

# SF3580

## HW 3

Anna Broms & Fredrik Fryklund

2018/11/29

### Task 2

(a)

The QR-algorithm is implemented and applied to the given matrix, see Julia code and Figure 1.

(b) and (c)

The eigenvalues of the matrix A are (by construction)  $[2^{[0:7]}, \lambda_9, 2^9]$ , where

$$\lambda_9 = 2^9 \left( 0.99 - \frac{1}{5\alpha} \right). \quad (1)$$

After  $n$  iterations, the elements below the diagonal in the QR-iterates will be *proportional* to  $|\lambda_i/\lambda_j|^n$ , where  $i < j$ . For large  $\alpha$ , the dominating eigenvalues are  $\lambda_9$  and  $\lambda_{10}$ . Computing  $|\lambda_i/\lambda_j|^n$ , for  $i = 9$  and  $j = 10$ , we obtain

$$|\lambda_i/\lambda_j|^n = \left( 0.99 - \frac{1}{5\alpha} \right)^n. \quad (2)$$

Taking this as a measure of the error and setting a tolerance of  $10^{-10}$ , we obtain

$$\begin{aligned} \left( 0.99 - \frac{1}{5\alpha} \right)^n &\leq 10^{-10} \\ \Leftrightarrow n \cdot \log\left(0.99 - \frac{1}{5\alpha}\right) &\leq -10 \log(10) \\ n &\geq \frac{-10 \log(10)}{\log\left(0.99 - \frac{1}{5\alpha}\right)}. \end{aligned} \quad (3)$$

The predicted number of iterations is plotted together with the obtained number of iterations in Figure 1. The predicted number of iterations is proportional to the true number of iterations, as expected.

### Task 3

(a)

Given two nonzero vectors  $x, y \in \mathbb{R}^n$ , we derive a formula for a Householder reflector (represented by a normal direction  $u \in \mathbb{R}^n$ ) such that  $Px = \alpha y$  for some value  $\alpha$ .

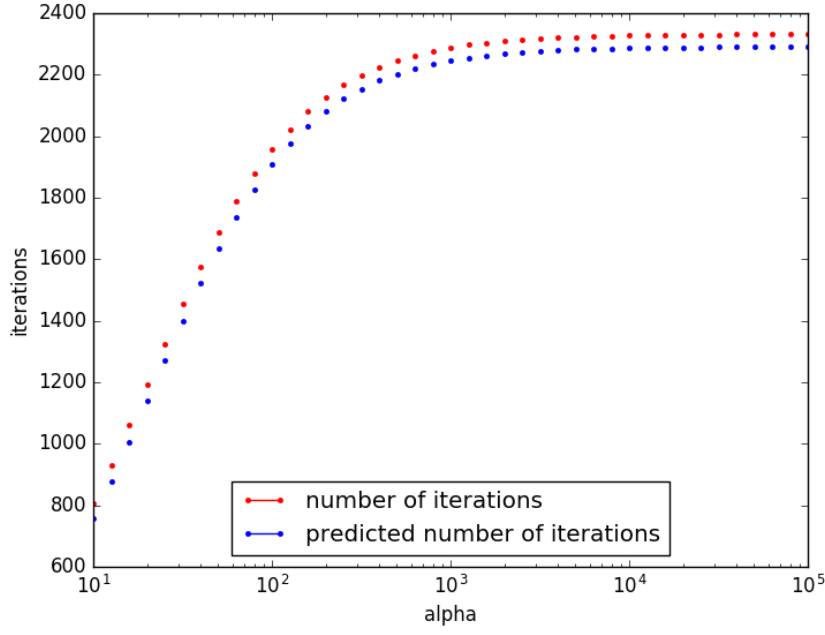


Figure 1: Task 2: obtained and predicted number of iterations for the QR-method applied to the given matrix.

