

Prosjekt 1

Henrik Modahl Breitenstein and Carl Petter Duedahl
(Dated: September 13, 2021)

<https://github.com/henrikbreitenstein/FYS3150.git>

PROBLEM 1

Poisson likningen

$$-\frac{d^2 u}{dx^2} = f(x)$$

Bytter $f(x)$ med gitt funksjon

$$\begin{aligned} -\frac{d^2 u}{dx^2} &= 100e^{-10x} \\ -d^2 u &= 100e^{-10x} dx^2 \end{aligned}$$

Tar integralene

$$\begin{aligned} -\iint d^2 u &= \iint 100e^{-10x} dx^2 \\ -u &= \int -10e^{-10x} + c_1 dx \\ -u &= e^{-10x} + c_1 x + c_2 \\ u &= -e^{-10x} - c_1 x - c_2 \end{aligned}$$

Bruker initialbetingelsene

$$u(0) = 0 \Rightarrow -1 - c_2 = 0 \quad (1)$$

$$u(1) = 0 \Rightarrow -e^{-10} - c_1 - c_2 = 0 \quad (2)$$

Med **1** og **2** får vi:

$$\begin{aligned} c_2 &= -1 \\ c_1 &= 1 - e^{-10} \end{aligned}$$

Ved å sette inn for c_1 og c_2 får vi:

$$u = 1 - (1 - e^{-10})x - e^{-10x} \quad (3)$$

PROBLEM 2

I repository'et under FYS3150/Project1/main.cpp så har vi skrevet koden som regner ut verdiene til den eksakte løsningen, og i FYS3150/Project1/plot.py så tegnes grafen. Kan kjøre programmet 'main.cpp' med commandoen

```
$ make all
og kjøre 'plot.py' med
```

```
$ python plot.py
Og vi får plottet 1:
```

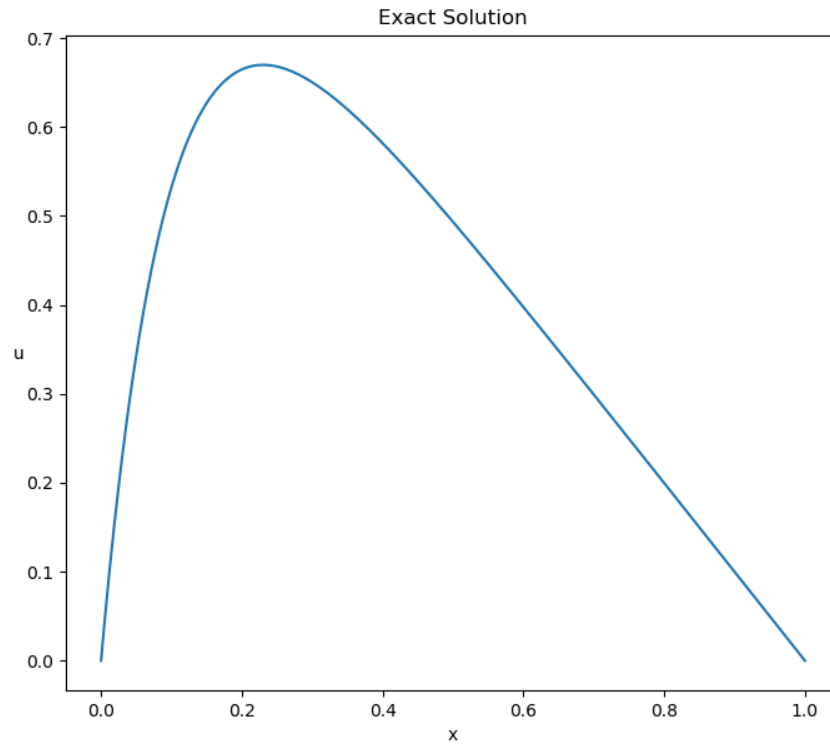


FIG. 1. Eksakte løsning $u(x)$

PROBLEM 3

Vi starter med Poisson likningen:

$$-\frac{d^2 u(x)}{dx^2} = f(x)$$

Vi har så via Taylor ekspansjon at:

$$u(x+h) = u(x) + u'(x)h + \frac{1}{2}u''(x)h^2 + \frac{1}{6}u'''(x)h^3 + O(h^4)$$

$$u(x-h) = u(x) - u'(x)h + \frac{1}{2}u''(x)h^2 - \frac{1}{6}u'''(x)h^3 + O(h^4)$$

