymbol{m_*}$ and $\boldsymbol{s}$, the following algorithm will be used:

Algorithm 1 This code determines m_* and s given A, *tolerance*(*tol*), m_{max}, and p_{max}

procedure $[m_*,s]$=PARAMETERS(A,TOL)
if equation(3)=True:
    $m_* = argmin_{1\le m\le m_{max}} m\lceil \|A\|_1/\theta_m \rceil$
    $s = \lceil \|A\|_1/\theta_m \rceil$
else:
    Let $m*$ be the smallest m achieving the minimum in equation(4) .
    $s = max(C_{m_*}(A)/m_*, 1)$.
end

Where:

$$\|A\|_1 \le \frac{4 * \theta_{m_{max}} * p_{max} * (p_{max}+3)}{m_{max}} \tag{3}$$

$$C_{m_*} = min\{m\lceil \alpha_p(A)/\theta_m \rceil : 2 \le p \le p_{max}, p(p-1)-1 \le m \le m_{max}\} \tag{4}$$

In the rest of the section, we will look at all the elements of algorithm 1 and the origin of (3) and (4).

Let

$$\Omega_m = \{X \in \mathbb{C}^{n\times n} : \rho(e^{-X}T_m(X) - I) < 1\}$$

where ρ is the spectral radius. Then the function

$$h_{m+1}(X) = log(e^{-X}T_m(X)) \tag{5}$$

is defined for $X \in \Omega_m$. We defined Ω_m so that $log(e^{-X}T_m(X))$ is a valid operation. Then for $X \in \Omega_m$ we have $T_m(X) = e^{X+h_{m+1}(X)}$. Now s is chosen such that $s^{-1}A \in \Omega_m$. Then

$$T_m(s^{-1}A)^s = e^{(s^{-1}A+h_{m+1}(s^{-1}A))s} = e^{A+sh_{m+1}(s^{-1}A)} =: e^{A+\Delta A}$$

where the matrix ΔA represents the backward error resulting from the truncation errors in approximating e^A by $T_m(s^{-1}A)$. Over Ω_m, the function $h_{m+1}(X)$ is

$$h_{m+1}(X) = \sum_{k=m+1}^{\infty} c_k X^k$$

We want to ensure that

$$\frac{\|\Delta A\|}{\|A\|} = \frac{\|h_{m+1}(s^{-1}A)\|}{\|s^{-1}A\|} \le tol$$

for any matrix norm and a given tolerance, tol. By [[3],Thm. 4.2(a)] we have

$$\frac{\|\Delta A\|}{\|A\|} = \frac{\|h_{m+1}(s^{-1}A)\|}{\|s^{-1}A\|} \le \frac{\tilde{h}_{m+1}(s^{-1}\alpha_p(A))}{s^{-1}\alpha_p} \tag{6}$$

where $\tilde{h}_{m+1}(x) = \sum_{k=m+1}^{\infty} |c_k|x^k$ and $\alpha_p(A)$ is

$$\alpha_p(A) = \max(d_p, d_{p+1}), \text{where } d_p(A) = \|A\|_p \tag{7}$$

with p subject to $m+1 \geq p(p-1)$. The reason for working with α_p is that (6) is sharper than the bound $\tilde{h}_{m+1}(s^{-1}\|A\|/(s^{-1}\|A\|))$.

Next, we define

$$\theta_m = \max\{\theta : \tilde{h}_{m+1}(\theta)/\theta \leq tol\} \tag{8}$$

then for any m and p with $m+1 \geq p*(p-1)$ we have $\|\Delta A\| \leq \text{tol}\|A\|$ given $s \geq 1$ is picked such that $s^{-1}\alpha_p(A) \leq \theta_m$. For each m, the optimal value of the integer s is given by $s = \max(\lceil \alpha_p(A)/\theta_m \rceil, 1)$.