Let  $z = x - \frac{\|x\|}{\|y\|}y$  and  $u = \frac{z}{\|z\|}$ . Then, the matrix  $P = I - 2uu^T$  is a Householder reflector since  $u$  is normalized. Further,  $Px$  can be computed as

$$Px = (I - 2uu^T)x. \quad (4)$$

Consider the product

$$uu^T x = \frac{zz^T x}{\|z\|\|z\|} = \frac{z(z^T x)}{z^T z}, \quad (5)$$

where

$$z^T x = \left(x - \frac{\|x\|}{\|y\|}y\right)^T x = \|x\|^2 - \frac{\|x\|}{\|y\|}y^T x \quad (6)$$

and

$$z^T z = \left(x - \frac{\|x\|}{\|y\|}y\right)^T \left(x - \frac{\|x\|}{\|y\|}y\right) = \|x\|^2 - 2x^T y \frac{\|x\|}{\|y\|} + \|x\|^2 = 2 \left(\|x\|^2 - x^T y \frac{\|x\|}{\|y\|}\right). \quad (7)$$

Now, we easily identify that

$$Px = x - z = \frac{\|x\|}{\|y\|}y = \alpha y, \quad (8)$$

which is what we wanted to show.

## (b) and (c)

Both a naive and an improved Hessenberg reduction algorithm is implemented in Julia. We compare the algorithms by computing a Hessenberg reduction of a given matrix of size  $m \times m$ . The result is found in Table 1.

## (d)

Let  $\hat{H}$  be the result of one step of the shifted QR-method with shift  $\sigma$  for the matrix

$$A = \begin{bmatrix} 3 & 2 \\ \epsilon & 1 \end{bmatrix}. \quad (9)$$

Table 1: CPU-time in seconds for the naive and improved Hessenberg reduction algorithm applied on a specified matrix

	CPU-time naive algorithm	CPU-time improved algorithm
$m = 10$	$1.4 \cdot 10^{-4}$	$4.5 \cdot 10^{-5}$
$m = 100$	$3.3 \cdot 10^{-2}$	$1.5 \cdot 10^{-2}$
$m = 200$	$3.0 \cdot 10^{-1}$	$1.3 \cdot 10^{-1}$
$m = 300$	$1.0 \cdot 10^0$	$3.4 \cdot 10^{-1}$
$m = 400$	$4.6 \cdot 10^{-2}$	$9.8 \cdot 10^{-1}$

Results are found in Table 2.

Table 2: Task 3(d)

$\epsilon$	$ \hat{h}_2, 1 $ $\sigma = 0$	$ \hat{h}_2, 1 $ $\sigma = A_{2,2}$
0.4	0.0961	0.0769
0.1	$3.3 \cdot 10^{-3}$	$5.0 \cdot 10^{-5}$
0.01	$3.3 \cdot 10^{-4}$	$5.0 \cdot 10^{-7}$
$10^{-3}$	$3.3 \cdot 10^{-5}$	$5.0 \cdot 10^{-9}$
$10^{-4}$	$3.3 \cdot 10^{-6}$	$5.0 \cdot 10^{-11}$
$10^{-5}$	$3.3 \cdot 10^{-7}$	$5.0 \cdot 10^{-13}$
$10^{-6}$	$3.3 \cdot 10^{-8}$	$5.0 \cdot 10^{-15}$
$10^{-7}$	$3.3 \cdot 10^{-9}$	$5.0 \cdot 10^{-17}$
$10^{-8}$	$3.3 \cdot 10^{-10}$	$5.0 \cdot 10^{-19}$
$10^{-9}$	$3.3 \cdot 10^{-11}$	$5.0 \cdot 10^{-21}$
$10^{-10}$	$3.3 \cdot 10^{-12}$	$5.0 \cdot 10^{-23}$

The values in the table corresponds to the values on the off- diagonal which can be seen as the error while computing the eigenvalues of the matrix  $A$ . The error decreases much faster with  $\epsilon$  when the shifted QR-method is used than when the unshifted QR-method is used (corresponding to  $\sigma = 0$ ).

## Task 5

(a)

The matrix  $A$  is diagonalizable with the eigendecomposition  $A = QDQ^{-1}$ , where  $D$  is a diagonal matrix. For such structures it holds that  $\sin(A) = Q \sin(D)Q^{-1}$ . Thus we can validate the result for the Schur-Parlett method, which is

$$\sin(A) = \sin\left(\begin{bmatrix} 1 & 4 & 4 \\ 3 & -1 & 3 \\ -1 & 4 & 4 \end{bmatrix}\right) \approx \begin{bmatrix} 0.846192 & 0.0655435 & -0.187806 \\ 0.33476 & 0.385017 & -0.141244 \\ -0.190921 & 0.192478 & 0.848269 \end{bmatrix}. \quad (10)$$

which in norm differs  $4.28e - 16$  from  $Q \sin(D)Q^{-1}$ .

