Laboration 1 Numerisk Analys

Josefine Klintberg, Anna Larsson, Louise Abrahamsson Kwetczer

3 oktober 2018

Innehåll

1 Syfte med laborationen

Avsikten med labben är att snabbt komma igång med matlabprogrammering. Att öva på numerisk lösning av differentialekvationer med Matlab. Att lösa ett optimeringsproblem med en differentialekvation som bivillkor. Att repetera och hantera metoden med Lagrangefunktioner för att lösa optimeringsproblem med bivillkor. Att studera ett exempel när en differentialekvation används för att bestämma koefficienter i ekvationen, det vill säga ett inverst problem.

2 Del A1

2.1 Uppgift

En populations antal individer $y(t)$ vid tiden $t > 0$ beskrivs av differentialekvationen

$$y'(t) - ky(t)(a - y(t)).t > 0$$

med begynnelsevärdet $y(0) = y_0$ där k, a och y_0 är givna positiva konstanter. Byt variabler $\tau = tka$ och $z(\tau) = y(t)/a$ och visa att

$$z'(\tau) = z(\tau)(1 - z(\tau)), > 0 \tag{1}$$

Skriv ett matlabprogram som löser denna differentialekvation med Eulers metod

$$\frac{z_{n+1} - z_n}{\Delta\tau} = f(z_n), n = 0, 1, 2, ..., N - 1$$

där z_n är en approximation av $z(n\Delta\tau)$ med tidssteget $\Delta\tau = T/N$ och $f(z) = z(1 - z)$, för olika val av begynnelsedata $z(0)$. Visa figurer av Eulerapproximation av funktionen $z : [0, T] \to \mathbb{R}$ med lämpligt val av tidssteg $\Delta\tau$ och sluttid T. Presentera även en figur som illustrerar Eulermetodens noggrannhet med avseende på $\Delta\tau$ parametern.

2.2 Resultat

De beräkningar vi behövde göra var följande:

$$y'(t) = kay - ky^2, t > 0, \tau = tka$$

Vi sätter att

$$z(\tau) = \frac{y}{a} \to y = z(\tau)a$$

och

$$y' = z'(\tau)a^2k$$

Detta leder till att

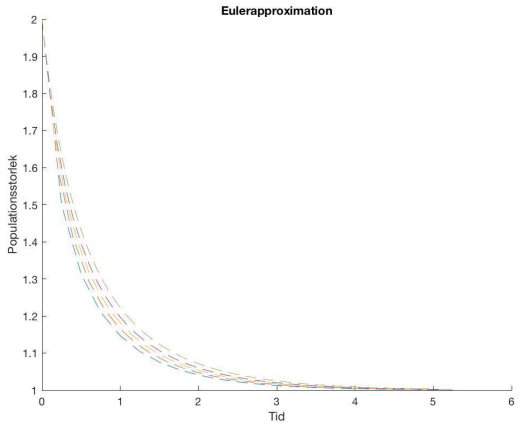
$$z'(\tau)a^2k = ka^2z(\tau) - kz^2(\tau)a^2 \Leftrightarrow z'(\tau) = z(\tau) - z^2(\tau) \Leftrightarrow z(\tau)(1 - z(\tau)) = z'(\tau)$$

Den sista ekvationen använder vi oss av för att implementera Eulermetoden. Eulermetoden är ett sätt att lösa ordinära differentielekvationer med ett givet begynnelsevärde. Metoden utgår från att man delar in differentialekvationen i diskreta stegintervall och sedan approximerar steget med hjälp av en finit differensmetod. Den explicita Eulermetoden fås av:

$$\frac{z_{n+1} - z_n}{\Delta} = f(z_n) \; n = 1, 2, ..., N - 1$$

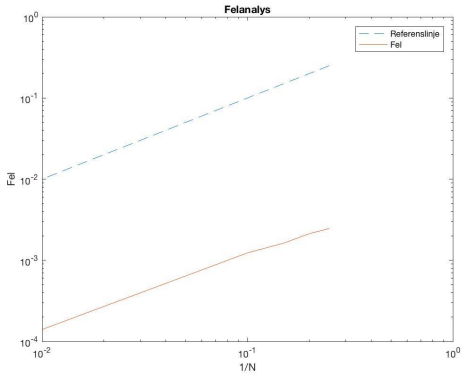
Fullständig matlabkod kan hittas i appendix (6.1).

När vi implementerar Eulermetoden i Matlab så får vi följande lösningskurvor när vi plottar populationsstorleken mot tiden och med olika storlekar på $\Delta\tau$. Några lösningskurvor visas i figur 1.



Figur 1: Jämförelse av populationsstorlek per tid med olika $\Delta\tau$

2.3 Felanalys



Figur 2: Loglog-plot av det globala felet för olika $\Delta\tau$

Ekvation 1 är en separabel differentialekvation som kan lösas analytiskt, med lösningen:

$$z(\tau) = \frac{e^\tau}{C + e^\tau}$$

Vi har därmed möjlighet att jämföra vår approximation men den riktiga lösningen. I figur 2 har vi bestämt det globala felet för ett antal olika $\Delta\tau$. Vi ser att lutningen på linjen är ungefär 1. Därmed minskar det globala felet linjärt med ordning 1, i enlighet med Eulers metod. Vid en finare indelning med ett mindre $\Delta\tau$ får vi alltså en bättre approximation av den riktiga lösningen.