As shown in (1), $B_s \approx e^A B$ can be calculated by the recurrence

$$B_{i+1} = T_m(s^{-1}A)B_i \tag{9}$$

which computational cost cost would be

$$C_m := s*m = m*\max(\lceil \alpha_p(A)/\theta_m \rceil, 1) \tag{10}$$

The sequence $\{C_m(A)\}$ is generally decreasing as the sequence $\{\alpha_p(A)\}$ has generally non-increasing trend for any A[[3],sect.1]. Moreover with tol in (8) corresponding to double precision it is found that $\{m/\theta_m\}$ is strictly decreasing. Therefore, larger m corresponds to lesser cost. However, a large value of m is generally unsuitable in floating point arithmetic as it can lead to the evaluation of $T_m(A)B$ with a large $\|A\|$ which further may lead to numerical instability. It will be further explored in upcoming section. Therefore, a limit of m_{max} will be imposed on m to obtain the minimizer m_* over all p such that $p*(p-1) \leq m_{max}+1$. For rest of the analysis in the section, we will drop the max in (10) as its purpose was to deal with nil-potent A, that is, $A^j = 0$ for $j \geq p$. Therefore we have

$$C_m(A) = m*\lceil \alpha_p(A)/\theta_m \rceil$$

We note that $d_1 = \|A\|_1 \geq d_k = \|A\|_k$ for $k \geq 1$ as a result $\alpha_1(A) \geq \alpha_k(A)$. Hence, $p = 1$ is not considered. Let p_{max} denote the largest positive integer p such that $p*(p-1) \leq m_{max}+1$. Then the optimal cost is

$$C_{m_*} = \min\{m\lceil \alpha_p(A)/\theta_m \rceil : 2 \leq p \leq p_{max}, p(p-1)-1 \leq m \leq m_{max}\} \tag{11}$$

where m_* is the smallest value of m at which the minimum is attained. The optimal scaling parameter then is $s = C_{m_*}(A)/m_*$, by (10). From the original Authors' experience $m_{max} = 55$ and $p_{max} = 8$.

In the original paper[4], authors uses their norm estimator algorithm to calculate $\alpha_p(A)$ which has the characteristics that if

$$\|A\|_1 \leq \frac{4*\theta_{m_{max}}*p_{max}*(p_{max}+3)}{m_{max}} \tag{12}$$

then the computational cost, that is, $C_m(A)$ is better approximated by $C_m(A) = m_{max}*\|A\|_1/\theta_{m_{max}}$ instead of $C_m(A) = m_{max}*\|A\|_k/\theta_{m_{max}}$. However, we are using built-in arbitrary norm estimator in Julia, but we assume that Julia's arbitrary norm estimator also has the same characteristics as defined in (12). As we can see in the, section (5), the assumption did not affect error analysis.

3. Preprocessing and termination criterion

Further refinement of the scaling parameter s can be obtained by picking a suitable point about which the Taylor series of the exponential function can be expanded. For a given μ we will approximate e^A while using the shifted matrix $A - \mu I$ by the following expression:

$$e^A = [e^{\mu/s} T_m(s^{-1}(A - \mu I))]^s$$

Original authors empirically found that the shift that minimizes $\|A - \mu I\|_F$ leads to the smaller value of the d_p and hence $\mu = \text{trace}(A)/n$. Importantly, usage of the shift does not make the error analysis invalid, that is, if m_* is chosen based on $\alpha_p(A - \mu I)$ values, then the same backward error bound can be shown to hold.

The derivation of the θ_m did not account for the matrix B, so the choice of m is likely to be larger than necessary for some B. The algorithm 2 returns $e^{A+\Delta A}B$ with the norm-wise relative backward error $\|\Delta A\|/\|A\| \leq$ tol. We now consider truncating the the evaluation of $T_m(A)B_i$ in (9), and by doing that it allows for a norm-wise relative forward error of at most tol to be introduced. With A denoting the scaled and shifted matrix, we accept $t_k(A)B_i$ for the first k as

$$\frac{\|A^{k-1}B_i\|}{(k-1)\,!} + \frac{\|A^k\|B_i}{k\,!} \leq \text{tol}\|T_k(A)B_i\| \tag{13}$$

It was empirically shown that the two terms gives reliable performance as the left-hand side of (13) is meant to better approximate the tail of the series, $\sum_{j=k+1}^{m} A^j B_i/J\,!$.