(b) & (c)

It is clear from Figure 2 that the number of flops required for Schur-Parlett is not discernibly affected by  $N$ , at least for  $N \in 10, 50, 100, 150, 200, 250, 300$ . This is not suprising, as the most computationally demanding part of the Schur-Parlett method is often performing the Schur decomposition, which scales like  $O(n^3)$ . Once obtained, the function  $f$  is only applied to the diagonal elements, which are scalars.

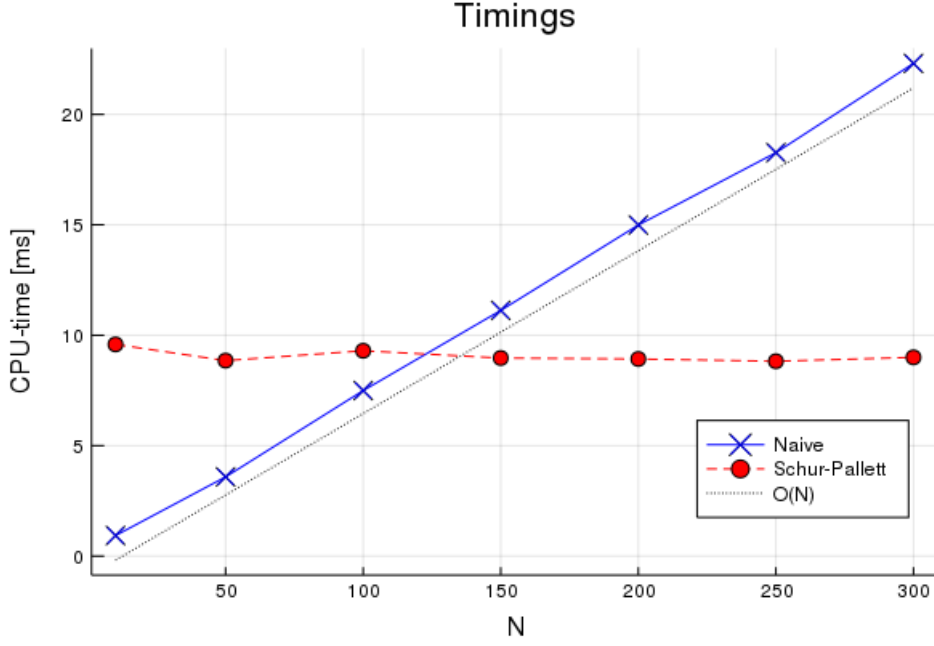


Figure 2: Task 5, (b) & (c): CPU-time in milliseconds, as a function of  $N$ .

For the naive approach the number of flops is proportional to  $N$ . A matrix multiplication is of  $O(n^3)$ , thus performing  $N$  matrix gives  $O(Nn^3)$ , which we read from Figure 2. The black line corresponds to the line  $0.08 + 0.07 N$ .

## Task 6

(a)

The matrix

$$A = \begin{bmatrix} \pi & 1 \\ 0 & \pi + \varepsilon \end{bmatrix}, \quad (11)$$

with  $\varepsilon > 0$ , has two eigenvalues:  $\lambda_1 = \pi$  and  $\lambda_2 = \pi + \varepsilon$ . The Jordan canonical form definition gives that

$$p(A) = X \operatorname{diag}(p(J_1), p(J_2)) X^{-1}, \quad (12)$$

where  $A = X \operatorname{diag}(J_1, J_2) X^{-1}$ , with  $J_1 = \lambda_1$  and  $J_2 = \lambda_2$ .

A simple consequence is

$$g(A) = X \operatorname{diag}(g(\lambda_1), g(\lambda_2)) X^{-1} = X \operatorname{diag}(p(\lambda_1), p(\lambda_2)) X^{-1} = p(A), \quad (13)$$

since the polynomial  $p$  interpolates the function  $g$  in the eigenvalues of  $A$ , i.e.  $p(\lambda_1) = g(\lambda_1)$  and  $p(\lambda_2) = g(\lambda_2)$ .

