NumIntro Aflevering

Victor Z. Nygaard, nfq499.

November 28, 2020

Opgave 1

Se "prog1.py" for implementeringen, og kommentering af programmet.

Det bemærkes i denne forbindelse at der i selve programmet forekommer betydeligt udvidet kommentering og tilbagerapportering (printing) af hvad der måske kan være at foretrække i en faktisk implementering, da dette tænkes at kunne være til fordel for læser.

Vi får gennem implementeringen følgende resultater til følgende rapporterede præcision;

$$\exp(Ax) \overset{stop\ ved}{=} S_{47_A} = \begin{pmatrix} 0.3659571 & 0.35453832 & 0.27950458 \\ 0.36527461 & 0.35510049 & 0.2796249 \\ 0.36551524 & 0.35489927 & 0.27958549 \end{pmatrix}.$$

$$\exp\left(\frac{A}{20}x\right) \overset{stop\ ved}{=} S_{9_{\frac{A}{20}}} = \begin{pmatrix} 0.79463194 & 0.09063793 & 0.11473013 \\ 0.0915412 & 0.76331361 & 0.14514519 \\ 0.15237133 & 0.18188312 & 0.66574555 \end{pmatrix}.$$

Resultaterne viser sig at være i rigtig god overensstemmelse med 'expm'-funktionen fra scipy.linalg.
Overstående stop (ved hhv. $M = 47, 9$ i summerne) forekommer på baggrund af en standard sat epsilon tolerance på 10^{-8} for inf-normen af bidraget ved det efterfølgende afsnitssum led. For S_n værende afsnitssummen op til n for enten $\frac{A}{20}$ eller A stopper vi altså summerne og afgiver S_n når

$$||S_{n+1} - S_n||_\infty < \varepsilon = 10^{-8}.$$

Implementeringen betragter dog ikke S_{n+1} og S_n men i stedet de underliggende følgeled.

Opgave 2

Opgave 2.1

Vi siger at en matrix $A = \{a_{ij}\}_{i,j=1,\dots,n} \in M_{n,n}(\mathbb{R})$ er en sub-intensitets matrix hvis;

$$a_{ij} \geq 0, \qquad i \neq j \tag{1}$$

$$a_{ii} < 0 \leq -\sum_{\substack{j=1 \\ j\neq i}}^{n} a_{ij}, \qquad i = 1, \dots, n, \tag{2}$$

mens den kaldes en intensitetsmatrix hvis den svage ulighed i (2) ændres til lighedstegn, altså hvis den opfylder (1) og

$$a_{ii} = -\sum_{\substack{j=1 \\ j\neq i}}^{n} a_{ij},\ i = 1, \dots, n. \tag{3}$$

Vi definerer ligeledes $\eta := \max_{i=1,\dots,n} \{-a_{ii}\}$, og betragter

$$P := I + \frac{1}{\eta} A,$$

for I værende $n \times n$ identitetsmatricen. Antag nu at A er en sub-intensitets matrix - Vi ønsker at vise at P dermed vil være substokastisk, altså at den opfylder

$$p_{ij} \geq 0 \qquad i,j = 1,\dots,n \tag{4}$$

$$\sum_{j=1}^{n} p_{ij} \leq 1 \qquad i = 1,\dots,n. \tag{5}$$

Bemærk at hvis P skulle være en stokastisk matrix ville den skulle opfylde (4) og

$$\sum_{j=1}^{n} p_{ij} = 1 \;\; i = 1,\dots,n. \tag{6}$$

Fra definitionen af $P = \{p_{ij}\}_{i,j=1,\dots,n}$ fås

$$p_{ij} = \begin{cases} 1 + \frac{1}{\eta} a_{ii}, & i = j \\ \frac{1}{\eta} a_{ij}, & i \neq j \end{cases}, \tag{7}$$

da I betegner identitetsmatricen.
Fra (2) haves for vilkårligt $i = 1,\dots,n$ at $a_{ii} < 0$ og dermed $\eta > 0$, således at $\frac{a_{ii}}{\eta} < 0$.
Fra definitionen af η kan det samtidigt bemærkes at idet

$$|\eta| \equiv \left| \max_{i=1,\dots,n} \{-a_{ii}\} \right| \geq |a_{ii}|, \;\; i = 1,\dots,n,$$

vil

$$\left| \frac{a_{ii}}{\eta} \right| \leq 1,$$

således at $\frac{a_{ii}}{\eta} \in [-1, 0)$, hvormed $p_{ii} \equiv 1 + \frac{a_{ii}}{\eta} \geq 0$.

