Title of the document

Henrik Modahl Breitenstein and Carl Petter Duedahl
(Dated: September 12, 2021)

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

PROBLEM 1

Poisson likningen

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

Bytter $f(x)$ med gitt funksjon

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

Tar integralene

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

Bruker initialbetingelsene

$$u(0) = 0 \Rightarrow -1 - c_2 = 0 \tag{1}$$

$$u(1) = 0 \Rightarrow -e^{-10} - c1 - c2 = 0 \tag{2}$$

Med 1 og 2 får vi:

$$\begin{aligned} c2 =& -1 \\ c1 =& 1 - e^{-10} \end{aligned}$$

Ved å sette inn for c_1 og c_2 får vi:

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

PROBLEM 2

I repostory'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

PROBLEM 5

Problem a

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

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

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

Matrisemultipliserer vi disse får vi

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

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

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 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 $i = 2, 3, ..., n$ **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 $j = n-1, n-2, ..., 1$ **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

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

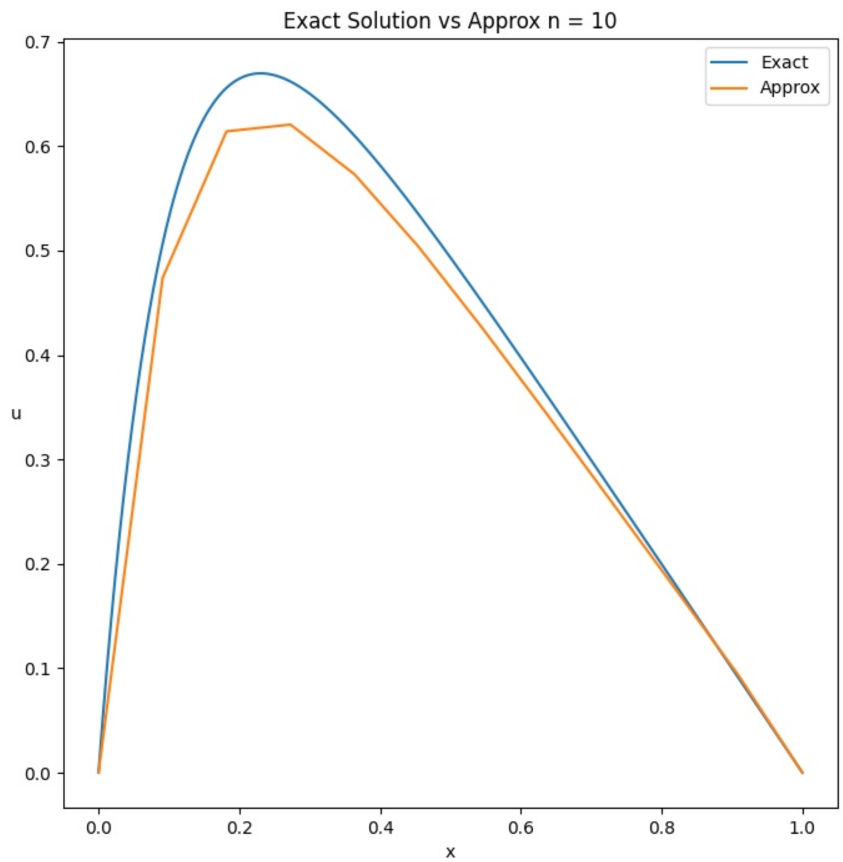

FIG. 1. Eksakt $u(x)$ og tilnærmingen $v(x)$ hvor vi har brukt $n = 10$ som antall steg.

Problem b

Vi kjørte for $n = 10$ i 2, $n = 100$ i 2 og $n = 1000$ i 3.

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.

PROBLEM 8

Problem a

Vi har illustrert den absolutte feilen mellom den eksakte verdien og vår tilnærming i 4.