The value of θ_m for double precision is provided by authors. Moreover, the cost of the algorithm is determined by the number of matrix-vector products shown in line 16 of bellow described algorithm.

Algorithm 2 Given $A \in \mathbb{C}^{n\times n}$, $B \in \mathbb{C}^{n\times 1}$,and a tolerence tol, this algorithm produces an approximation $F \approx e^A * B$.

1: **procedure** [F]=IMPEXPMV(A,B)
2: $\mu = trace(A)/n$
3: $A = A - \mu I$
4: *if* $\|A\|_1 = 0$
5: $m_* = 0$
6: $s = 1$
7: *else*
8: $[m_*, s] = \text{parameters}(A, tol)$
9: end
10: $F = B$,$\eta = e^{\mu/s}$
11: *for i=1:s*
12: $c_1 = \|B\|_\infty$
13: for $j = 1 : m_*$
14: $B = A * B/(s * j)$, $c_2 = \|B\|_\infty$
15: $F = F + B$
16: if $c_1 + c_2 \leq \text{tol}\|F\|_\infty$, break, end
17: $c_1 = c_2$
18: end
19: $F = \eta F$, $B = F$
20: *end*

4. Rounding error analysis and conditioning

For more detail on this section refer to the original paper [4]

5. Bench-marking of impexpmv

To benchmark my implementation named **impexpmv** in Matlab and **impexp** in Julia I calculated the relative error against **expm** in matlab and **exp** in julia respectively. Moreover, I also calculated relative errow between my implementation **impexpmv** against **expmv**, which was implemented by the original authors[4].

Figure(1),(2), and(3) were generated in matlab. In Figure(1), we see the relative error between Higham's implementation **expmv** vs built-in **expm** as a function of randomly generated square matrix and randomly generated corresponding vector of sizes varying from 100 to 1000. The relative error is in the order of 10^{-14}. Similarly, in Figure(2) we see the relative error between **impexpmv** vs **expm** and **expmv** vs **expm**; the relative error for both are in the order of 10^{-14}. Moreover, in the Figure(3) we see the relative error between my implementation **impexpmv** vs Higham's implementation **expmv**. Again the relative error is in the order of 10^{-14}. Therefore, we can conclude that my implementation **impexpmv** behaves similar to Higham's implementation **expmv** and both of them are comparable to built-in matlab function **expm**.

Figure(4),(5),(6), and (7) were generated in julia. In Figure(4) and (5) we look at the relative error of my implementation **impexp** vs built-in **exp** for 1000 by 1000 and 5000 by 5000 randomly generated sparse matrix and their corresponding randomly generated vectors for varying density from 0 to 1. In the Figure (4) and (5), we see the error is in the order of 10^{-15} irrespective of the density from which we can conclude that **impexp** works for matrix with any density.

In Figure(6), we look at the rate of convergence of relative error vs matrix-vector multiplication for **impexp** as a function of tolerence. The tolerence is varying as $10^{-2}, 10^{-4}, 10^{-6}, \cdots 10^{-16}$. In Figure(7), we see the rate of relative convergence between **impexp** and **krylovkit**. Even though my implementation **impexp** has decent convergence rate, in comparision to **krylovkit**, **impexp** is very slow.