Og finner så summen:

$$u(x+h) + u(x-h) = 2u(x) + f''(x)h^2 + O(h^4)$$

Som gir oss

$$u''(x) = \frac{u(x+h) - 2u(x) + u(x-h)}{h^2} + O(h^2)$$

Om vi så fjerner error-leddet:

$$v''(x) = \frac{v(x+h) - 2v(x) + v(x-h)}{h^2}$$

Som vi kan bruke til å skrive Poisson likningen:

$$-\frac{v(x+h) - 2v(x) + v(x-h)}{h^2} = f(x)$$

PROBLEM 4

Om vi skriver det om som:

$$-v(x+h) + 2v(x) - v(x-h) = h^2 f(x)$$

Om vi har x-verdier:

$$\vec{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

Om vi så setter verdiene til v inne i:

$$\vec{v} = \begin{bmatrix} v(x_1) \\ v(x_2) \\ \vdots \\ v(x_n) \end{bmatrix}$$

Og det samme med verdiene til $f(x)$:

$$\vec{g} = \begin{bmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_n) \end{bmatrix}$$

Så skriver vi likningen om:

$$-v_{i+1} + 2v_i - v_{i-1} = h^2 g_i$$

Som kan skrives:

$$\begin{bmatrix} -1 \\ 2 \\ -1 \end{bmatrix} \begin{bmatrix} v_{i+1} \\ v_i \\ v_{i-1} \end{bmatrix} = h^2 g_i$$

Som vil gjelde for alle i :

$$\mathbf{A}\vec{v} = h^2 \vec{g}$$

Hvor \mathbf{A} må ha $\begin{bmatrix} -1 \\ 2 \\ -1 \end{bmatrix}$ forskøvet for hver rad slik at de henger sammen med de riktige verdiene til $v(x)$ i \vec{v} .

PROBLEM 5

Problem a

Siden vi gjør en matrismultiplikasjon så må bredden, n , til \mathbf{A} være det samme som lengden m til \vec{v} og \vec{g} . Så

$$n = m$$

Problem b

Av $A\vec{v} = \vec{g}$ vil vi finne alle verdier mellom grensebetingelsene $u(0) = u(1) = 0$. \vec{v}^* er den samme som \vec{v} men vi har lagt til v_0 og v_{n+1} som er grensebetingelsene.

PROBLEM 6

PROBLEM A

Vi har nå en vektor

$$\vec{v} = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{pmatrix}$$

og en $n \times n$ -matrise

$$\mathbf{A} = \begin{pmatrix} b_1 & c_2 & 0 & 0 & \cdots & 0 & 0 \\ a_2 & b_2 & c_2 & 0 & \cdots & 0 & 0 \\ 0 & a_3 & b_3 & c_3 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & \cdots & \cdots & \cdots & a_n & b_n \end{pmatrix}$$

Matrisemultipliserer vi disse får vi

$$\begin{aligned} \mathbf{A}\vec{v} = & \begin{aligned} (I) & \quad b_1 v_1 + c_1 v_2 & = g_1 \\ (II) & \quad a_2 v_1 + b_2 v_2 + c_3 v_3 & = g_2 \\ (III) & \quad \quad a_3 v_2 + b_3 v_3 + c_3 v_4 & = g_3 \\ & \quad \quad \quad \vdots \\ (n) & \quad \quad \quad a_n v_{n-1} + b_n v_n & = g_n \end{aligned} \end{aligned}$$

$$(III)^* \quad 0 \quad 0 \quad b_3 - \frac{c_2 \cdot a_3}{b_2^*} \quad c_3 \quad g_3 - g_1 \frac{a_3}{b_2^*}$$

og vi kan igjen definere $b_3^* = b_3 - \frac{c_2 \cdot a_3}{b_2^*}$ og $g_3^* = g_3 - g_1 \frac{a_3}{b_2^*}$. Og vi kan da fortsette med dette nedover som $(k) - \frac{a_k}{b_{k-1}^*} (k-1)^*$. Så har vi fjernet a -ene så da må vi fjerne c -ene og normalisere b -ene. For å gjøre dette starter vi i siste ledd med