Ligeledes vil gælde at $p_{i \neq j} \equiv \frac{a_{i \neq j}}{\eta} \geq 0$, da $a_{i \neq j} \geq 0$ per (1) og $\eta > 0$. - Ergo er kriterie 1, som set i (4) for at være substokastisk matrix; $p_{i,j} \geq 0$, $i,j = 1,\dots,n$ opfyldt.

For at vise at P også opfylder (5), betragt vilkårligt fast $i \in \{1,\dots,n\}$ og bemærk at

$$\begin{aligned} \sum_{j=1}^{n} p_{ij} &\equiv p_{ii} + \sum_{\substack{j=1 \\ j \neq i}}^{n} p_{ij} \\ &\overset{(7)}{=} 1 + a_{ii} \frac{1}{\eta} + \sum_{\substack{j=1 \\ j \neq i}}^{n} \frac{1}{\eta} a_{ij} \\ &= 1 + \frac{1}{\eta} \left(a_{ii} + \sum_{\substack{j=1 \\ j \neq i}}^{n} a_{ij} \right) \end{aligned}$$

$$\overset{(2)}{\underset{(*)}{\leq}} 1 + \frac{1}{\eta} \cdot 0 = 1.$$

Hermed er både (4) og (5) opfyldt.

Antages det i stedet at A ikke "bare" er en sub-intensitetsmatrix, men er en "fuld" intensitetsmatrix som dermed opfylder (1) og (3) kan overstående argument følges uden ændringer indtil uligheden $(*)$, som idet vi nu har (3), vil der i stedet for en ulighed være en lighed, hvormed vi får

$$\sum_{j=1}^{n} p_{ij} = 1,$$

for vilkårligt $i \in \{1, \ldots, n\}$, således at P vil være en stokastisk matrix.

Opgave 2.2

Vi bemærker at vi fra kurser som Diff (Kapitel 2.5 ligning 3 i kursusmaterialet 2019-2020), og Stok1 har det nyttige resultat vedrørende $n \times n$ matricer M og P;

$$MP = PM \Rightarrow \exp(M+P) = \exp(M)\exp(P). \tag{8}$$

Bemærk nu at idet vi for A værende en sub-intensitets matrix i Opgave 2,1 har vist at $P = I + \frac{1}{\eta}A$ for P nødvendigvis værende en substokastisk matrix, at $A = (P - I)\eta$.
Herved haves at

$$\exp(Ax) \equiv \exp((P-I)\eta x) = \exp((P\eta x + (-I\eta x))). \tag{9}$$

Da η og x er skalarer vil både $P\eta x$ og $(-I\eta x)$ fortsat være $n \times n$ matricer og da identitetsmatricen kommuterer med alting, specielt; $PI \equiv P \equiv IP$, vil vi idet matricer også kommuterer med skalarmultiplikation af konstanter (per definition/konvention fra LinAlg) også have at $(P\eta x)((-1)I\eta x) = ((-1)I\eta x)(P\eta x)$. Vi kan hermed bruge (8) idet vi arbejder videre med (9) til at konkluderer (**)

$$\begin{aligned}
\exp(Ax) &= \exp((P\eta x + (-I\eta x))) \\
&\overset{(**)}{=} \exp(P\eta x)\exp((-1)I\eta x) \\
&\equiv \exp(P\eta x)\left(\sum_{j=0}^{\infty} \frac{(-1)^j I^j \eta^j}{j!} x^j\right) \\
&= \exp(P\eta x)\left(\sum_{j=0}^{\infty} \frac{(-1)^j I \eta^j}{j!} x^j\right) \\
&= \exp(P\eta x) I \left(\sum_{j=0}^{\infty} \frac{(-1)^j \eta^j}{j!} x^j\right) \\
&= \exp(P\eta x)\left(\sum_{j=0}^{\infty} \frac{(-\eta)^j}{j!} x^j\right) \\
&= \exp(P\eta x)\exp(-\eta x) \\
&= \exp(-\eta x)\exp(P\eta x) = \exp(-\eta x)\sum_{n=0}^{\infty} \frac{(\eta x)^n}{n!} P^n,
\end{aligned} \tag{10}$$

som ønsket.