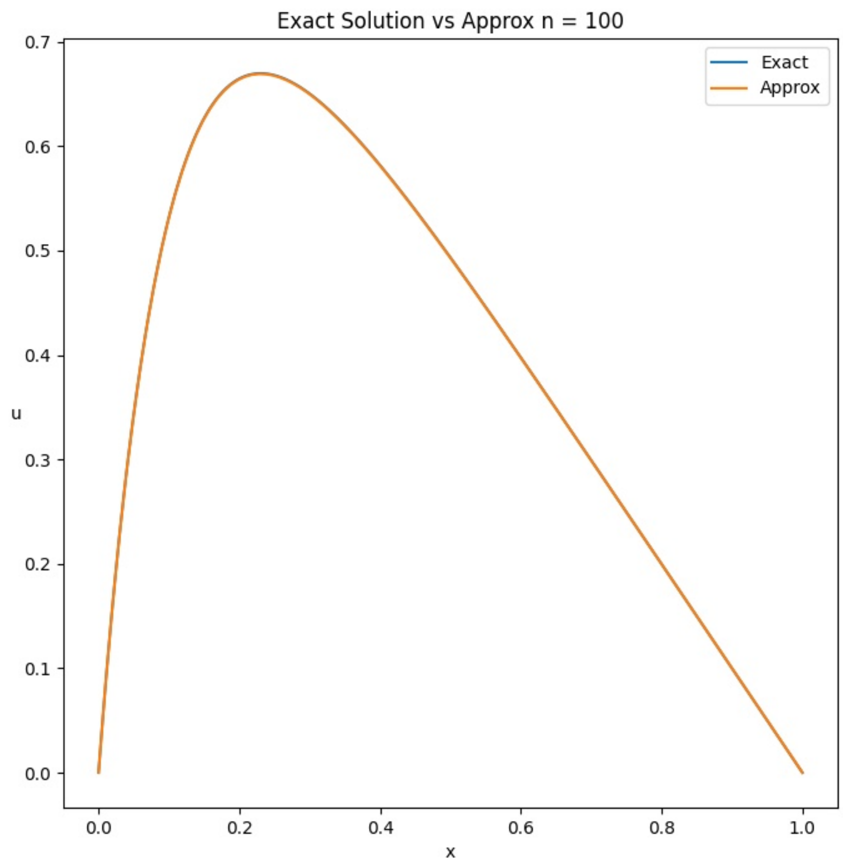

FIG. 2. Eksakt $u(x)$ og tilnærmingen $v(x)$ hvor vi har brukt $n = 100$ som antall steg.

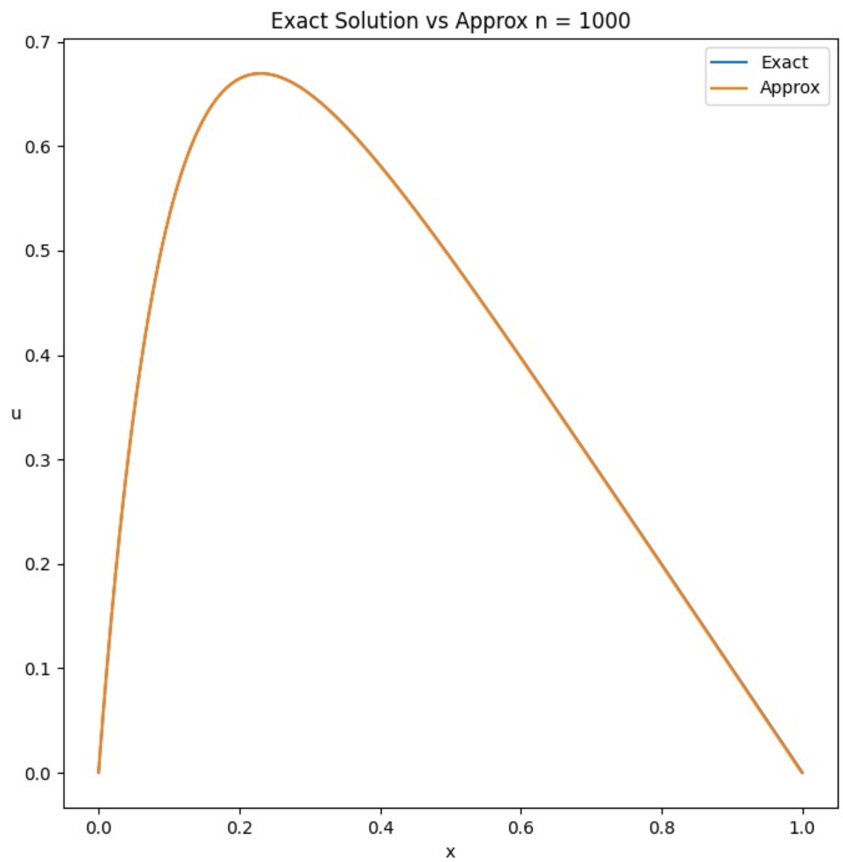

FIG. 3. Eksakt $u(x)$ og tilnærmingen $v(x)$ hvor vi har brukt $n = 1000$ som antall steg.