$$(n)/b_n$$

så

$$b_n = \frac{b_n}{b_n}$$

og

$$g_n^* = \frac{g_n}{b_n}$$

Da får vi at $A_n v_n = g_n$ blir satt ned til

$$v_n = \frac{g_n}{b_n}$$

Så kan vi se på leddet $n-1$ her har vi både en b_{n-1} og en c_{n-1} . Vi må derfor gjøre

$$(n-1) = ((n-1) - (n) \cdot c_{n-1})/b_{n-1}$$

så vi også får fjernet c_{n-1} og står kun igjen med et ettall i A matrisen som gir oss at

$$v_{n-1} = \frac{g_{n-1} - g_n \cdot c_{n-1}}{b_{n-1}}$$

og herfra kan vi generalisere til

$$v_i = \frac{g_i - v_{i+1} \cdot c_n}{b_{n-1}}$$

Så da får vi algoritmen

Algorithm 1 Radredusering av tridiagonal matrise

for $i = 2, 3, \dots, n$ **do**

▷ Forward substitution, $n-1$ repetisjoner

$$t \leftarrow \frac{a_i}{b_{i-1}}$$

▷ 1 FLOP

$$b_i \leftarrow b_i - c_{i-1} \cdot t$$

▷ 2 FLOPs

$$g_i \leftarrow g_i - g_{i-1} t$$

▷ 2 FLOPs

▷ Til sammen $7 \cdot (n-1)$ FLOPs i loopen

$$v_n = g_n / b_n$$

▷ 1 FLOP

for $j = n-1, n-2, \dots, 1$ **do**

▷ Backward Substitution, $n-1$ repetisjoner

$$v_j \leftarrow (g_j - v_{j+1} \cdot c_j) / b_j$$

▷ 3 FLOPs

▷ Til sammen $3 \cdot (n-1)$ FLOPs i loopen

Problem b

Så antall FLOPs i loopen blir til sammen

$$7(n-1) + 1 + 3(n-1) = \underline{\underline{10(n-1) + 1}}$$

PROBLEM 7

Problem a

Sriptet 'Problem7new.cpp' og 'Problem7func.cpp' bruker den generelle algoritmen til å løse matriselikningen. For å kjøre de sammen:

```
$ make pr7all
```

Problem b

Vi kjørte for $n = 10$ i 3, $n = 100$ i 3 og $n = 1000$ i 4.

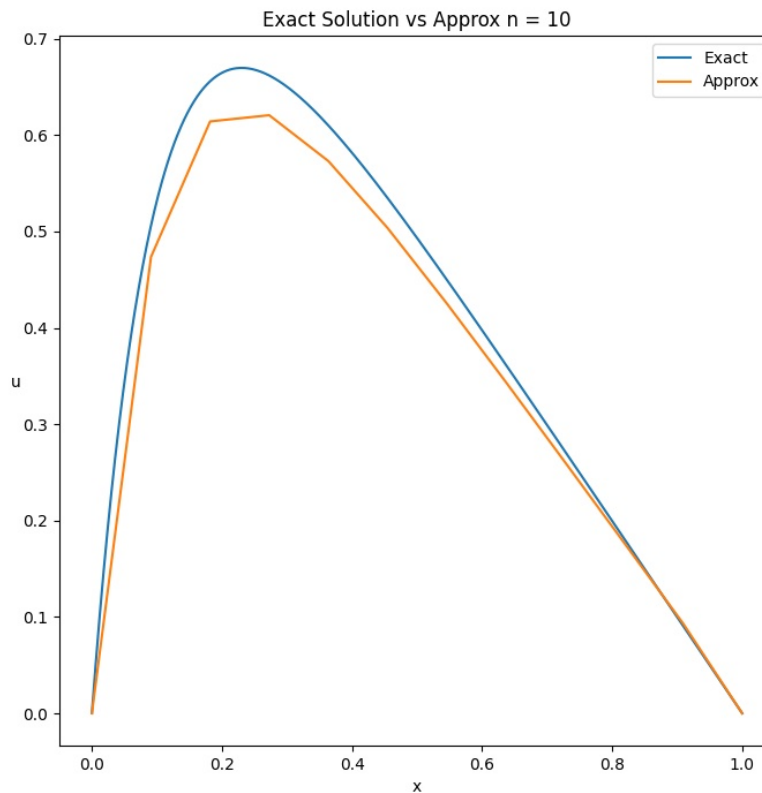


FIG. 2. Eksakt $u(x)$ og tilnærmingen $v(x)$ hvor vi har brukt $n = 10$ som antall steg.