Two points define a unique polynomial of order 1, thus we may choose a  $p$  in  $\mathbb{P}^1$  and write  $p(z) = \alpha + \beta z$ . The unknown coefficients are obtained by solving

$$\begin{cases} \alpha + \beta \lambda_1 = g(\lambda_1) \\ \alpha + \beta \lambda_2 = g(\lambda_2) \end{cases} \Leftrightarrow \begin{cases} \alpha = \frac{g(\lambda_1)\lambda_2 - g(\lambda_2)\lambda_1}{\lambda_2 - \lambda_1} \\ \beta = \frac{g(\lambda_2) - g(\lambda_1)}{\lambda_2 - \lambda_1} \end{cases} \quad (14)$$

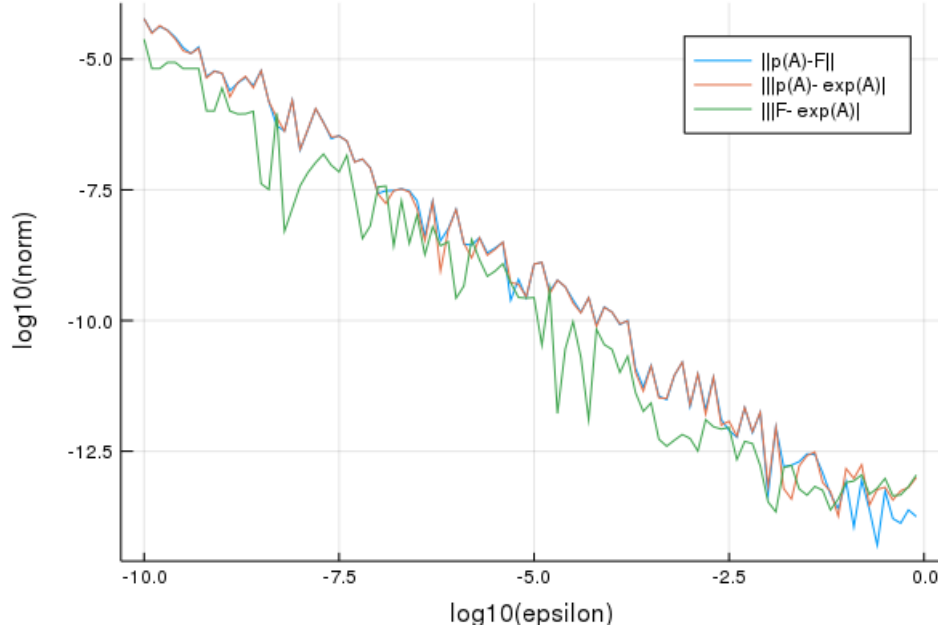


Figure 3: Task 5, (b) & (c): CPU-time in milliseconds, as a function of  $N$ . Here  $f(A)$  corresponds to (15),  $F$  to evaluating via Jordan canonical form and  $\exp$  Julia's native function.

(b)

Given  $g := \exp$  we have from (a) and (b) that

$$p(A) = \alpha I + \beta A = \frac{\exp(\pi)(\pi + \varepsilon) - \exp(\pi + \varepsilon)\pi}{\varepsilon} I + \frac{\exp(\pi + \varepsilon) - \exp(\pi)}{\varepsilon} A \quad (15)$$

$$= \frac{\exp(\pi)}{\varepsilon} (\varepsilon I + (1 - \exp(\varepsilon))(\pi I - A)) \quad (16)$$

(c)

It is known that the Jordan decomposition is unstable for non-symmetric matrices, as the eigenvalues may lie close to each other. For the given matrix  $A$  this can be tuned artificially by setting  $\varepsilon = |\lambda_1 - \lambda_2|$ , thus smaller  $\varepsilon$  should give larger errors. In Figure 15 we see that this is indeed the case, or rather the difference  $\|f(A) - F\|$  grows as epsilon decreases. Still we are sceptical to the values produced by (15). As  $\varepsilon$  becomes small the coefficients  $\alpha$  and  $\beta$  will most likely suffer from cancellation. Therefore we tried to use Julia's native function `exp` to compute reference values. However, judging from Figure 15, the function `exp` doesn't seem to perform better than the other two methods. Thus it is hard to claim which method that is preferable.