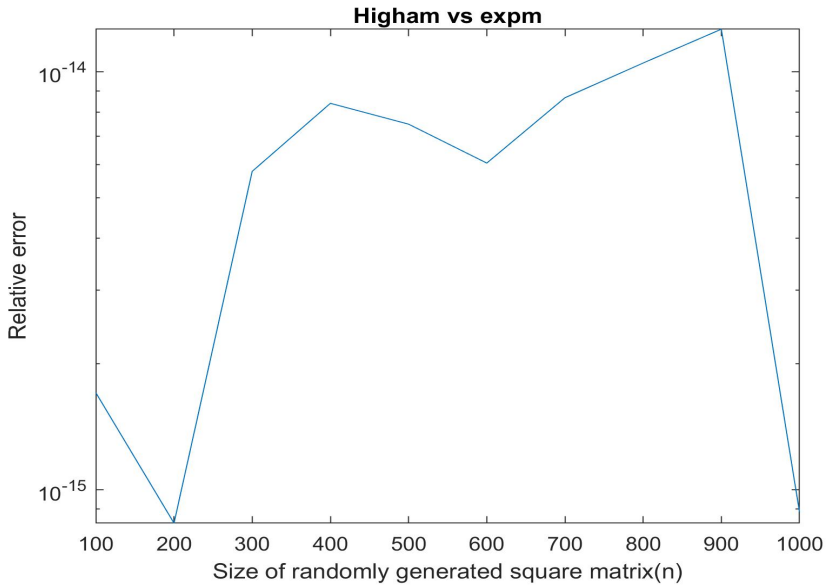

FIGURE 1. Higham's implementation vs built-in expm in matlab

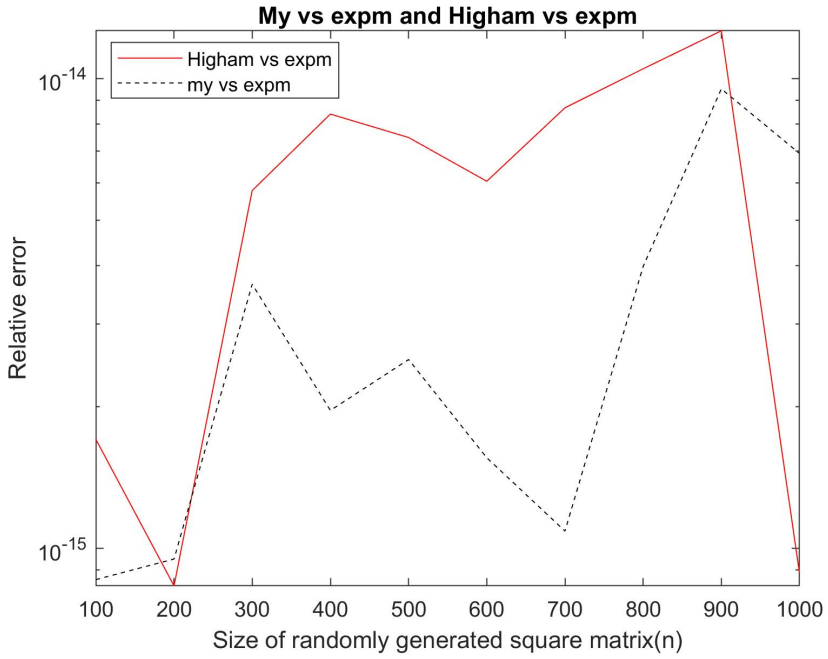

FIGURE 2. Relative error comparison between my implementation vs expm with Higham's implementation vs expm