Opgave 2.3

Antag

$$e^{-\eta x}\sum_{n=0}^{l}\frac{(\eta x)^n}{n!} > 1-\varepsilon. \tag{11}$$

Vi bemærker til resten af denne opgave at vi har med komposition med objekter i $\mathbb{R}_{\geq 0}$ at gøre, og at den velkendte komplekse ekspotential funktion er overalt konvergent. Vi ser ved brug af (11) at

$$\begin{aligned}
1 = e^{-\eta x}e^{\eta x} &\equiv e^{-\eta x}\sum_{n=0}^{\infty}\frac{(\eta x)^n}{n!} \\
&= e^{-\eta x}\left(\sum_{n=0}^{l}\frac{(\eta x)^n}{n!} + \sum_{n=l+1}^{\infty}\frac{(\eta x)^n}{n!}\right) \\
&= e^{-\eta x}\sum_{n=0}^{l}\frac{(\eta x)^n}{n!} + e^{-\eta x}\sum_{n=l+1}^{\infty}\frac{(\eta x)^n}{n!} \\
&\overset{(11)}{>} 1-\varepsilon+e^{-\eta x}\sum_{n=l+1}^{\infty}\frac{(\eta x)^n}{n!}.
\end{aligned}$$

Hvorfra vi får

$$\begin{aligned}
1 &> 1-\varepsilon+e^{-\eta x}\sum_{n=l+1}^{\infty}\frac{(\eta x)^n}{n!} \\
&\Leftrightarrow \\
0 &> -\varepsilon+e^{-\eta x}\sum_{n=l+1}^{\infty}\frac{(\eta x)^n}{n!} \\
&\Leftrightarrow \\
e^{-\eta x}\sum_{n=l+1}^{\infty}\frac{(\eta x)^n}{n!} &< \varepsilon
\end{aligned} \tag{12}$$

Bemærk for $||\cdot||$ betegnende matrix normen induceret af inf-normen på $\mathbb{R}^n$, at vi ved brug af (12) og omskrivningen i opgave 2.2 kan konkludere

$$\begin{aligned}
\left\|\exp Ax - e^{-\eta x}\sum_{n=0}^{l}\frac{(\eta x)^n}{n!}P^n\right\| &\overset{Opg2.2}{\equiv} \left\|e^{-\eta x}\sum_{n=0}^{\infty}\frac{(\eta x)^n}{n!}P^n - e^{-\eta x}\sum_{n=0}^{l}\frac{(\eta x)^n}{n!}P^n\right\| \\
&= \left\|e^{-\eta x}\left(\sum_{n=0}^{\infty}\frac{(\eta x)^n}{n!}P^n - \sum_{n=0}^{l}\frac{(\eta x)^n}{n!}P^n\right)\right\| \\
&= \left|e^{-\eta x}\right|\cdot\left\|\sum_{n=l+1}^{\infty}\frac{(\eta x)^n}{n!}P^n\right\| \\
&\overset{\Delta}{\leq} e^{-\eta x}\cdot\sum_{n=l+1}^{\infty}\left\|\frac{(\eta x)^n}{n!}P^n\right\| \\
&= e^{-\eta x}\cdot\sum_{n=l+1}^{\infty}\left(\left\|\frac{(\eta x)^n}{n!}\right\|\|P^n\|\right) \\
&= e^{-\eta x}\cdot\sum_{n=l+1}^{\infty}\left(\left\|\frac{(\eta x)^n}{n!}\right\|\|P\|^n\right)
\end{aligned}$$

$$= e^{-\eta x} \cdot \sum_{n=l+1}^{\infty} \left(\frac{(\eta x)^n}{n!} \| P \|^n \right). \tag{13}$$

Vi bemærker at vi i uligheden ved Δ har gjort brug af trekants-uligheden for normer over uendelige summer, hvilket vi kan tillade os (An1), idet $\| \cdot \|$ er en norm, og idet vi ved at den konkrete række kan majoriseres ved brug af rækken som giver $\exp Ax$.
Vi bemærker nu, at vi fra T4.4.2 på side 189 i bogen har, at for den matrix-afledte inf-norm gælder, at normen af en matrix er mindre end den største absolutte rækkesum, at idet vi ved at P er substokastisk, således at den opfylder (4) og (5), vil $\| P \| \leq 1$, således at $\| P \|^n \leq 1$. Vi kan hermed videreføre (13);

$$\begin{aligned} \left\| \exp Ax - e^{-\eta x} \sum_{n=0}^{l} \frac{(\eta x)^n}{n!} P^n \right\| &\leq e^{-\eta x} \cdot \sum_{n=l+1}^{\infty} \left(\frac{(\eta x)^n}{n!} \| P \|^n \right) \\ &\leq e^{-\eta x} \cdot \sum_{n=l+1}^{\infty} \frac{(\eta x)^n}{n!} \\ &\overset{(12)}{<} \varepsilon, \end{aligned}$$

som ønsket.