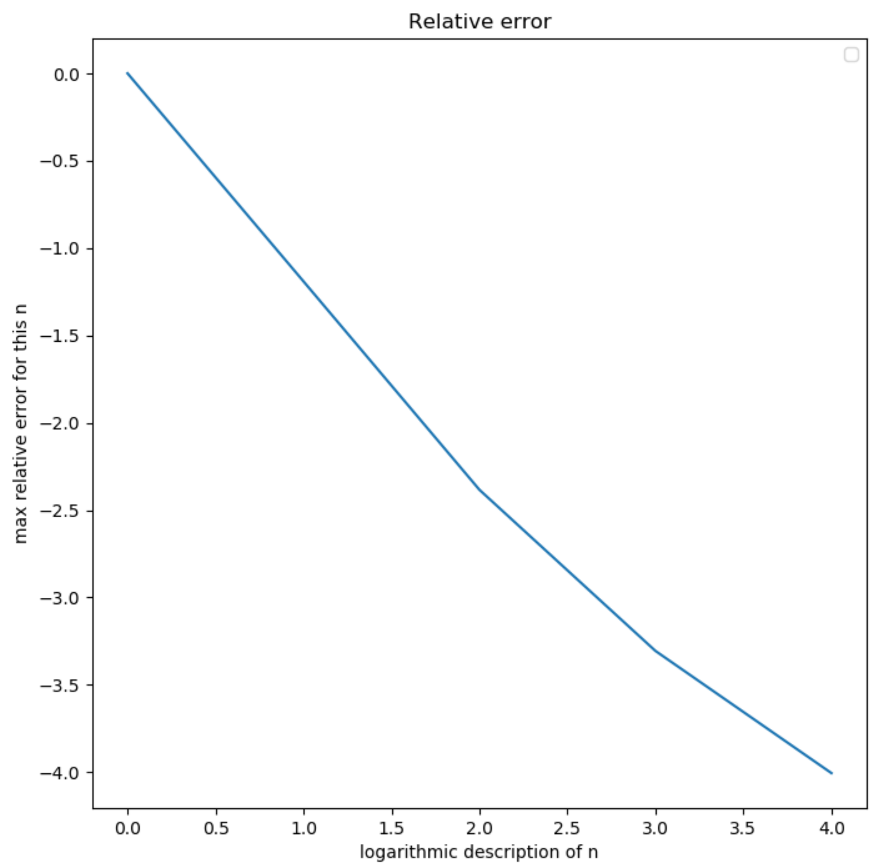

FIG. 4. Den absolutte feilen mellom den eksakte verdien, $u(x)$, og vår tilnærming $v(x)$.

problem b

Problem c

PROBLEM 9

Problem a

Algorithm 2 Spesialisert algoritme

$$a \leftarrow -1$$

$$b \leftarrow 2$$

$$c \leftarrow -1$$

$$d \leftarrow a \cdot c$$

▷ 1 FLOP

for $i = 0, 1, 2, \cdots, n-1$ **do** ▷ Forward elemination

$$\widetilde{b}_{i+1} \leftarrow \widetilde{b}_{i+1} - \frac{d}{\widetilde{b}_i}$$

▷ 2 FLOPs

$$\widetilde{g}_{i+1} \leftarrow g_{i+1} - \frac{a}{\widetilde{b}_i} g_i$$

▷ 3 FLOPs

for $i = n-1, n-2, \cdots, 0$ **do** ▷ Backward elemination

$$v_i \leftarrow \frac{\widetilde{g}_i - cv_{i+1}}{\widetilde{b}_i}$$

▷ 3 FLOPs

---

problem b

Vi regner ut produktet $a \cdot c$ som gir oss 1 FLOP. For "Forward Elemination" loopen så har vi 5 FLOPs per iterasjoner, som gir tilsammen $5n$. I loopen for "Backwards elemination" så har 3 FLOPs per iterasjon som gir oss $3n$ FLOPs tilsammen. Totalt ender vi da opp med $8n + 1$ FLOPs.

Problem c

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

$ make p9all

PROBLEM 10

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.