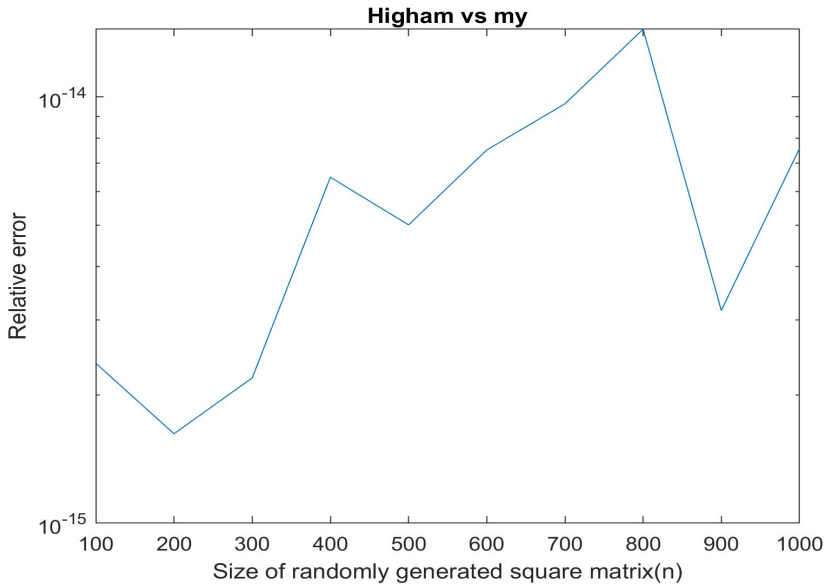

FIGURE 3. Relative error between my implementation vs Higham's implementation

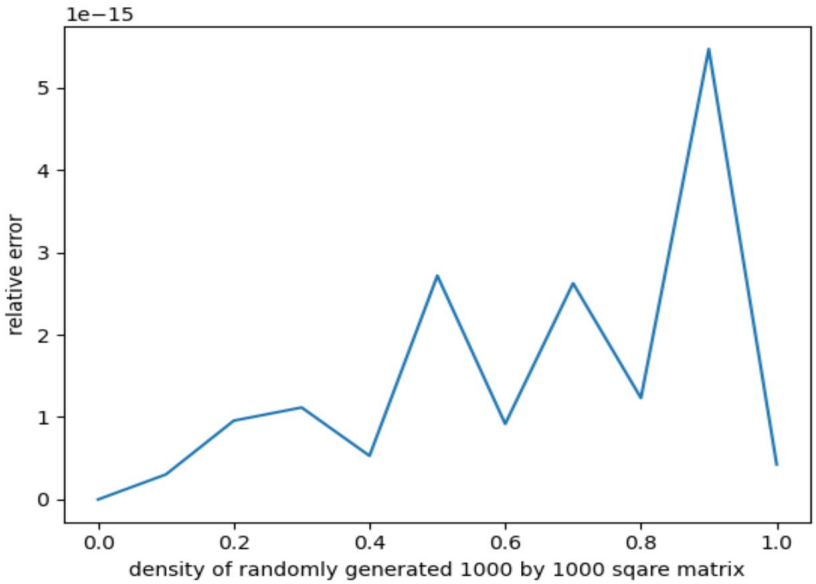

FIGURE 4. Relative error for my implementation vs built-in exp in julia for 1000 by 1000 randomly generated sparse matrix with variable density

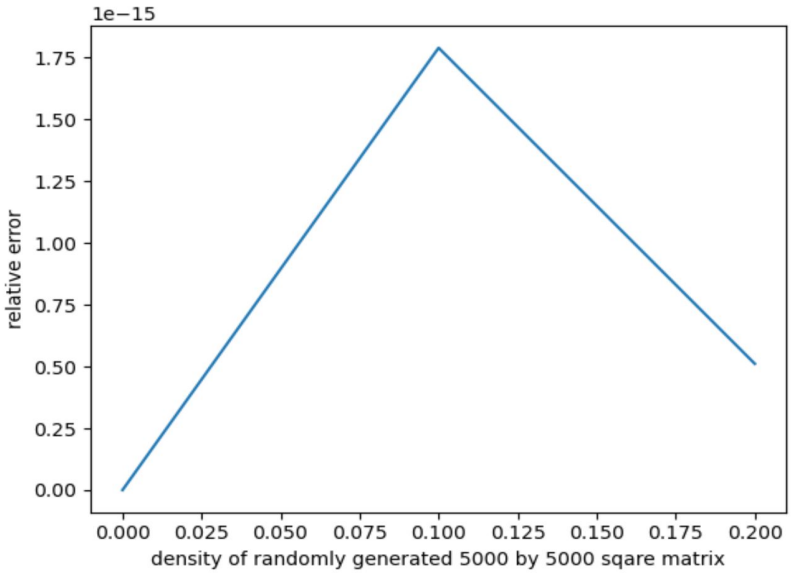

FIGURE 5. Relative error for my implementation vs built-in exp in julia for 5000 by 5000 randomly generated sparse matrix with variable density