3 Del A2

3.1 Uppgift

Den implicita Eulermetoden för problemet i uppgift A1 lyder

$$\frac{z_{n+1} - z_n}{\Delta\tau} = f(z_{n+1}), n = 0, 1, 2, ..., N - 1$$

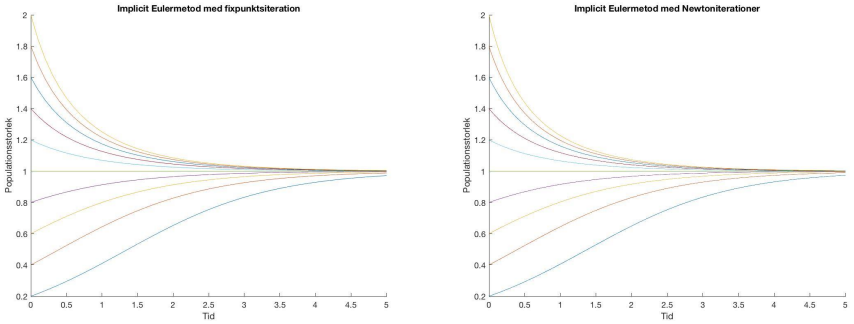
Skriv ett matlabprogram för implicita Eulermetoden där $z_0 = 1$, baserat på
a) fixpunktsiterationer (utan att använda f').
b) Newtoniterationer (som är en slags fixpunktsiteration).

3.2 Fixpunktsiteration och Newtons metod

Fixpunktsiteration innebär allmänt att om vi har $f(x) = 0$ så kan vi skriva detta som $x = g(x)$. En lösning till denna ekvation kallas för fixpunkt till g.

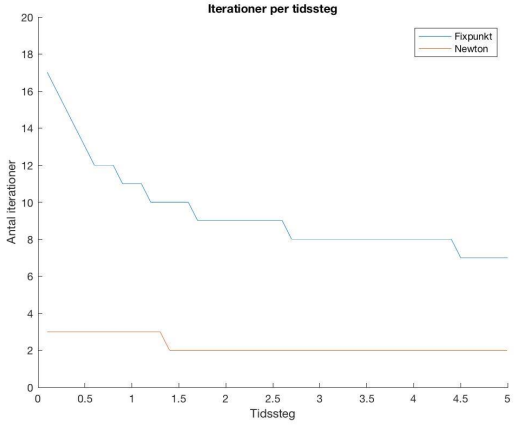
Lösningsmetoden går sedan ut på att vi väljer en startpunkt x_0 och därefter beräknar vi x_k genom rekursionen $x_{k+1} = g(x_k)$.

Newtons metod är en typ av fixpunktsiteration. Om vi åter betraktar ett allmänt fall av ekvationen $f(x) = 0$, då kan vi skriva detta som en fixpunktsekvation $x = g(x)$. Vi börjar åter igen med en startgissning x_0 och förbättrar gissningen x_k genom att följa tangenten till f genom $(x_k, f(x_k))$ till x-axeln. Denna approximation ges av $x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$.



(a) Implicit Euler med fixpunktsiteration (b) Implicit Euler med newtoniterationer

Figur 3: Lösningskurvor till differentialekvationen med olika initialvärden



Figur 4: Jämförelse av antal iterationer för Newtons metod och fixpunktsmetoden med initialvärde $z_0 = 2$.

Som vi ser i figur 4 så krävs det väsentligt färre antal iterationssteg för

Newtons metod än för fixpunktsiterationen. Detta beror på att fixpunktsmetoden har linjär konvergens medan Newtons metod har kvadratisk konvergens. Även om fixpunktsiterationerna minskar i antal ju fler tidssteg som tas blir den aldrig lika snabb som Newtons metod.

4 Jämförelse av explicit och implicit Eulermetod

4.1 Stabilitet

För att undersöka stabilitet hos den explicita och implicita Eulermetoden betraktar vi följande modell

$$z'(t) = \lambda z(t)$$
$$z(0) = 1$$

där λ är en positiv konstant.