Legger merker til at etter $n = 100$ så ser man ikke den eksakte grafen lenger, og det er vanskelig å si med det blåtte øyet om tilnærmingen blir bedre eller ikke.

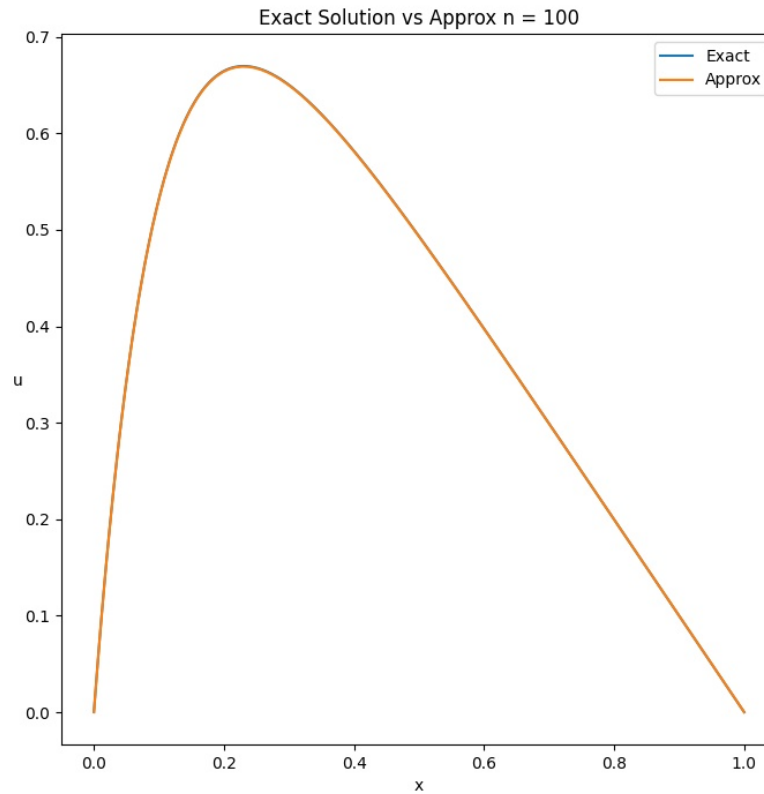


FIG. 3. Eksakt $u(x)$ og tilnærmingen $v(x)$ hvor vi har brukt $n = 100$ som antall steg.

PROBLEM 8

Problem a

Vi fant fant den absolutte feilen mens vi plottet den tilnærmede og eksakte løsningen og har illustrert den absolutte feilen mellom den eksakte verdien og vår tilnærming i 5, 6 og 7.

problem b

Slik som i 8a) fant vi også den relative feilen mens vi plottet den tilnærmede og eksakte mot hverandre og denne finnes da for $n = 10$ i 8, for $n = 100$ i 9 og for $n = 1000$ i 10.

Problem c

Vi plotter den relative maxfeilen for forskjellige n som sett i 11.