## Task 7

(a)

*Comment: We have two alternative solutions in this exercise where we have used the definition of a matrix function in two different ways. Please comment if the second way to solve the problem also is valid, where we see the function  $f$  as a function of two variables  $f = f(z, t)$  and make a Taylor expansion in  $z$  only.*

### Solution

Let  $\mu \in \mathbb{C}$  be an expansion point, then

$$f(tA) = \sum_{i=0}^{\infty} \frac{f^{(i)}(\mu)}{i!} (tA - \mu I)^i. \quad (17)$$

For  $f(z) = \exp(z)$  the function is analytic and we can without loss of generality set  $\mu = 0$ . For now assume that  $\frac{d}{dt}(tA)^i = iA(tA)^{i-1}$ , then

$$\frac{d}{dt} \exp(tA) = \frac{d}{dt} \sum_{i=0}^{\infty} \frac{\exp(0)}{i!} (tA - 0I)^i = \sum_{i=0}^{\infty} \frac{1}{i!} \frac{d}{dt} (tA)^i = A \sum_{i=1}^{\infty} \frac{1}{(i-1)!} (tA)^{i-1} = A \sum_{i=0}^{\infty} \frac{1}{i!} (tA)^i = A \exp(tA), \quad (18)$$

by shifting the indices  $i \rightarrow i+1$ . We now motivate the claim above. By definition

$$\frac{d}{dt}(tA)^i = \lim_{\epsilon \rightarrow 0} \frac{((t+\epsilon)A)^i - (tA)^i}{\epsilon} = \lim_{\epsilon \rightarrow 0} \frac{(t+\epsilon)^i - t^i}{\epsilon} A^i = \frac{d}{dt}(t^i) A^i = iA(tA)^{i-1}. \quad (19)$$

Since  $it^{i-1}A^{i-1}A = iAt^{i-1}A^{i-1}$  we have  $A \exp(tA) = \exp(tA)A = \frac{d}{dt} \exp(tA)$ .

### Alternative solution

Consider the function  $f(z, t) = e^{tz}$ . We want to investigate the matrix valued function  $f(A, t) = e^{At}$ . Let  $\mu \in \mathbb{C}$  be an expansion point. Then,

$$f(A, t) = \sum_{i=0}^{\infty} \frac{f^{(i)}(\mu, t)}{i!} (A - \mu I)^i = \sum_{i=0}^{\infty} \frac{t^i e^{t\mu}}{i!} (A - \mu I)^i. \quad (20)$$

If  $A \in \mathbb{C}^{n \times n}$ , then  $f : \mathbb{C}^{n \times n} \rightarrow \mathbb{C}^{n \times n}$ . Now, compute the derivative of  $f(A, t)$  with respect to time:

$$\begin{aligned} \frac{d}{dt} e^{tA} &= \frac{d}{dt} \sum_{i=0}^{\infty} \frac{t^i e^{t\mu}}{i!} (A - \mu I)^i = \\ &= \sum_{i=0}^{\infty} \frac{d}{dt} \left( \frac{t^i e^{t\mu}}{i!} (A - \mu I)^i \right) = \\ &= \sum_{i=0}^{\infty} \frac{d}{dt} \left( \frac{t^i e^{t\mu}}{i!} \right) (A - \mu I)^i = \\ &= \sum_{i=0}^{\infty} \left( \frac{it^{i-1} e^{t\mu} + t^i \mu e^{t\mu}}{i!} \right) (A - \mu I)^i. \end{aligned} \quad (21)$$

The last expression can be identified as  $g(A)$ , where  $g(z) = z e^{tz}$  as the expression

$$(it^{i-1} e^{t\mu} + t^i \mu e^{t\mu}) \quad (22)$$

is the  $i$ th derivative of the product  $z \cdot e^{tz}$ , which can be seen using the general Leibniz rule  $((f_1 f_2)^{(n)}) = \sum_{k=0}^n \binom{n}{k} f_1^{(n-k)}(x) f_2^{(k)}(x)$ . Thus, we can conclude that  $\frac{d}{dt} e^{tA} = A e^{tA}$ . The matrix function  $e^{tA} A$  has the same Taylor expansion expression as  $A e^{tA}$ . Thus,  $\frac{d}{dt} e^{tA} = A e^{tA} = e^{tA} A$ , which is what we wanted to show.