Detta ger upphov till den explicita Eulermetoden enligt:

$$z'(t) = \frac{z_{n+1} - z_n}{\Delta t} \Rightarrow \frac{z_{n+1} - z_n}{\Delta t} = -\lambda z_n$$
$$\Rightarrow z_{n+1} = z_n - \lambda \Delta t z_n = (1 - \lambda \Delta t) z_n$$

Och den implicita Eulermetoden:

$$z_{n+1} = z_n + \Delta t(-\lambda z_{n+1}) \Rightarrow z_{n+1}(1 + \lambda \Delta t) = z_n$$
$$\Rightarrow z_{n+1} = \frac{z_n}{1 + \lambda \Delta t}$$

Ekvationen kan även lösas analytiskt med hjälp av en integrerande faktor:
$z'(t) = -\lambda z$
Integrerande faktor: $e^{\lambda t} \Rightarrow z(t) = ce^{\lambda t}$
Med begynnelsevärdet fås $z(t) = e^{\lambda t}$
Insättning av $t = (n+1)\Delta t$ ger:
$z((n+1)\Delta t) = e^{-\lambda((n+1)\Delta t)} = e^{\lambda n \Delta t - \lambda \Delta t} = e^{-\lambda \Delta t} z(n\Delta t)$

Om vi låter $n \to \infty$ ser vi att den exakta lösningen går mot 0 enligt:

$$z((n+1)\Delta t) = e^{-\lambda((n+1)\Delta t)} = e^{\lambda n \Delta t - \lambda \Delta t} = e^{-\lambda \Delta t} z(n\Delta t) = ... = e^{-n\lambda \Delta t} z_0 \to 0$$

Vi gör samma undersökning för den explicita och implicita metoden.

Explicit:
$z_{n+1} = (1 - \lambda\Delta t)^n z_0$ Då $n \to \infty$ får vi tre fall beroende på tecknet på $\lambda\Delta t$.
Fall 1: $\lambda\Delta t < 1 \to z_{n+1} = 0$
Fall 2: $\lambda\Delta t = 1 \to z_{n+1} = 0$
Fall 3: $\lambda\Delta t > 1$ ger upphov till två fall:

$$|1 - \lambda\Delta t| < 1 \to z_{n+1} = 0$$

$$|1 - \lambda\Delta t| > 1 \to z_{n+1} \to \pm\infty$$

Implicit:
$z_{n+1} = \frac{z_n}{1+\lambda\Delta t} = ... = \frac{z_0}{(1+\lambda\Delta t)^n}$

För alla val av λ och Δt kommer nämnaren $1 + \lambda\Delta t > 1$ vilket medför att den implicita metoden alltid kommer att konvergera mot 0 likt den exakta lösningen.

4.2 Konvergens

När vi jämför mellan den explicita och den implicita Eulermetoden får vi en skillnad i beräkningshastighet. Båda metoderna är av konvergensordning 1, men i den explicita Eulermetoden så behöver ingen ekvation lösas i tidsstegen, medan för den implicita Eulermetoden så måste en ekvation lösas i varje tidssteg.

Fördelar med implicita metoder är att de i allmänhet har obegränsade stabilitetsområden medan de explicita metoderna har begränsat stabilitetsområde.

Valet om huruvida en explicit eller implicit metod ska användas beror på målet med beräkningen. I det enklare fall vi har studerat i denna uppgift så är skillnaden i tidsåtgång inte markant, men för större mängd data så kommer den implicita Eulermetoden att ta allt längre tid till följd av ekvationslösningen i varje steg.

Matlabprogrammen för metoderna kan ses i appendix (6.2 samt 6.3).

5 Del B

5.1 Bakgrund

Vi vill studera utböjningen av en fritt upplagd balk. Vi låter $u(x)$ beteckna utböjningen av balken i punkten x, vidare löser $u(x)$ Bernoulli-Eulers ekvation

$$(b(x)u''(x))'' = f(x), x \in (0,1)$$

$$u(1) = b(1)u''(1) = u(0) = b(0)u''(0) = 0 \tag{2}$$

där $b(x)$ betecknar böjstyvheten och $f(x)$ kraften per längdenhet. Bernoulli-Eulers ekvation är ett samband mellan mekanisk spänning och deformation i en balk under böjning som är viktig inom hållfasthetslära. Med variabelbytet $w(x) = b(x)u''(x)$ kan denna ekvation skrivas om till systemet

$$w''(x) = f(x), x \in (0,1), w(0) = w(1) = 0$$

$$b(x)u''(x) = w(x), x \in (0,1), u(0) = u(1) = 0 \tag{3}$$

i vårt fall antar vi att böjstyvheten $b(x)$ är styckvis konstant. Där balken delas in i m stycken lika långa segment l .

$$b(x) = \begin{cases} b_1, x < 1/M \\ .. \\ b_M, x > \frac{M-1}{M} \end{cases}$$

Målet är att välja parametrarna $b_1 > 0$, ... ,$b_M > 0$ under villkoret
$b_M = 1 - b_1 - ... - b_{M-1} > 0$ så att utböjningen $u(x_*)$ minimeras, där x_* är en position i balken. Villkoret $b_1 + ... + b_M = 1$ kan tolkas som att vi har en given mängd material att göra vår balk av, med varierande bredd och konstant höjd.