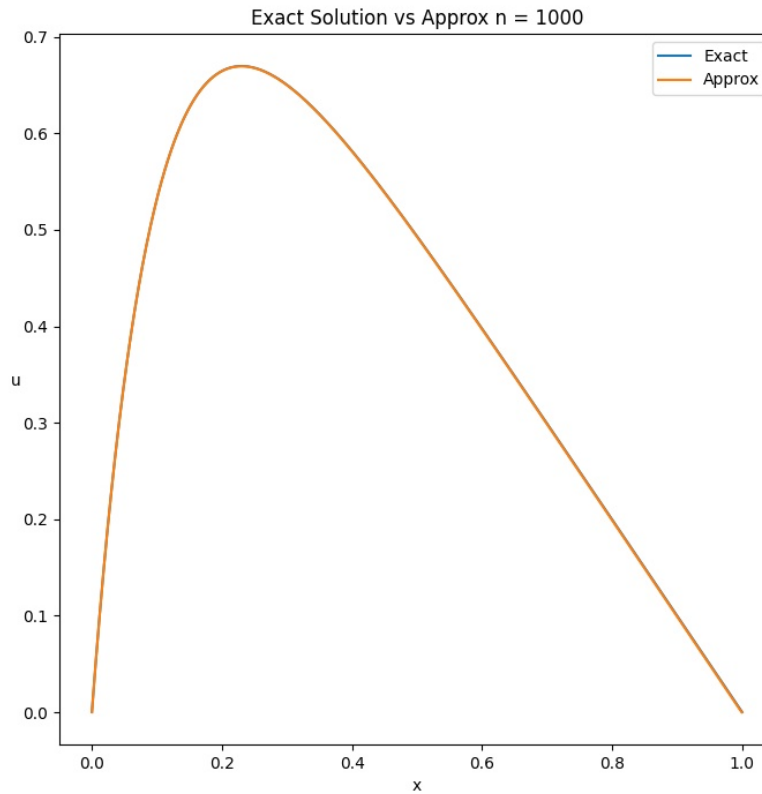


FIG. 4. Eksakt $u(x)$ og tilnærmingen $v(x)$ hvor vi har brukt $n = 1000$ som antall steg.

PROBLEM 9

Problem a

Nå som vi har en spesiell logaritme kan vi se på hvordan \tilde{b}_i utvikler seg

$$\tilde{b}_2 = b_2 - c_1 \frac{a_2}{b_1} = 2 - \frac{1}{2} = \frac{3}{2}$$

$$\tilde{b}_3 = b_3 - c_2 \frac{a_3}{\tilde{b}_2} = 2 - \frac{1}{\frac{3}{2}} = 2 - \frac{2}{3} = \frac{4}{3}$$

og fortsetter vi videre ser vi at det følger mønsteret:

$$\tilde{b}_i = \frac{i+1}{i}$$

og vi husker fra den generelle løsnningen at

$$\tilde{g}_i = g_i - \frac{\tilde{g}_{i-1}}{b_i}$$

så vi får istedenfor at

$$\tilde{g}_i = g_i + \tilde{g}_i \frac{i}{i+1}$$

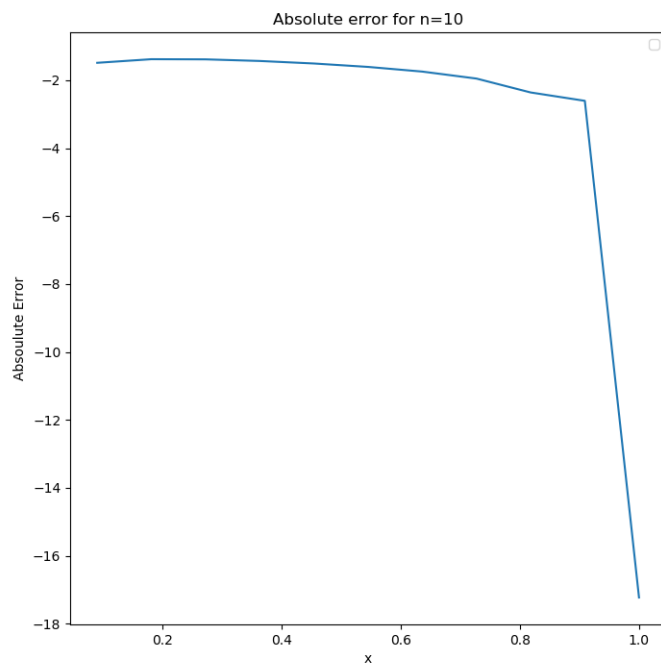


FIG. 5. Plot for den absolutte verdien for $n=10$, 100 og 1000

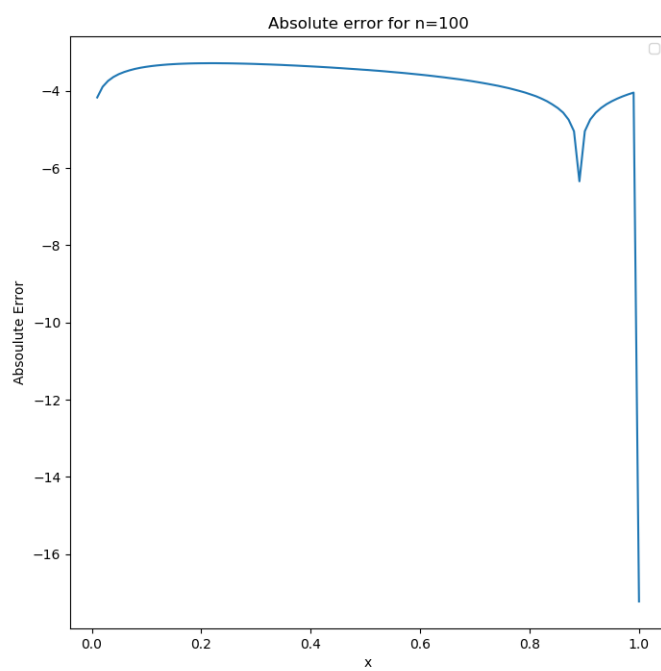


FIG. 6. plot for den absolutte verdien for $n=100$

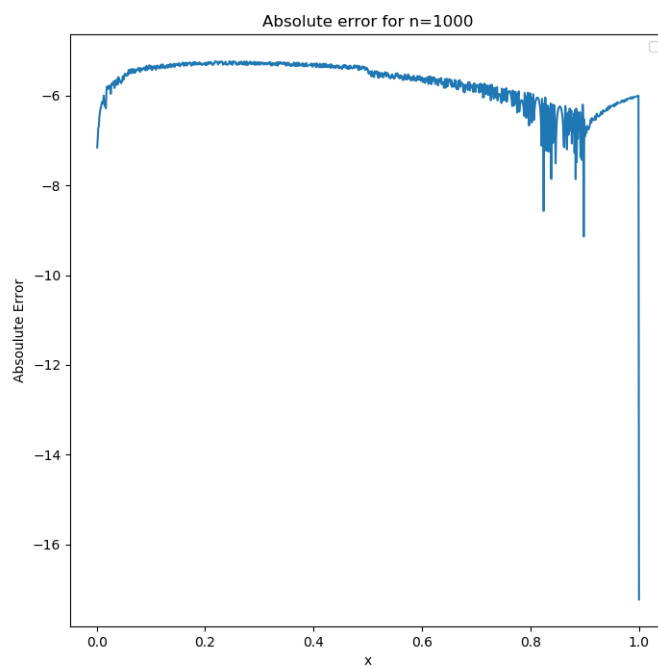