Opgave 2.4

Se "prog2.py" for implementeringen, og kommentering af programmet. - Der er igen en god del ekstra kommentering og tilbagerapportering inkluderet i programmet. Vi bemærker dog følgende korrespondancer mellem Python variable, og matematiske udtryk;

$$seqtest_n := e^{-\eta x} \frac{\eta^n \cdot x^n}{n!} \tag{14}$$

$$Sseqtest_n := e^{-\eta x} \sum_{k=0}^{n} \frac{\eta^k \cdot x^k}{k!} \tag{15}$$

$$Sseq_n := e^{-\eta x} \sum_{k=0}^{n} \frac{\eta^k \cdot x^k}{k!} P^k, \tag{16}$$

for $k \in \mathbb{N}_0$.
Vi får gennem implementeringen følgende resultater til følgende rapporterede præcision;

$$\exp(Ax) \overset{stop\ ved}{=} S_{29_A} \overset{\text{I Python som}}{=} Sseq_{29} = \begin{pmatrix} 0.36595709 & 0.35453831 & 0.27950457 \\ 0.3652746 & 0.35510048 & 0.27962489 \\ 0.36551523 & 0.35489925 & 0.27958549 \end{pmatrix}.$$

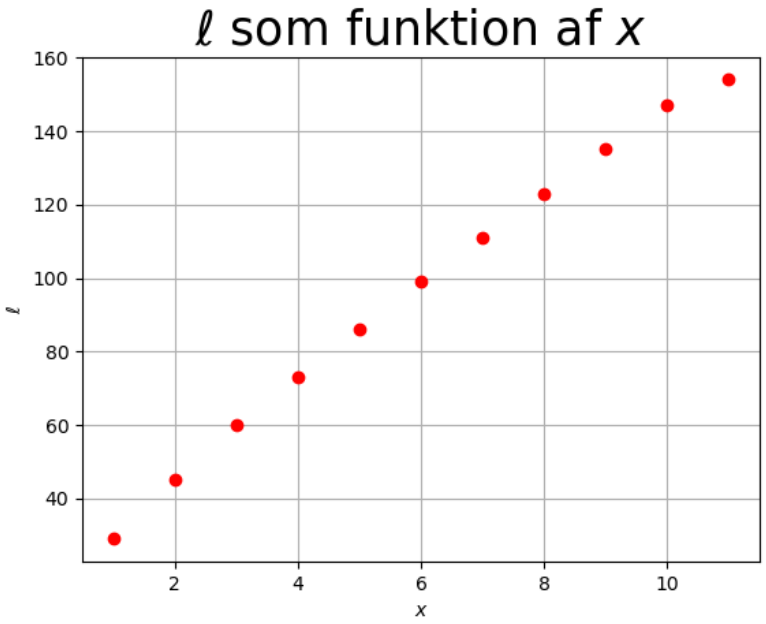
$$\exp\left(\frac{A}{20}x\right) \overset{stop\ ved}{=} S_{7_{\frac{A}{20}}} \overset{\text{I Python som}}{=} Sseq_7 = \begin{pmatrix} 0.79463193 & 0.09063792 & 0.11473012 \\ 0.09154119 & 0.7633136 & 0.14514518 \\ 0.15237132 & 0.18188311 & 0.66574554 \end{pmatrix}.$$

Overstående stop (ved hhv. 29 og 7 i summerne) forekommer på baggrund af en standard sat epsilon tolerance på 10^{-8} for inf-normen af bidraget ved det efterfølgende afsnitssum led jf. ligning 4 i opgaven.

Det kan eventuelt bemærkes at vi i implementeringen regner på teststørrelsen 'seqtest' hvor vi både har η^n og x^n i tælleren. Dette ender med at give problemer med overflow for mellemstore værdier af x (fra $x = 11$ og opefter) da Python regner tælleren ud først, og dermed ved produktet i tællere opnår et tal som har overflow - mere vedrørende, inklusiv en minifiks betragtes i opgave 2.5 nedenfor.