5.2 Lagrangefunktionen

Vi gör approximationen $U_n \simeq u(x_n)$ och inför indelningen $x_n = n\Delta x, n = 0, 1, 2, .., N, \Delta x = \frac{1}{N}$. Andraderivatan approximeras av andradifferensen:

$$(D^2U)_n = \frac{(U_{n+1} - 2U_n + U_{n-1})}{(\Delta x)^2}, n = 1, ..., N-1$$

Ekvationen (2) approximeras nu med:

$$(D^2BD^2U)_n = F_n, n = 1, ...N-1, (D^2U)_0 = (D^2U)_N = 0, U_0 = U_N = 0 \tag{4}$$

Och ekvationssystemet (3) blir:

$$(D^2W)_n = F_n, n = 1, ..., N-1, W_0 = W_N = 0$$

$$B_n(D^2U)_n = W_n, n = 1, ..., N-1, U_0 = U_N = 0 \tag{5}$$

med $B_n \simeq b(x_n)$ och $F_n \simeq f(x_n)$. Vi kan tolka D^2U som en matris vektor multiplikation, med den tridiagonala matrisen som har $-2/(\Delta x)^2$ i huvuddiagonalen och i $1/(\Delta x)^2$ i övre och undre diagonalen.

$$D^2 = \frac{1}{(\Delta x)^2}\begin{bmatrix} -2 & 1 & & & \\ . & . & . & & \\ & 1 & -2 & 1 & \\ & . & . & . & \\ & & & 1 & -2 \end{bmatrix}$$

Vi ska nu minimera utböjningen $U(x_{n_*})$ under bivillkoret att ekvation (4) är uppfylld. För att göra detta betraktar vi Lagrangefunktionen $L : \mathbb{R}^{N-1} \times \mathbb{R}^{N-1} \times \mathbb{R}^2 \to \mathbb{R}$ definierad av

$$L(U, \Lambda, b) := \sum_{n=1}^{N-1} (U(x_{n_*})\delta_{nn_*}(\Delta x)^{-1} + (F - D^2BD^2U)_n\Lambda_n)\Delta x,$$

där

$$\delta_{nk} = \begin{cases} 1, n = k \\ 0, n \neq k \end{cases}$$

Här är B diagonalmatrisen med B_n i diagonalen,

$$B = diag(B_n) = \begin{bmatrix} b_1 & & & & & \\ & . & & & & \\ & & b_2 & & & \\ & & & . & & \\ & & & & 1 - b_1 - ... - b_{M-1} & \\ & & & & & . \end{bmatrix} \tag{6}$$

Låt $G_n := \delta_{nn_*}(\Delta x)^-1$ och skriv skalärprodukten i $\mathbb{R}^{N-1}$ som $G*U = \sum_{n=1}^{N-1} G_nU_n$. Förklara varför Lagrangefunktionen satisfierar

$$L(U, \Lambda, b) = (G * U + F * \Lambda - BD^2U * D^2\Lambda)\Delta x$$

och ger ekvationerna

$$0 = \frac{\partial}{\partial \Lambda_n}L(U, \Lambda, B) = (F_n - (D^2BD^2U)_n)\Delta x, n = 1, ..., N-1, (D^2U)_0 = (D^2U)_N = 0, U_0 = U_N = 0$$

$$0 = \frac{\partial}{\partial U_n}L(U, \Lambda, B) = (G_n - (D^2BD^2\Lambda)_n)\Delta x, n = 1, ..., N-1, (D^2\Lambda)_0 = (D^2\Lambda)_N = 0, \Lambda_0 = \Lambda_N = 0$$

$$0 = \frac{\partial}{\partial b_i}L(U, \Lambda, B) = (-\frac{\partial B}{\partial b_i}D^2U * D^2\Lambda)\Delta x, i = 1, 2. \tag{7}$$

5.2.1 Resultat

Genom att $G*U = \sum_{n=1}^{N-1} G_n U_n$ och definitionen av Kroneckerdelta ger trivialt att $\sum_{n=1}^{N-1} U(x_{n^*})\delta_{nn^*}(\Delta t)^{-1} = G*U$.

Vi fortsätter sedan med att skriva om den andra termen i Lagrangefunktionen.

$(F - D^2BD^2U)_n\Lambda_n = F_n\Lambda_n - (D^2BD^2U)_n\Lambda_n \rightarrow \sum_{n=1}^{N-1}(F - D^2BD^2U)_n\Lambda_n = F\Lambda - (D^2BD^2U)*\Lambda$

Vi vill nu skriva om $(D^2BD^2U)*\Lambda$:

$D^2BD^2U*\Lambda = \Lambda^T(D^2BD^2U) = (\Lambda^T D^2)(BD^2U) = (D^2 symmetrisk) = (\Lambda^T D^{2^T})(BD^2U) = ((AB)^T = B^TA^T) = (D^2\Lambda)^T(BD^2U) = (BD^2U)*(D^2\Lambda)$

Detta medför att $L = (G*U + F*\Lambda - (BD^2U)*(D^2\Lambda))\Delta x, [*]$. Men vi vill även visa att Lagrangefunktionen satisifierar de olika ekvationerna. Detta gör vi genom att först hitta de kritiska punkterna till L, det vill säga då $\nabla L = \overrightarrow{0}$.

Deriverar vi summan så får vi:

$$\frac{\partial L}{\partial \Lambda_n} = (F - D^2BD^2U)_n$$

vilket är den första ekvationen vi skulle visa. Om vi sedan fortsätter och deriverar [*] så får vi:

$$\frac{\partial L}{\partial b_i} = (-\frac{\partial B}{\partial b_i}D^2U * D^2\Lambda)\Delta x$$

vilket var den tredje ekvationen. För att kunna få $\frac{\partial L}{\partial U_n}$ så måste vi skriva om L lite grann.