FIG. 7. plot for den absolutte verdien for $n=1000$

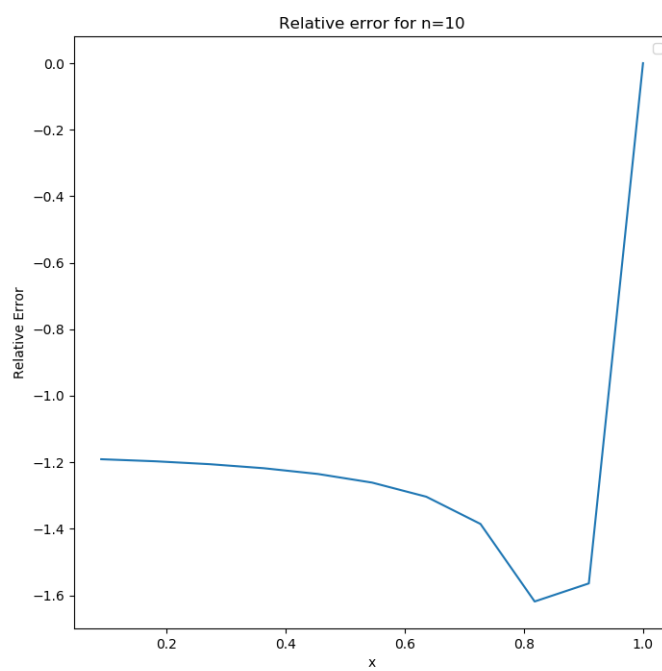


FIG. 8. Plot for den absolutte verdien for $n=10$, 100 og 1000

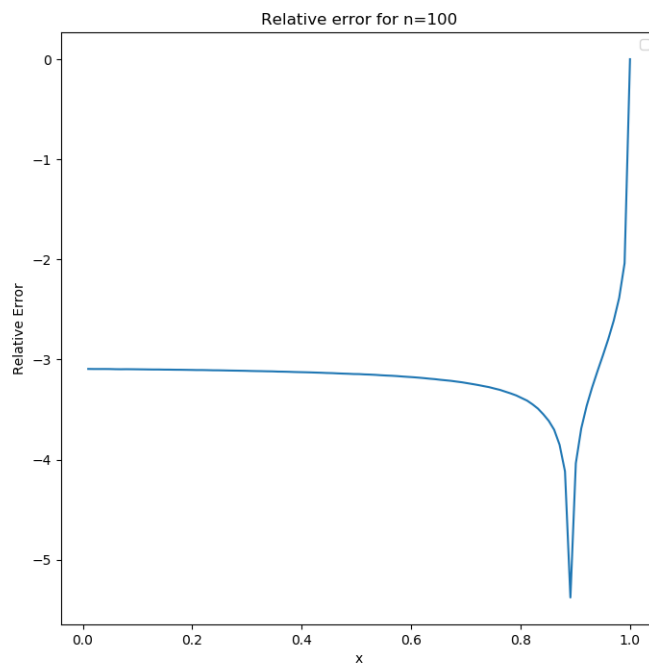


FIG. 9. plot for den absolutte verdien for n=100

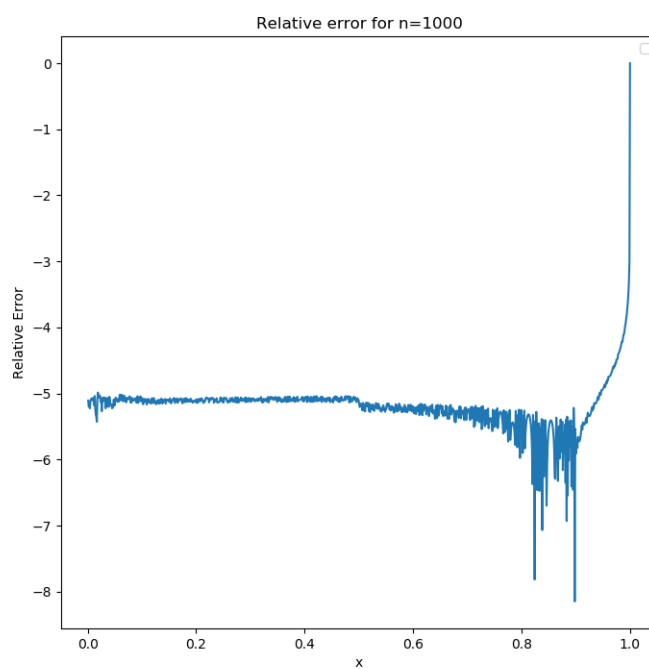
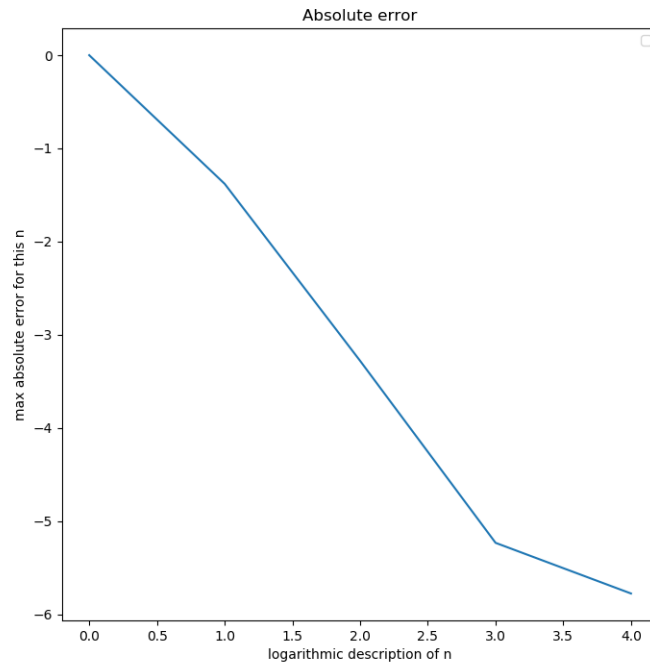


FIG. 10. plot for den absolutte verdien for n=1000

FIG. 11. Den realtive maksfeilen for forskjellige n .

