NUMERICAL ANALYSIS

EN SAMMANFATTNING AV DAVID KARLSSON E08

Figur. Cubic splines kan användas för att ge mjuka kamerarörelser i 3d animationer.

Förord

För allt material, satser och definitioner som jag sammanfattat här refereras till

- *An introduction to Numerical Analysis av Endre Süli och David Mayers*
- *Numerical Analysis av Timothy Sauer*
- *Claus Führers föreläsningar*

(Reservation görs för eventuella fel).

Innehåll

Kapitel 1

Iteration

1.0.1 Sats 1.1 Funktioner som korsar x-axeln.

Om f är en reell funktion, kontinuerlig på intervallet $[a, b]$ på reella axeln och $f(a)f(b) \leq 0$. Då existerar ett ξ i $[a, b]$så att så att $f(\xi) = 0$. Detta innebär att f har en rot i $[a, b]$

1.0.2 Sats 1.2 Brouwer's fixpunkt

Antag att g är en reell funktion, kontinuerlig på intervallet $[a, b]$ på reella axeln och låt $g(x) \in [a, b]$ för alla $x \in [a, b]$. Då finns ett ξ i $[a, b]$ så att $\xi = g(\xi)$. Det reella talet ξ kallas då en fixpunkt till funktionen g.

1.0.3 Definition 1.1 Enkel iteration

Antag att g är en reell funktion, kontinuerlig på intervallet $[a, b]$ på reella axeln och låt $g(x) \in [a, b]$ för alla $x \in [a, b]$. Givet att $x_0 \in [a, b]$, kallas

$$x_{k+1} = g(x_k),\ k = 0, 1, 2, ..., \tag{1.1}$$

för en enkel iteration (simple iteration), k är iteratorvariabeln.

1.0.4 Definition 1.2 Kontraktion (Contraction)

Antag att g är en reell funktion, kontinuerlig på intervallet $[a, b]$ på reella axeln. g sägs vara en kontraktion (åtstramning) på $[a, b]$ då det finns en konstant L så att $0 < L < 1$ och

$$|g(x) - g(y)| \leq L|x - y| \quad \forall x, y \in [a, b] \tag{1.2}$$

1.0.5 Sats 1.3 Kontraktionsavbildning (Contraction mapping)

Antag att g är en reell funktion, kontinuerlig på intervallet $[a, b]$ på reella axeln och låt $g(x) \in [a, b]$ för alla $x \in [a, b]$. Antag också att g är en kontraktion på [a,b]. Då har g en unik fixpunkt ξ i intervallet $[a, b]$ och sekvensen x_k (Def 1.1) konvergerar mot ξ då $k \longrightarrow \infty$ för alla startvärden x_0 i $[a, b]$.

1.0.6 Sats 1.5 konvergens med hjälp av derivata

Antag att g är en reell funktion, kontinuerlig på intervallet $[a, b]$ på reella axeln och låt $g(x) \in [a, b]$ för alla $x \in [a, b]$. Låt $\xi = g(\xi) \in [a, b]$ vara en fixpunkt till g och antag att g har en kontinuerlig derivata i närheten av ξ med $|g'(\xi)| < 1$. Då konvergerar sekvenesen x_k (Def 1.1) mot ξ då $k \longrightarrow \infty$ förutsatt att startvärdet x_0 ligger tillräckligt nära ξ.

1.0.7 Definition 1.4 Konvergens

Antag att $\xi = \lim_{k\to\infty} x_k$. Vi säger att sekvensen x_k konvergerar mot ξ åtminstone lineärt om det finns en sekvens ,ϵ_k, av reella positiva tal som konvergerar mot 0 och $\mu \in (0, 1)$, så att

$$|x_k - \xi| \leq \epsilon_k \quad k = 0, 1, 2, ..., \; och \; \lim_{k\to\infty} \frac{\epsilon_{k+1}}{\epsilon_k} = \mu \tag{1.3}$$

Dessutom sägs sekvensen (x_k) vara

- Superlineär om $\lim_{k\to\infty} \frac{\epsilon_{k+1}}{\epsilon_k} = \mu = 0$
- Sublinäer om $\mu = 1$ och $\epsilon_k = |x_k - \xi|, \; k = 0, 1, 2, ..., \;$. Hastigheten är långsammare än lineär.

1.0.8 Definition 1.5 Konvergenshastighet (rate of convergence)

Hastigheten med vilken en iteration konvergerar kallas konvergenshastighet och definieras av

$$\rho = -\log_{10} \mu \tag{1.4}$$

Kapitel 2

Ekvations lösning

För lösning av den homogena funktionen $f(x) = 0$ behövs iterationsmetoder som med stor säkerhet konvergerar. Bisektions metoden är robust men Newtons metod vinner när iterationsvariabeln kommit tillräckligt nära lösningen.