$(BD^2U)*(D^2\Lambda) = (D^2\Lambda)^T(BD^2U) = (\Lambda^T D^{2^T})(BD^2U) = \Lambda^T D^2BD^2U = \Lambda^T(D^{2^T}B^TD^{2^T})U = \Lambda^T(D^2BD^2)^TU = (D^2BD^2\Lambda)^TU = (D^2BD^2\Lambda)*U$

Detta medför att vi kan skriva L som:

$$L = (G*U + F*\Lambda - D^2BD^2\Lambda*U)\Delta x$$

och om vi deriverar detta uttrycket med avseende på U_n så får vi:

$$\frac{\partial L}{\partial U_n} = (G_n - (D^2BD^2\Lambda)_n)\Delta x = (G - D^2BD^2\Lambda)_n\Delta x$$

vilket var den sista ekvationen som vi skulle visa att Lagrangefunktionen satisifierar.

5.3 Lösning av balkproblemet

Vi ser att U löser ekvation (4) och Λ löser motsvarande ekvation med högerledet G istället för F:

$$(D^2\overline{W})_n = G_n, n = 1, ..., N-1, W_0 = W_N = 0$$
$$B_n(D^2\Lambda)_n = W_n, n = 1, ..., N-1, \Lambda_0 = \Lambda_N = 0 \quad (8)$$

Ekvationssystemet (7) kan lösas iterativt genom att implementera till exempel gradientmetoden.

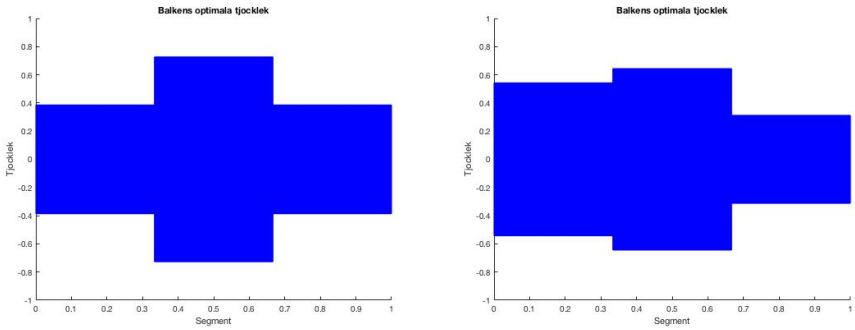
$$b_i^{m+1} = b_i^m - \alpha \frac{\partial}{\partial b_i} L(U^m, \Lambda^m, B^m), i = 1, 2,$$

där α väljs lämpligt litet för att iterationerna ska konvergera och $b_i > 0$.

Gradientmetoden för att minimera en funktion $F : \mathbb{R}^n \to \mathbb{R}$ går ut på att hitta de punkter där gradienten ∇F är nollvektorn. Vi startar med en första gissning av minimipunkt $\overline{x}$. Därefter försöker vi hitta en bättre kandidat till minimipunkten genom att utnyttja att den riktning i vilken funktionen i $\overline{x}$ avtar snabbast är $-\nabla F(\overline{x})$. Så vi beräknar $\overline{x} - \lambda \nabla F(\overline{x})$ för något tal λ och sätter den resulterande vektorn som det nya värdet på $\overline{x}$ och iterationerna börjar om igen. Vi upprepar så denna idé och för tillräckligt många iterationer så bör metoden resultera i ett $\overline{x}$ som är nära ett lokalt minimum till $F(\overline{x})$.

5.3.1 Resultat

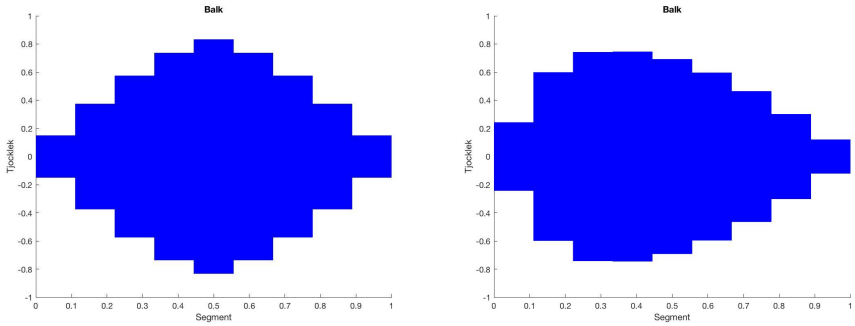
Med hjälp av Bernoulli-Eulers ekvation och Lagrangefunktionen så har vi kunnat ställa upp ett minimeringsproblem där minimeringen utförs iterativt med hjälp av gradientmetoden. Våra resultat då optimeringspunkten är vald till $x_{n^*} = 0.5$ respektive $x_{n^*} = 0.2$ illustreras i bilderna nedan.