Så husker vi at

$$v_i = (g_i + v_{i+1})/b_i$$

så vi får

$$v_i = (g_i - v_{i+1}) \cdot \frac{i}{i+1}$$

Så lager vi en vektor $i1$ som vil bestå av b_i^{-1} -verdiene hvor

$$i1(i) = \frac{i}{i+1}$$

. Da får vi at løsningsalgoritmen er:

Algorithm 2 Spesialisert algoritme

for $i = 1, 2, \dots, n$ **do**

▷ Forward elimination, $n - 1$ repetisjoner

$$\tilde{g}_i \leftarrow g_i - g_i \cdot i1_i$$

▷ 3 FLOPs

▷ Til sammen $3(n - 1)$ repetisjoner

$$v_n = g_n * i2(n)$$

▷ 1 FLOP

for $i = n - 1, n - 2, \dots, 0$ **do**

▷ Backward elimination, $n - 1$ repetisjoner

$$v_i \leftarrow (g_i + v_{i+1}) \cdot i1_{i+1}$$

▷ 3 FLOPs
▷ Til sammen $3(n - 1)$ repetisjoner

problem b

Vi ser av tellinga over at antall FLOPs vi får er

$$3(n-1) + 1 + 3(n-1) = \underline{\underline{6(n-1) + 1}}$$

som er en god del mindre enn vi fikk i den generelle.

Problem c

I filen 'problem9.cpp' har vi kodet den spesielle algoritmen. Kjøres ved kommandoen

\$ make p9all

PROBLEM 10

Kjører både den generelle og spesifikke algoritmen 100 ganger for hver n og tar så gjennomsnittet. Resultaten er vist i plottet 12.

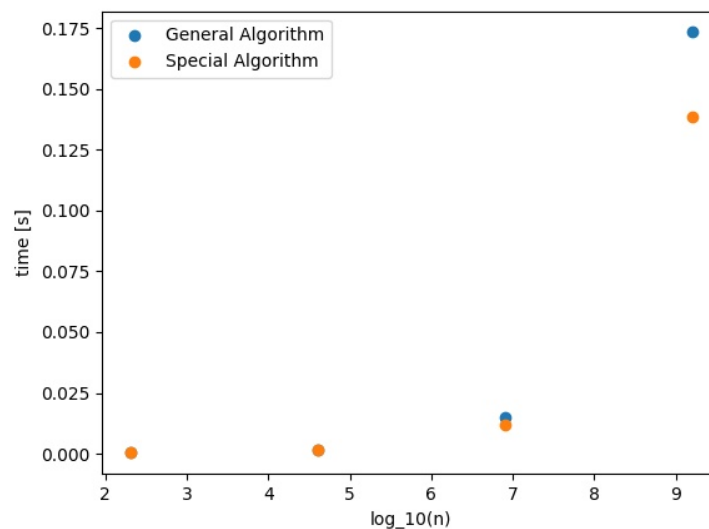


FIG. 12. Tiden brukt til både den genrelle og spesifikke agoritmen for forskjellige n .

Kan se at ved små n så er tidsbruken nesten helt lik, men når man kommer opp i større n så bruker den spesielle merkbart mindre tid enn den generelle.

PROBLEM 11

Med LU dekomposisjon så vil man bruke i utgangspunktet N^3 FLOPs kun for dekomposisjonen og så skalerer kompleksiteten til å løse hver enkelt $\mathbf{A}\vec{v} = \vec{g}$ likning med N^2 . For kun én slik likningen kan vi se at både den generelle og spesielle algoritmen når man har en tridiagonal matrise skalerer to ordner lavere enn LU dekomposisjon.