(För lösning av $g(x) = x$ transformeras $g(x)$ till $f(x) = 0$ genom $f(x) = g(x) - x$).

2.1 Bisektionsmetoden

2.1.1 Sats 2.1 Bolzanos sats, mellanliggande värden

Låt f vara en reell funktion, kontinuerlig på intervallet $[a, b]$ på reella axeln. Om $f(a) \neq f(b)$ och c är ett tal som ligger mellan $f(a)$ och $f(b)$ så finns ett motsvarande tal x_c mellan a och b

$$c = f(x_c) \tag{2.1}$$

2.1.2 Sats 2.2 Tolerans

För en given tolerans fås det absoluta felet hos algoritmen

$$|x_{i+1} - x_i| < tolerans \tag{2.2}$$

och det relativa felet, i fall lösningen är något skilld från 0

$$\frac{|x_{i+1} - x_i|}{|x_{i+1}|} < tolerans \tag{2.3}$$

2.1.3 Definition 2.1 Bisektionsmetoden

Givet en funktion som i ett intervall $[a, b]$ korsar x-axeln ($f(a)f(b) < 0$). Tag en punkt $c = \frac{a+b}{2}$, om c inte är nollställe så har c samma tecken som antingen f(a) eller f(b). Skapa ett nytt intervall och ersätt den av a, b som ger $f(x)$ samma tecken som c med c. Processen upprepas och man närmar sig nollstället. Sluta när approxiamtionen av nollstället har godtagbar precision.

2.1.4 Kod 2.1 Bisektionsmetoden

Givet (Sats 1.1) ett intervall $[a, b]$ så att $f(a)f(b) < 0$

```
while (b-a)/2 > tolerance
    c:=(a+b)/2
    if f(c)=0 : end
    if f(a)f(c) < 0
        b:=c
    else
    a:=c
end
```

Det slutliga intervallet $[a, b]$ innehåller en rot. Man approximerar roten som

$$x = \frac{a+b}{2} \tag{2.4}$$

2.2 Fixpunkt-iteration (FPI)

2.2.1 Definition 2.2 Fixpunkt iterationen som lösning

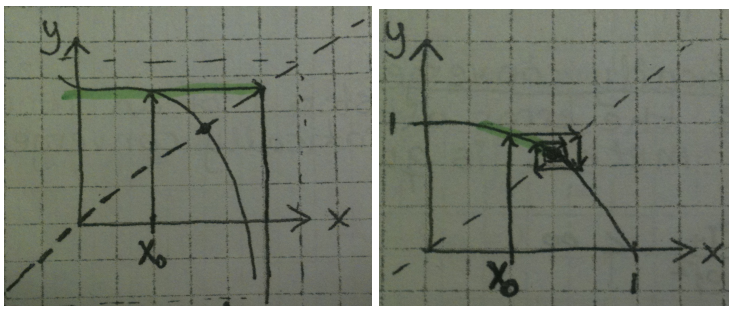
Om f är en reell funktion, kontinuerlig på intervallet $[a, b]$ på reella axeln och $x_0 \in [a, b]$ Så kallas

$$x_{n+1} = f(x_n) \quad n = 0, 1, 2, ..., \tag{2.5}$$

fixpunkt iterationen för f, vilken ger sekvensen $x_0, x_1, x_2, ...$, vilken förhoppningvis konvergerar mot x. Om f är kontinuerlig så är x en fixpunkt till f (Sats 1.2).

2.2.2 Sats 2.3 Konvergens hos FPI

Låt f vara en reell kontinuerlig funktion, att $g(\xi) = \xi$ och att $S = |g'(\xi)| < 1$. Då konvergerar fixpunkt iterationen lineärt mot ξ med hastigheten S för startvärden (x_0) tillräckligt nära ξ. Detta kan tydligare åskådliggöras med cobweb-diagram, figur 2.1.



(a) $g(x) = 1 - x^3$: Divergerar, stegen blir större och större.

(b) $g(x) = (1 - x)^{\frac{1}{3}}$: Konvergerar, stegen blir mindre och mindre.

Figur 2.1: Cobweb diagram för FPI av två omskrivningar av funktionen $x^3 + x - 1 = 0$. Derivatan är markerad i grönt.