Opgave 2.5 + 2.6

Vi betragter implementeringen lavet i Opgave 2.4 med approksimation af $\exp(Ax)$.
For $x = 1$ blev det sidste led i summen opnået på baggrund af epsilon-kriteriet ledet med $n = 29 =: \ell_1$.
Betragtes større værdier af x ses at for progressivt større værdier af x vil det sidste led i den korresponderende sum ligeledes vokse - Det tager altså flere og flere led før summen konvergerer. Dette kan ses illustreret for $x = 1, \ldots, 11$ i Python grafen nedenfor:



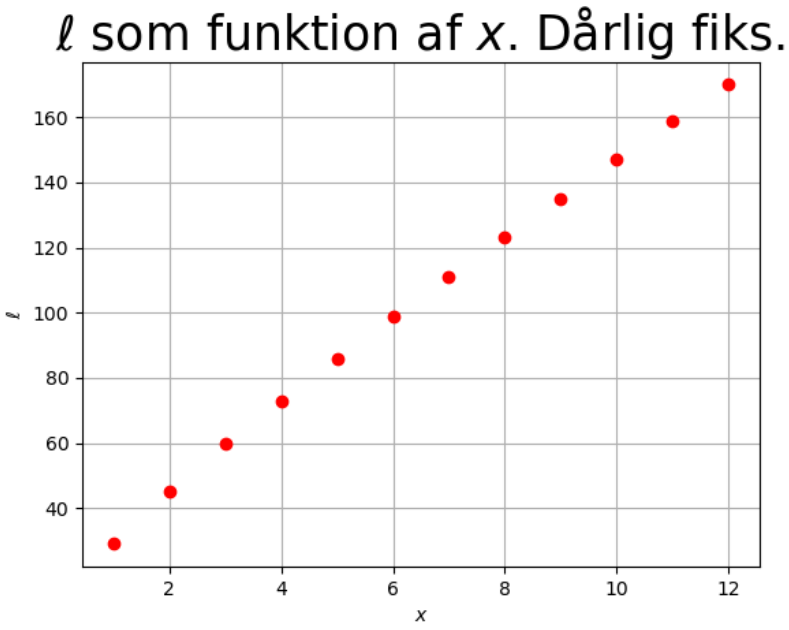
Vi stopper her ved $x = 11$, med $\ell_{11} = 154$ idet allerede dette led faktisk er misvisende. For på trods af at den resulterende matrix passer ganske fint i overensstemmelse med eksempelvis dén som bliver produceret af *expm*-funktionen, stoppes afsnitssummen ikke ved ℓ_{11} på baggrund af epsilon kriteriet, men på baggrund af en eksplosion i *seqtest* fra $seqtest_{154} \overset{\text{printet}}{\underset{\text{som}}{=}} 6.96209847400239 \cdot 10^{-8}$ til $seqtest_{155} \overset{\text{printet}}{\underset{\text{som}}{=}} \text{inf}$, hvilket ikke umiddelbart virker som en naturlig progression hvis man alene betragter det matematiske udtryk.
Ved nærmere undersøgelse viser der sig at være tale om et overflow af tælleren af *seqtest*, altså størrelsen $\eta^n \cdot x^n$, således at $tæller_{seqtest_{154}} \cong 2.37 \cdot 10^{160} \cdot 8.98 \cdot 10^{146} \cong 2.13 \cdot 10^{307}$, mens $tæller_{seqtest_{155}} \cong 2.61 \cdot 10^{161} \cdot 8.08 \cdot 10^{147} \cong 2.11 \cdot 10^{309} \overset{\text{printet}}{\underset{\text{som}}{=}} \text{inf}$.