(a) Optimeringspunkt 0.5

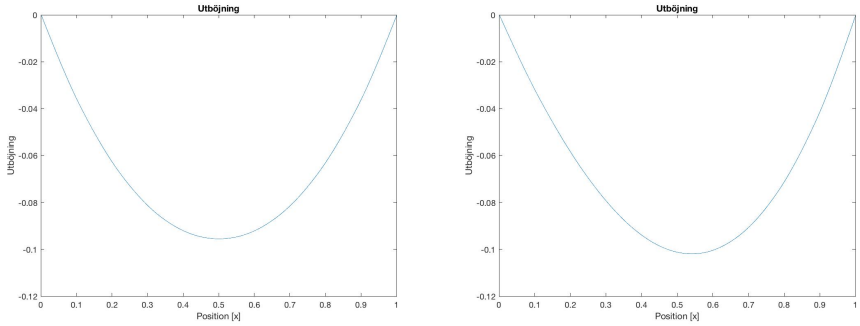
(b) Optimeringspunkt 0.2

Figur 5: Balkens optimala tjocklek då antalet segment är tre.



(a) Optimeringspunkt 0.5 (b) Optimeringspunkt 0.2

Figur 6: Balkens optimala tjocklek då antalet segment är nio.



(a) Optimeringspunkt 0.5 (b) Optimeringspunkt 0.2

Figur 7: Balkens utböjning vid optimeringspunkt $x_* = 0.5$ samt $x_* = 0.2$

Som vi ser i figurena ovan så ger vår metod olika resultat beroende på vilken optimeringspunkt vi har valt. När vi väljer att optimera med avseende på punkten 0.5 så får vi en helt symmetrisk tjocklek på balken, både när vi väljer M=3 och M=9. Däremot när vi väljer optimeringspunkten 0.2 så får vi en balk som ej är symmetrisk och dessutom blir det en ganska stor skillnad mellan fallen M=3 och M=9.

När vi väljer optimeringspunkten 0.2 blir balken tjockare till höger för att göra balken styvare på denna sida, och därmed minimera utböjningen. Vi ser att då vi gör en finare indelning med fler antal segment så får vi en bättre approximation av hur balken optimalt bör se ut.

5.3.2 Förklaring av kod

För ett iterationssteg, $n = 1, 2, ...$, och en given böjstyvhetsmatris $B = B_n$ kan differensekvationen (5) lösas och ger $U = U_n$. På samma sätt erhålls $\Lambda = \Lambda_n$ från (8) med $B = B_n$.

Vi börjar med att definiera vår steglängd samt att bygga upp matriserna. D2 får vi från differensekvationen och F och G fås enligt definition i uppgiften. Eftersom alla randvärden är noll behöver vi inte lägga till dem i F och G. Vi använder funktionen CreateB för att kunna generera B på ett smidigt sätt. I den funktionen utnyttjar vi matlabfunktionen ceil (som avrundar ett element till närmaste heltal) för att skapa ett index så att vi kan avgöra när i B's diagonal som vi ska ha b_1, b_2,..., b_M.

Vi använder likaså funktionen CreatedB för att skapa den matris som innehåller derivatorna av B-matrisen deriverad med avseende på diagonalelementen, dB. W och W2 fås genom definitionen i uppgiften och kan direkt lösas genom att Gausseliminera D2 med avseende på F respektive G. Därefter implementerar vi gradientmetoden och optimerar våra värden på $b_1, b_2, ..., b_M$. I varje iteration uppdateras U_n med hjälp av föregående B_n-matris och slutligen uppdateras B_n-matrisen.

Vi använde oss av matlabkommandot sparse i alla matriser, sparse är ett kommando som komprimerar matriserna genom att inte spara värden som är noll. Detta gjorde att våra beräkningstider snabbades upp markant och vi kunde utföra beräkningarna även för relativt stora värden på N. I slutet av programmet så ritar vi balken sett uppifrån (m.h.a. vår funktion rita) för att visualisera tjockleken av den optimala balken. Vi har valt att visualisera balkens bredd på detta sätt då man kan betrakta $b_1+b_2+...+b_M = 1$ som en materialbegränsning och varje b-värde som en materialmängd.

5.4 Störningsanalys och felanalys

5.4.1 Störningsanalys

Genom att utföra experimentell störningsanalys kunde slutsatser dras om huruvida stora konditionstal var inblandade eller ej. Konditionstalet fås av att ta det relativa framåtfelet genom det relativa bakåtfelet. Genom att variera vissa parametrar kan man då se hur känslig modellen är. Vi valde att ändra summan av den totala massfördelningen $b_1 + b_2 + ... + b_M$ från 1 till t.ex 1.1, 1.01 och 1.001. Nedan följer en tabell av de värden vi fick.

Fel i indata	Fel i utdata	Konditionstal
0.1	0.0185	1
0.01	0.00185	1
0.001	1.85 *10⁻4	1

Tabell 1: Tabell över några störningar i den totala massfördelningen.

De relativa felen i in- och utdata beräknades genom att dividera felen med respektive ostörda värden. Beräkning av konditionstalet resulterade i de tabellförda värdena. Vi kunde med hjälp av dessa värden dra slutsatsen att det inte är några stora konditionstal inblandade.