### (b)

Introduce

$$[B, A]_n = [[B, A]_{n-1}, A], n = 0, 1, 2, \dots, \quad \text{where } [B, A]_1 = [B, A] = BA - AB \text{ and } [B, A]_0 = B. \quad (23)$$

We will later in the proof use that  $[A + B, C]_n = [A, C]_n + [B, C]_n$ , which is shown by induction: the initial case is  $[A + B, C]_1 = AC - CA + BC - CB = [A, C]_1 + [B, C]_1$ . Now assume  $[A + B, C]_n = [A, C]_n + [B, C]_n$  holds, then

$$[A + B, C]_{n+1} = [AC - CA + BC - CB, A]_n = [AC - CA, C]_n + [BC - CB, C]_n \quad (24)$$

$$= [[A, C], C]_n + [[B, C], C]_n = [A, C]_{n+1} + [B, C]_{n+1}. \quad (25)$$

Let  $G(t) = \exp(-tA)B \exp(tA)$ , which is analytic in  $t$ . Thus we may write

$$G(t) = \sum_{i=0}^{\infty} \frac{t^i}{i!} G^{(i)}(\mu) = \sum_{i=0}^{\infty} \frac{t^i}{i!} G_i, \quad (26)$$

where  $G_0 = B$ . By (a) and that  $[A + B, C]_n = [A, C]_n + [B, C]_n$  we have that

$$\frac{d}{dt} G(t) = G(t)A - AG(t) = [G(t), A] = \left[ \sum_{i=0}^{\infty} \frac{t^i}{i!} G_i, A \right] = \sum_{i=0}^{\infty} \frac{t^i}{i!} [G_i, A] \quad (27)$$

Setting this to be equal to the the derivative of (26) with respect to  $t$  gives

$$\sum_{i=0}^{\infty} \frac{t^i}{i!} [G_i, A] = \sum_{i=1}^{\infty} \frac{t^{i-1}}{(i-1)!} G_i. \quad (28)$$

By shifting the indexing from  $i = 1, 2, \dots$  to  $i = 0, 1, \dots$  for the right hand side we get

$$\sum_{i=0}^{\infty} \frac{t^i}{i!} [G_i, A] = \sum_{i=0}^{\infty} \frac{t^i}{i!} G_{i+1}. \quad (29)$$

We conclude that  $G_{i+1} = [G_i, A]_i$ , that is  $G_1 = [G_0, A]_0 = B$  and

$$G(t) = B + t[B, A] + \frac{t^2}{2!} [[B, A], A] + \frac{t^2}{2!} [[B, A], A, A] + \dots \quad (30)$$

(c)

We identify the integrand as  $G(t)$ , that is

$$P = \int_0^{\tau} \exp(tA^T)B \exp(tA) dt = \int_0^{\tau} \exp(-tA)B \exp(tA) dt = \int_0^{\tau} G(t) dt. \quad (31)$$

Introduce

$$P_n = \int_0^{\tau} \sum_{i=0}^n \frac{t^i}{i!} G_{i+1} dt = \sum_{i=0}^n \int_0^{\tau} \frac{t^i}{i!} G_{i+1} dt, \quad (32)$$

Since the integrand is uniformly convergent, assuming  $\tau$  finite, it holds that

$$\lim_{n \rightarrow \infty} P_n = \lim_{n \rightarrow \infty} \int_0^{\tau} \sum_{i=0}^n \frac{t^i}{i!} G_{i+1} dt = \int_0^{\tau} \lim_{n \rightarrow \infty} \sum_{i=0}^n \frac{t^i}{i!} G_{i+1} dt = \int_0^{\tau} G(t) dt = P \quad (33)$$

where the limit was moved inside due to the dominated convergence theorem. Furthermore,

$$\int_0^{\tau} \frac{t^i}{i!} G_{i+1} dt = [G_i, A] \frac{\tau^{i+1}}{(i+1)!}, \quad (34)$$

for every  $i$ . Thus

$$P = \lim_{n \rightarrow \infty} \sum_{i=0}^n \int_0^{\tau} \frac{t^i}{i!} G_{i+1} dt = \sum_{i=0}^{\infty} [G_i, A] \frac{\tau^{i+1}}{(i+1)!}. \quad (35)$$

**(d)**

Task: Let  $C_k = [C_{k-1}, A]$ , with  $C_0 = B$ . We want to show that  $\|C_k\| \leq 2^k \|A\|^k \|B\|$ .