Det sker på baggrund af, at idet vi regner med 64-bit double precision floating-point tal, har vi en 11-bit "biased exponent" til rådighed. Med double precision kan vi dermed udtrykke tal op til $2 \cdot 2^{1023} = 2^{1024} \cong 1.8 \cdot 10^{308}$, hvorefter vi får overflow. Det ses at da også at $tæller_{seqtest_{154}} \cong 2.13 \cdot 10^{307} < 1.8 \cdot 10^{308} < 2.11 \cdot 10^{309} \cong tæller_{seqtest_{155}}$, således at vi endnu ikke når at opnå overflow i $n = 154$-ledet, hvorfor Python kan bruge tælleren mere konkret i beregningen af resten af $seqtest_{154}$, således at $seqtest_{154} \overset{\text{printet}}{\underset{\text{som}}{=}} 6.96209847400239 \cdot 10^{-8}$, mens $seqtest_{155}$ "overflower" til inf - Python forsøger da også at gøre os opmærksomme på at der sker overflow i udregninen af $seqtest_{155}$, gennem udstedelsen af en fejlbesked. Dette overflow føres da videre i beregningerne, således at $Sseqtest_{155}$ også overflower, hvormed stopkriteriet sætter ind idet "$Sseqtest_{155} = \text{inf} > 1 - \varepsilon$", og programmet dermed stopper og returnerer $Sseq_{154}$.
Et umiddelbart forekommende, dog kun midlertidigt og ikke særligt robust idé til et fiks før man eksempelvis når til "scaling-and-squaring", er at rykke rundt på *seqtest*-udtrykket, således at man ikke evaluerer produktet af det i nabolaget af $x = 11$ store tal $\eta^n \cdot x^n$, men i stedet ladet et mindre udtryk som ikke laver overflow

blive "tæmmet" af $\frac{1}{n!}$. Vi kan eksempelvis betragte, at hvis vi i Python regner *seqtest* som

$$seqtest_n = \eta^n \frac{e^{-\eta x} \cdot x^n}{n!},$$

således vil opnå nedenstående $\ell - x$-figur, hvor $Sseq_\ell$, igen, passer forholdsvis godt med "expm", og hvor vi allerede for $x = 12$ stopper på grund af for store tal. Denne gang ikke på grund af tælleren, men idet konvergensen for $x = 12$, med implementeringen sker så langsomt at vi skal ud over $n = 171$, således at $n!$ i nævneren når at vokse sig større end end de $\cong 1.8 \cdot 10^{308}$. Da dette tal ikke kan indpasses i FP64, som *seqtest* regnes i, smider Python en fejlbesked, og $Sseq_n$ stopper derfor ved $n = 170$.



Disskusionen og graferne ovenfor giver os et hint om hvorfor "scaling squaring" kan være praktisk. Graferne lader til at angive proportionel korrelation mellem størrelsen af x idet vi bruger metoden udviklet i Opgave 2.4, antallet af led krævet før vi når vores epsilon-kriterie. Hvis korrelationen holder til kausalitet (evt. gennem confoundere) er det helt sikkert en strategi at forsøge at finde en måde at få samme resultat mens størrelsen af koefficienten der ganges på A reduceres, for dermed at undgå problemerne med "store x" som beskrevet ovenfor.
Vi ser da også gennem vores implementering i "prog3.py" at det tilføjer en betydelig robusthed til programmet at betragte $\left(\exp A\frac{x}{m}\right)^m$ gennem (10) i implementeringen af løsningen i Opgave 2.4, i stedet for at betragte $\exp Ax$ gennem (10), idet vi eksempelvis uden problemer problemer kan betragte problemet med $x = 100$.
Helt præcist ser vi for A som defineret som i opgaveformuleringens ligning (2) at

$$\exp\left(A\frac{x}{m}\right)^m \overset{stop\ ved}{=} S_{33_{x=1}} = \begin{pmatrix} 0.3659571 & 0.35453832 & 0.27950458 \\ 0.36527461 & 0.35510049 & 0.27962489 \\ 0.36551524 & 0.35489926 & 0.27958549 \end{pmatrix}.$$

$$\exp\left(A\frac{x}{m}\right)^m \overset{stop\ ved}{=} S_{37_{x=10}} = \begin{pmatrix} 0.3655914 & 0.35483871 & 0.27956989 \\ 0.3655914 & 0.35483871 & 0.27956989 \\ 0.3655914 & 0.35483871 & 0.27956989 \end{pmatrix}.$$

$$\exp\left(A\frac{x}{m}\right)^m \overset{stop\ ved}{=} S_{43_{x=100}} = \begin{pmatrix} 0.36559139 & 0.35483871 & 0.27956989 \\ 0.36559139 & 0.35483871 & 0.27956989 \\ 0.36559139 & 0.35483871 & 0.27956989 \end{pmatrix}.$$

Resultaterne viser sig igen at være i rigtig god overensstemmelse med 'expm'-funktionen fra scipy.linalg. Implementeringen af prog3 forekommer gennem brug af prog2 og stopkriteriet er dermed "det samme". Bemærk dertil også at vi i implementeringen har løst uligheden $1 < 2^N < 2$, for N og på basis heraf afklaret at vi ved valg af $m = 2^{\left(\lfloor \frac{\log x}{\log 2} \rfloor\right)}$, kan begrænse $y := \frac{x}{m} \in (1, 2]$.