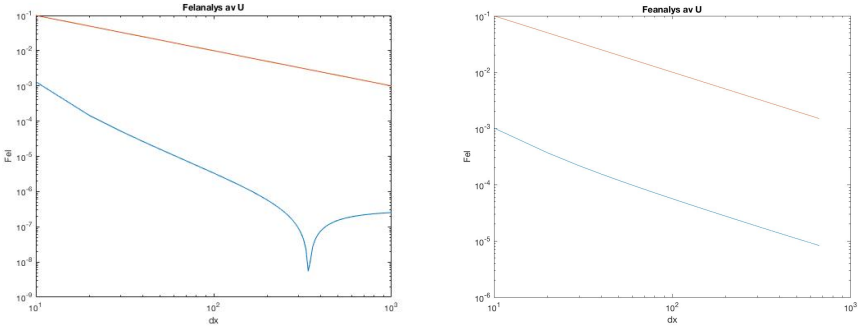
5.4.2 Felanalys

De felkällor som förekommer i lösningen beror bland annat på de approximationer som görs, fel i modellen samt vald noggrannhet av vissa parametrar. Ett problem som kan uppstå när vi gör en finare indelning av balken är att fel i beräkningar av vissa funktioner förstärks. Detta beror på att fler iterationer görs och att vissa fel då kan växa med varje iteration, exempelvis approximationen av derivatan och att det i varje iteration sker ett visst avrundningsfel. Toleransen för hur litet felet får vara kan visserligen sättas väldigt lågt, men det kommer fortfarande att förekomma ett visst fel. Låg feltolerans resulterar dock i långa och krävande beräkningar så därför görs en avvägning mellan tolererat fel och beräkningstid.

Vid beräkningar av dessa typer kommer det alltid att förekomma ett visst fel på grund av att modellen aldrig kan vara helt korrekt. Vet vi verkligen att vi har mängden 1 balk? Och att kraften är precis 1? I den här modellen förutsätts bland annat att kraften är precis lika med 1 och jämnt fördelad över hela balken. I verkligheten kan detta naturligtvis variera, men som modell fungerar det bra.

Noggrannheten kan testas utan att ha tillgång till en exakt lösning genom att välja ett väldigt stort värde på N, det vill säga en väldigt fin indelning. Dock ger detta problem då det i allmänhet kräver mer beräkningar i varje iteration och antalet beräkningar begränsar oss för hur stort N vi kan välja.

Vi valde att köra beräkningarna en gång för N=20 000 vilket ger en lösning som är mycket nära den korrekta och som vi därför kunde jämföra med. Vi betraktar sedan hur felet i utböjningen U beror av hur fin indelningen är. Vi låter N variera mellan 6 och 1200 och jämför U-värdet i utböjningspunkten med motsvarande värde i vår exakta lösning. Nedan följer resultatet då antalet segment i balken är tre respektive nio.



(a) Felanalys av U då antalet segment är tre (b) Felanalys av U då antalet segment är nio

Figur 8: Felanalys av U med fixt B, röd linje är referenslinje med lutning 1.

Vi jämför felet med en referenslinje med lutning 1 och kan se att lutningen på felkurvan är samma. Den dipp som förekommer i vänstra figuren är intressant och beror troligtvis på att vi vid just detta värde har en väldigt bra lösning.

6 Appendix med Matlab-kod

6.1 A1 Explicit Eulermetod

```
>> z0 = 2;
>> t0 = 0;
>> T = 5;
>> dtV=[0.25 0.2 0.15  0.1 0.01]
>> for j=1:length(dtV)
>>      dt = dtV(j)
>>      N=T/dt
>>      t=[t0]
>>      z=[z0]
>>      for i=0:N
>>          t=[t t(end)+dt]
>>          zn1=z(end)+dt*(z(end)*(1-z(end)))
>>          z=[z zn1]
>>      end
>>      zend=[zend z(end)]
>>      plot(t, z, '--')
>>      title('Eulerapproximation')
>>      ylabel('Populationsstorlek')
>>      xlabel('Tid')
>> end

>> rs = @(t) (z0/(1-z0) * exp(t) / (1 + (z0 / (1-z0)) * exp(t)));

>> fel= abs(zend-rs(T));
>> figure
>> loglog(dtV, dtV, '--')
>> title('Felanalys')
>> ylabel('Fel')
>> xlabel('1/N')
>> hold on
>> loglog(dtV, fel)
>> legend('Referenslinje', 'Fel')
```

6.2 A2 Implicit Eulermetod med fixpunktsiteration

```
>> close all
>> hold on

>> t0=0;
>> z0=0.2:0.2:2;
>> T=5;
```

```
>> dt=0.1;
>> N = (T-t0)/dt;
>> tol= 1e-10;

>> f= @(w,z) dt*w - dt*w^2 + z;

>> for j=1:length(z0)
>>     z=[z0(j)];
>>     t=[t0];

>>     for i=1:N
>>         w = z(end);

>>         while abs(f(w,z(end))-w) > tol
>>             w = f(w,z(end));
>>         end

>>         z=[z w];
>>         t=[t t(end) + dt];
>>     end
>> plot(t,z)
>> ylabel('Populationsstorlek')
>> xlabel('Tid')
>> title('Implicit Eulermetod med fixpunktsiteration')
>> end
```