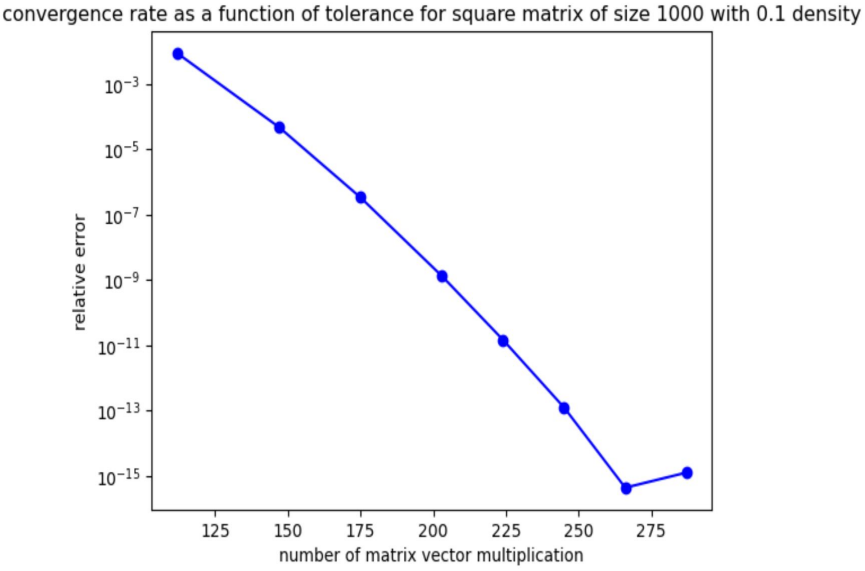

FIGURE 6. Convergance rate of relative erro as a function of tolerance varying from $10^{-2}, 10^{-4}$ to 10^{-16}.

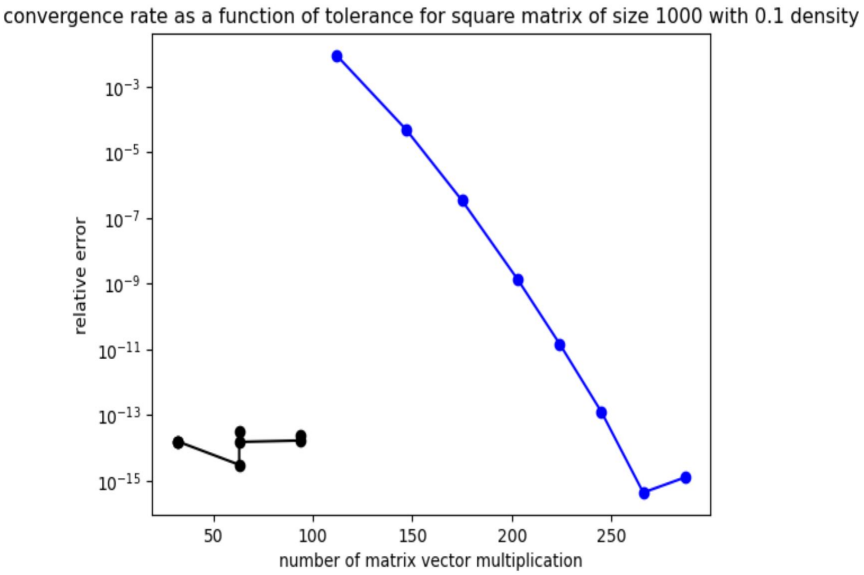

FIGURE 7. Comparison of convergence rate between impexp vs krylovkit. Black is krylovkit, and blue is impexp.

REFERENCES

[1] Nicholas J Higham. "The scaling and squaring method for the matrix exponential revisited". In: *SIAM Journal on Matrix Analysis and Applications* 26.4 (2005), pp. 1179–1193.

[2] Nicholas J Higham. "The scaling and squaring method for the matrix exponential revisited". In: *SIAM review* 51.4 (2009), pp. 747–764.

[3] Awad H Al-Mohy and Nicholas J Higham. "A new scaling and squaring algorithm for the matrix exponential". In: *SIAM Journal on Matrix Analysis and Applications* 31.3 (2010), pp. 970–989.

[4] Awad H Al-Mohy and Nicholas J Higham. "Computing the action of the matrix exponential, with an application to exponential integrators". In: *SIAM journal on scientific computing* 33.2 (2011), pp. 488–511.

[5] Cleve Moler and Charles Van Loan. "Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later". In: *SIAM review* 45.1 (2003), pp. 3–49.

[6] Daniel E Whitney. "More about similarities between Runge-Kutta and matrix exponential methods for evaluating transient response". In: *Proceedings of the IEEE* 57.11 (1969), pp. 2053–2054.