2.2.3 Kod 2.2 FPI

```
function [ xiter, flag ] = fpi( f,x0,lambda, TOL)
%fpi, fixed point iteration function
%Input: constant weight lambda (default=1), function to iterate f, initial value x0, toler
%Output: iteration result xiter, good result flag

    x(1)=x0;
    for i=1:100
        x(i+1)=x(i)+lambda*f(x(i));
        if abs(x(i+1)-x(i))<TOL
            xiter=x;
            flag=1;
            return;
        end
    end
    xiter=0;
    flag=0;
end
```

2.2.4 Felupskattning

FPI- algoritmens fel är hela tiden $g(x)-x$ eftersom målet är att det ska bli noll.

$$|x^{(i)} - x^*| \leq \frac{L}{1-L}|x^i - x^{i-1}| \tag{2.6}$$

Följande felskattningar definieras

Felet a Posteriori (efter)

$$|x^{(i)} - x^*| \leq \frac{L}{1-L}|x^n - x^{n-1}|, \text{ där } n \text{ är sista elementet} \tag{2.7}$$

Felet a Priori (före)

$$|x^{(i)} - x^*| \leq \frac{L}{1-L}|x^1 - x^0| \tag{2.8}$$

Felet ur medelvärdessatsen (Se medelvärdessatsen, Appendix hjälpsatser):

$$|g(x) - g(y)| = max|g'(\xi)||x - y|. \tag{2.9}$$

Om $max|g'(\xi)| < 1$ ger att vi har en kontraktiv funktion.

2.3 Viktad iteration

2.3.1 Definition 2.3 den viktade iterationen som lösning

Antag att f är en reellvärd funktion, kontinuerlig och i närheten av ett reellt tal ξ. viktad iteration (relaxed iteration) använder

$$x_{k+1} = x_k - \lambda f(x_k) \quad k = 0, 1, 2, ..., \tag{2.10}$$

där $\lambda \neq 0$ är ett fixt reellt tal och x_0 är ett givet startvärde nära ξ. Detta ger sekvensen $x_0, x_1, x_2, ...$, vilken förhoppningvis konvergerar mot x.

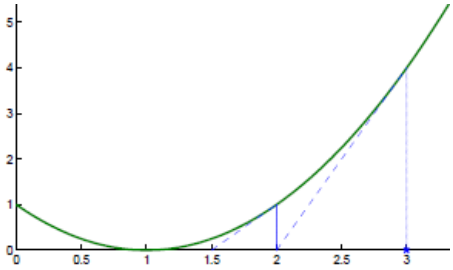
2.3.2 Sats 2.4 Viktad iteration → Newton iteration

En förbättring av den viktade iterationen får man då man sätter $\lambda = \frac{1}{f'(x_k)}$. Denna förbättrade metod kallas Newton iteration.

2.3.3 Definition 2.4 Newton iteration (Newton-Raphsons metod)

Newtons metod för att lösa $f(x) = 0$ defineras av

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} \quad k = 0, 1, 2, ..., \tag{2.11}$$



Figur 2.2: Geometrisk tolkning av Newton-Raphsons metod, $x_0 = 3$

2.3.4 Kod 2.3 Newton iteration

```
function [ xiter, flag ] = NewtIt(f,x0,TOL)
%Netwon method iteration function
%Input: function f, initialvalue x0 and tolerance TOL
%Output: iteration result xiter, good result flag
x(1)=x0;
format long
for i=1:100
    h=1.e-8;
    x(i+1)=x(i)-f(x(i))/((f(x(i)+h)-f(x(i)))/h)
    if abs(x(i+1)-x(i))<TOL
        xiter=x;
        flag=1;
        return
    end
end
        xiter=0;
        flag=0;
end
```

2.3.5 Sats 2.5 Konvergens hos Newton iteration

Antag att f är en reellvärd funktion, kontinuerlig, två gånger deriverbar (f'' existerar) och definerad på det stängda intervallet $I_\delta = [\xi - \delta, \xi + \delta]$, $\delta > 0$ så att $f(\xi) = 0$ och $f''(\xi) \neq 0$. Antag vidare att det finns en positiv konstant A så att

$$\frac{f''(x)}{f'(y)} \leq A \quad \forall x, y \in I_\delta \tag{2.12}$$

Om $|\xi - x_0| \leq h$ där $h = min(\delta, \frac{1}{A})$, så är sekvensen (x_k) definerad av Newtons metod kvadratiskt konvergent mot ξ.

Om vidare det finns ett reellt tal $X > \xi$ så att det i $[\xi, X]$ finns $f' \cap f'' > 0$ så konvergerar Newtons metod kvadratiskt för alla startvärden x_0 i $[\xi, X]$.