6.3 A2 Implicit Eulermetod med Newtoniteration

```
>> close all
>> hold on

>> t0=0;
>> z0=0.2:0.2:2;
>> T=5;
>> dt=0.01;
>> N = (T-t0)/dt;
>> tol= 1e-10;

>> f= @(w,z) w - dt*w + dt * w^2 - z;
>> df= @(w) 1 - dt + 2*dt*w;

>> for j=1:length(z0)
>>    z=[z0(j)];
>>    t=[t0];
```

```
>>    for i=1:N
>>        w = z(end);

>>        while abs(f(w,z(end))) > tol
>>             w = w - f(w,z(end)) / df(w);
>>        end

>>        z=[z w];
>>        t=[t t(end) + dt];
>>    end
>> plot(t,z)
>> title('Implicit Eulermetod med Newtoniterationer')
>> ylabel('Populationsstorlek')
>> xlabel('Tid')
>> end
```

6.4 Generaliserad balk, huvudkod

```
>> function Bv= B1gener(M,N,xp,alpha)
>> dx=1/N;
>> D2= sparse(diag(ones(N-1,1) * -2) + diag(ones(N-2,1),1) + diag(ones(N-2,1),-1));
>> D2= 1/(dx)^2 * D2;
>> Bv = 1/M * ones(1,M);
>> B=Create_B(Bv,M,N);
>> F= ones(N-1,1);
>> W= sparse(D2\F);
>> G = sparse(zeros(N-1,1));
>> G(round(xp*N)) = 1/dx;
>> W2= sparse(D2\G);
>> dB = Create_dB(M,N);
%Gradientmetod
>> Tol=1e-10;
>> a=alpha;
>> bo = zeros(1,length(Bv));
>> while max(abs(Bv-bo))>Tol
>>     bo = Bv;
>>     Bv(end)= 1;
>>     U =((B*D2)\W);
>>     Lambda =((B*D2)\W2);
>>     for i=1:M-1
>>         Bv(i) = Bv(i) + a*((dB{i}*D2*U)'*(D2*Lambda))*dx;
>>         Bv(end) = Bv(end) - Bv(i);
>>     end
>>     B = Update_B(N,M,B,Bv);
```

```
>> end
>> rita(Bv,M)
```

6.4.1 Skapa B

```
>> function B = Create_B(Bv,M,N)
>> B= zeros(N-1);
>> dx=1/N;
>> stepl=1/M;
>> for i=1:N-1
>>     x = i*dx;
>>     index = ceil(x / stepl);
>>     B(i,i)=Bv(index);
>> end
>> B = sparse(B);
>> end
```

6.4.2 Uppdatera B

```
>> function B = Update_B(N,M,Bin,Bv)
>> dx=1/N;
>> stepl=1/M;
>> for i=1:N-1
>>     x = i*dx;
>>    index = ceil(x / stepl);
>>     Bin(i,i)=Bv(index);
>> end
>> B = sparse(Bin);
>> end
```

6.4.3 Skapa dB

```
>> function DeltaB = Create_dB(M,N)
>> DeltaB=cell(M-1,1);
>> dx=1/N;
>> for j=1:M-1
>>     dB=sparse(zeros(N-1));
>>     for i=1:N-1
>>         if dx * i >=(j-1)* 1/M  && dx*i < j*1/M
>>             dB(i,i)=1;
>>         end
>>         if dx * i >= (M-1)/M
>>             dB(i,i) = -1;
>>         end
>>     end
```

```
>>     DeltaB{j}=dB;
>> end
```

6.5 Rita balk

```
>> function rita(Bv,M)
>> hold on
>> axis([0 1 -1 1])
>> x=0;
>> for b=1:length(Bv)
>>     y=-Bv(b)/2 * length(Bv);
>>     rect=rectangle('Position',[x,y,1/M,Bv(b) * length(Bv)]);
>>     set(rect,'Edgecolor','b');
>>     set(rect,'Facecolor','b');
>>     x=x+1/M;
>> end
```

6.6 Fel i U

```
>> close all
>> load('B_ex.mat');
>> M=3;
>> xp= 0.5;
>> Um=[];
>> A=[0];
>> for N=6:12:1200
>>     B = Create_B(B_ex,M,N);
>>  D2= sparse(diag(ones(N-1,1) * -2) + diag(ones(N-2,1),1) + diag(ones(N-2,1),-1));
>>  D2= 1/(1/N)^2 * D2;
>>     F= ones(N-1,1);
>>     W= D2\F;
>>     U =(B*D2)\W;
>>     Um = [Um U(xp*N)];
>>     A = [A A(end)+10];
>> end
>> load('U_ex.mat')
>> fel = abs(U_ex-Um);
>> loglog(A(2:end),fel)
>> hold on
>> loglog(A(2:end),1./A(2:end))
```