Vi har nu tre implementeringer af at regne expotentialmatricer på basis af subintensitetsmatricer. - Af disse lader prog3 til at være den klar mest robuste, da den tillader betydeligt større valg af x uden at det går galt i maskineriet (prog1 er betydeligt værre end prog2 til at håndtere store værdier af x). Så da prog3 lader til at kunne løse de samme problemer som blev kastet efter prog2 og prog1, til nogenlunde samme præcision og med mere robusthed tænker jeg at denne implementation vil være at foretrække.

Opgave 3

Opgave 3.1

Bemærk at vi fra lineær algebra for $w \in \mathbb{C}$ og $R \equiv \{r_{ij}\}_{i,j=1,\ldots,n} \in M_{n\times n}(\mathbb{C})$, at

$$w \cdot R \equiv R \cdot w = \{w \cdot r_{ij}\}_{i,j=1,\ldots,n} \tag{17}$$

Det bemærkes dertil at vi fra lineær algebra har resultatet, at for

$$Q = \begin{pmatrix} q_1 & & \\ & \ddots & \\ & & q_n \end{pmatrix} \equiv diag(q_1, \ldots, q_n)$$

værende en $n \times n-$diagonalmatrix, vil

$$Q^k \equiv diag(q_1, \ldots, q_n)^k = diag\left(q_1^k, \ldots, q_n^k\right). \tag{18}$$

Betragt derfor, ud fra definitionen af D at;

$$\begin{aligned}
\exp(Dx) &\equiv \sum_{k=0}^{\infty} \frac{D^k}{k!} x^k \qquad (19)\\
&\equiv \sum_{k=0}^{\infty} \frac{diag(\lambda_1, \ldots, \lambda_n)^k}{k!} x^k \\
&\overset{(18)}{=} \sum_{k=0}^{\infty} \frac{diag\left(\lambda_1^k, \ldots, \lambda_n^k\right)}{k!} x^k \\
&\overset{(17)}{=} \sum_{k=0}^{\infty} diag\left(\frac{\lambda_1^k}{k!}x^k, \ldots, \frac{\lambda_n^k}{k!}x^k\right) \\
&= diag\left(\sum_{k=0}^{\infty} \frac{\lambda_1^k}{k!}x^k, \ldots, \sum_{k=0}^{\infty} \frac{\lambda_n^k}{k!}x^k\right) \\
&= diag\left(e^{\lambda_1 x}, \ldots, e^{\lambda_n x}\right), \qquad (20)
\end{aligned}$$

som ønsket.

Opgave 3.2

Bemærk først som nævnt i opgaven at idet A er diagonaliserbar vil den kunne opskrives på formen $A \equiv VDV^{-1}$. Dette er netop smart, fordi det tillader os at regne potenser af A gennem potenser af den diagonale matrix D, som det nævnt i Opgave 3.1 er betydeligt nemmere at regne potenser for. Helt konkret har vi fra LinAlg at

$$\begin{aligned} A^k \equiv \left(VDV^{-1}\right)^k &\equiv \underbrace{\left(VDV^{-1}\right) \cdot \left(VDV^{-1}\right) \cdot \ldots \cdot \left(VDV^{-1}\right)}_{\text{k ganger}} \\ &\equiv \left(VDV^{-1}VDV^{-1} \cdot \ldots \cdot VDV^{-1}\right) \\ &= VD^kV^{-1}. \end{aligned} \tag{21}$$

Vi kan derfor betragte at

$$\begin{aligned} \exp(Ax) &= \exp\left(VDV^{-1}x\right) \\ &\equiv \sum_{k=0}^{\infty} \frac{\left(VDV^{-1}\right)^k}{k!} x^k \\ &\overset{(21)}{\equiv} \sum_{k=0}^{\infty} \frac{VD^kV^{-1}}{k!} x^k. \end{aligned}$$

Da denne række er lig $\exp(Ax)$ og dermed konvergent, kan vi idet V, V^{-1} uafhængige konstante matricer tillade os at flytte dem udenfor matrix-summen, som venstre hhv. højre multiplikanter, således at

$$\begin{aligned} \exp(Ax) &= \sum_{k=0}^{\infty} \frac{VD^kV^{-1}}{k!} x^k \\ &= V \sum_{k=0}^{\infty} \left(\frac{D^k}{k!} x^k\right) V^{-1} \\ &\overset{(19)}{=} V \exp(Dx) V^{-1} \overset{(20)}{=} V diag\left(e^{\lambda_1 x}, \ldots, e^{\lambda_n x}\right) V^{-1}, \end{aligned} \tag{22}$$

som ønsket.