The proof is done by induction. For  $k = 0$  we have that  $\|C_0\| = \|B\| \leq 2^0 \|A\|^0 \|B\|$ . Now, assume that  $\|C_k\| \leq 2^k \|A\|^k \|B\|$ . We want to show that  $\|C_{k+1}\| \leq 2^{k+1} \|A\|^{k+1} \|B\|$ :

$$\begin{aligned} \|C_{k+1}\| &= \|C_k A - A C_k\| = \|C_k A + (-A C_k)\| \leq \|C_k A\| + \|-A C_k\| = \|C_k A\| + \|A C_k\| = \|C_k A + (-1) A C_k\| \leq \|C_k\| \|A\| + \|A\| \|C_k\| = \\ &= 2 \|A\| \|C_k\| = 2^{k+1} \|A\|^{k+1} \|B\|, \end{aligned} \quad (36)$$

which is what we wanted to show.

**(e)**

Suppose  $\|A\| < \frac{1}{2}$  and  $t \leq 1$ . Let  $G_N(t)$  be the truncation of  $G(t)$ , where

$$G(t) = \sum_{k=0}^{\infty} \frac{t^k}{k!} C_k. \quad (37)$$

Then,

$$\begin{aligned} \|G_N(t) - G(t)\| &= \left\| \sum_{k=N+1}^{\infty} \frac{t^k}{k!} C_k \right\| \leq \sum_{k=N+1}^{\infty} \left( \frac{t^k}{k!} \right) \|C_k\| \leq \\ &\leq \sum_{k=N+1}^{\infty} \left( \frac{t^k}{k!} \right) 2^k \|A\|^k \|B\| \leq \|B\| \sum_{k=N+1}^{\infty} \frac{t^k}{k!} = \|B\| \left( \sum_{k=0}^{\infty} \frac{t^k}{k!} - \sum_{k=0}^N \frac{t^k}{k!} \right) = \|B\| \left( e^t - \sum_{k=0}^N \frac{t^k}{k!} \right). \end{aligned} \quad (38)$$

where we have first used the result from (d). Passing the limit  $N \rightarrow \infty$  we see that  $\|G_N(t) - G(t)\| \rightarrow 0$ . That is for all  $\|A\| \leq 1$  and  $t \leq 1$ . This implies that for some  $N$  the difference  $\|G_N(t) - G(t)\|$  is equal to the  $N + 1$ :th term in the expansion, evaluated in some unknown point  $T \in [0, 1]$ . In other words, the error bound is

$$\|G_N(t) - G(t)\| = \|B\| \frac{t^{N+1}}{(N+1)!} e^T. \quad (39)$$

**(f)**

Using the results from (c), with

$$P = \sum_{i=0}^{\infty} G_{i+1} \frac{\tau^{i+1}}{(i+1)!} = \sum_{i=1}^{\infty} G_i \frac{\tau^i}{i!} = \sum_{i=0}^{\infty} G_i \frac{\tau^i}{i!} - B, \quad (40)$$

we denote by  $P^N$  the computation of  $P$  truncated at  $N$  terms and obtain that

$$\|P - P^N\| = \|G_N(\tau) - G(\tau)\|. \quad (41)$$

Using the estimate in (e), we thus have that

$$\|P_N - P\| \leq \int_0^{\tau} \|B\| \frac{t^{N+1}}{(N+1)!} e^t = \|B\| \frac{\tau^{N+2}}{(N+2)!} e^T, \quad T \in [0, \tau]. \quad (42)$$



Thus we may a priori estimate the number of iterations that are required by estimating the error as  $1/(N+2)!$ . now design an algorithm for  $P$  (in pseudocode) as:

```

G = B; - Sets initial element  $G_0$ .
P = G; - Sets initial element, as we start at  $i = 1$ 
i = 1; - Number of iterations
err = 1;
t = 1;
for  $i = 1:N$  do
    G = GA-AG;
    t =  $t\tau/i$ ;
    P = P+Gt;
end

```

**Algorithm 1:** Algorithm for computing the integral  $P$ .

(g)

We compare the algorithm in (f) to the naive numerical integration approach in Julia. We use  $\tau = 1$  and the Neumann matrix from the library of the package MatrixDepot. See the attached code. We set an tolerance of  $1e-14$ . For the algorithm devised in (f) the CPU time in secons, including estimating  $N$ , is about 0.03 seconds and the estimated error is  $2.8e-15$ . As a substitute for Matlabs integral we used `quadgk` from the package sharing its name. The corresponding time is 1.21 seconds, which is about 33 times slower. The estimated error, which `quadgk` gives as an output, is  $1.0e-14$ . The difference between the two results are about  $1e-14$ .