Opgave 3.3

Se "prog4.py" for implementeringen, og kommentering af programmet. Det er værd at bemærke at den konkrete efterspurgte funktion først forekommer nederst i Python filen som funktionen "expomat4".
Ved hjælp af voreshjælpefunktioner "matexpmforskel" og "infmatnorm" som brugt i prog1, betragter vi følgende forskelle i inf-normen mellem implementeringen i prog3 og den indbyggede expm hhv. prog4 og expm, for de efterspurgte værdier af x;

$$\begin{aligned} \| expomat3 - expm \|_{(x=1)} &= 2.7654182999103938e-08 \\ \| expomat4 - expm \|_{(x=1)} &= 3.1086244689504383e-15 \end{aligned}$$

$$\begin{aligned} \| expomat3 - expm \|_{(x=10)} &= 8.991250111112237e-08 \\ \| expomat4 - expm \|_{(x=10)} &= 1.7708057242771247e-14 \end{aligned}$$

$$\| expomat3 - expm \|_{(x=100)} = 7.754297443596236e-07$$

$$\| expomat4 - expm \|_{(x=100)} = 2.349231920106831e - 13.$$

Den interne differens bliver;

$$\| expomat3 - expomat4 \|_{(x=1)} = 2.7654185941194953e - 08$$
$$\| expomat3 - expomat4 \|_{(x=10)} = 8.991251865264616e - 08$$
$$\| expomat3 - expomat4 \|_{(x=100)} = 7.754299792273045e - 07.$$

Det lader altså til, at idet vi betragter expomat3 med et epsilon på $\varepsilon = 10^{-8}$, vil differensen mellem resultatet af expomat4 og expm i inf-normen, være betydeligt lavere end ved brug af expomat3.
Ligeledes betragter vi gennem brug af "time"-biblioteket følgende rapporterede køretider:

$$Tidsforskel(expomat3_{(x=1)}) = 0.0009992122650146484$$
$$Tidsforskel(expomat4_{(x=1)}) = 0.0009999275207519531$$

$$Tidsforskel(expomat3_{(x=10)}) = 0.00099945068359375$$
$$Tidsforskel(expomat4_{(x=10)}) = 0.0010001659393310547$$

$$Tidsforskel(expomat3_{(x=100)}) = 0.0009999275207519531$$
$$Tidsforskel(expomat4_{(x=100)}) = 0.0010013580322265625,$$

dog der ofte blev rapporteret "0.0" tilbage.

Opgave 3.4

Man kan selvfølgeligt byde, at det at øge størrelsen af de betragtede matricer først og fremmest vil teste robustheden af implementeringerne som er lavet. I forhold til maskineriet bagved hver af prog3 og prog4 kan man eventuelt bemærke at prog3 benytter sig af de i opgave 2 - definerede eta'er som er defineret til at være maximum af absolutværdierne af de diagonale indgange. Da disse diagonale indgange skal være nummerisk større end absolutværdien af summen af i rækken øvrige elementer, vil det at øge størrelsen på matricen, tvinge eta større, hvilket eventuelt vil kunne øge antallet af nødvendige summeled, og eventuelt derigennem lede til overflow i prog3.
Man kan også overveje, at det ikke er helt billigt at inverterer en matrix $\mathcal{O}(\frac{4}{3}n^3)$, hvilket sagtens kan komme til at gå ud over hastigheden som størrelsen vokser.
Tilsidst kan man også spekulere i hvorvidt det at implementeringen af prog4 er lavet med funktioner som kommer udefra, og som sandsynligvis er optimeret betydeligt bedre, end hvad vi som studerende kan mostre, og har erfaring til, kunne gøre en forskel. Dette vil sandsynligvis være tilfældet i forhold til præcision, da Python kan gå helt tilbage og regne præcist på eigenvektorer, og eigenværdier af de givne matricer.