Kapitel 3

Ekvationssystem

Tidigare har vi löst ekvationer på formen $f(x) = 0$. Den enklaste av dessa är $ax = b$ där a, b är reella tal, $a \neq 0$. Lösningen till detta problem kan skrivas på formen

$$x = a^{-1}b \tag{3.1}$$

Problemet skrivs enkelt om för att omfatta ekvationssystem. Genom att låta a ersättas med systemmatrisen A fås

$$x = A^{-1}b \tag{3.2}$$

3.1 Gausselemination

3.1.1 Definition 3.1 Nedåt triangulär matris

Låt $n \geq 2$ vara ett heltal. Matrisen $L \in \mathbb{R}^{nxn}$ kallas nedåt triangulär om $l_{ij} = 0$ för varje i och j med $1 \leq i < j \leq n$.

$$L = \begin{pmatrix} l_{1,1} & 0 & \cdots & 0 \\ l_{2,1} & l_{2,2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ l_{n,1} & l_{n,2} & \cdots & l_{n,n} \end{pmatrix} \tag{3.3}$$

Är diagonalen på en nedåt triangulär matris bara ettor så är matrisen enhets-nedåt-triangulär (unit lower triangular).

3.1.2 Definition 3.2 Uppåt triangulär matris

Låt $n \geq 2$ vara ett heltal. Matrisen $U \in \mathbb{R}^{nxn}$ kallas uppåt triangulär om $U_{ij} = 0$ för varje i och j med $1 \leq j < i \leq n$.

$$U = \begin{pmatrix} u_{1,1} & u_{1,2} & u_{1,3} & u_{1,4} & \cdots \\ 0 & u_{2,2} & u_{2,3} & u_{2,4} & \cdots \\ 0 & 0 & u_{3,3} & u_{3,4} & \cdots \\ 0 & 0 & 0 & u_{4,4} & \cdots \\ 0 & 0 & 0 & 0 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix} \tag{3.4}$$

3.1.3 Definition 3.3 Gausselimination

Gausselemination går ut på att succesivt eleminera alla element under diagonalen i systemmatrisen genom att lineärkombinera rader. Detta gör man för att få matrisen triangulär. Systemets lösning är då väldigt enkel.

3.1.4 Kod 3.1 Gausselimination

```
function [x] = gaussElim(A,b)
%   This subroutine will perform Gaussian elmination
%   on the matrix that you pass to it.
%   i.e., given A and b it can be used to find x,
%        Ax = b
%
%   A - matrix for the left hand side.
%   b - vector for the right hand side
%
%   The routine will return the vector x.

%Gauss elimination
for j = 1 : n-1
    if abs(a(j,j))<eps; error('Diagonalelement = 0 hittat'); end
    for i = j+1 : n
        mult = a(i,j)/a(j,j);
        for k = j+1 : n
            a(i,k)= a(i,k)- mult*a(j,k);
        end
        b(i) = b(i) - mult*b(j)
    end
end

% Perform back substitution

 x = zeros(N,1);
 x(N) = b(N)/A(N,N);

 for j=N-1:-1:1,
   x(j) = (b(j)-A(j,j+1:N)*x(j+1:N))/A(j,j);
 end
```

3.1.5 Sats 3.1 Produkten av två triangulära matriser

Produkten av två nedåt triangulära matriser är en nedåt triangulär matris av samma ordning.

$$L_{1,nxn} \cdot L_{2,nxn} = L_{3,nxn} \tag{3.5}$$

På samma sätt är produkten av två uppåt triangulära matriser är en uppåt triangulär matris av samma ordning.

$$U_{1,nxn} \cdot U_{2,nxn} = U_{3,nxn} \tag{3.6}$$

3.1.6 Definition 3.4 Singulär matris

En kvadratisk matris som ej är inverterbar kallas singulär.

3.1.7 Sats 3.2 Singulär triangulär matris

En triangulär matris är singulär om alla diagonal element är 0.

3.1.8 Sats 3.3 Inversen till en triangulär matris

Inversen av en nedåt triangulär matris av ordning n (L_{nxn}) existrerar om matrisen inte är singulär.Inversen är då en nedåt triangulär matris av ordning n. På samma sätt existerar inversen av en uppåt triangulär matris av ordning n då matrisen inte är singulär. Inversen är då en uppåt triangulär matris av ordning n.

3.2 LU Faktorisering

3.2.1 Definition 3.5 LU-faktorisering

Systemmatrisen A kan faktoriseras som produkten av en uppåt triangulär- och en nedåt triangulär matris så att

$$A = LU \tag{3.7}$$

Detta görs genom att lineärkombinera rader och kolonner som vid gausselimination och på så vis få fram L och sedan U.

3.2.2 Sats 3.4 Lösa x ur $A = LU$

Antag L, U kända på grund av en LU-faktorisering. Problemet $Ax = b$ kan då skrivas

$$LUx = b \tag{3.8}$$

Definiera en hjälpvektor c så att

$$c = Ux \tag{3.9}$$

Då kan återsubstitution göras i två steg

- Lös $Lc = b$ för c
- Lös $Ux = c$ för x

De båda stegen ovan är okomplicerade då L och U ovan är triangulära matriser.

3.2.3 Definition 3.6 Permutationsmatris, radpivotering

En permutationsmatrisen, P, är en kvadratisk matris som har precis en etta i varje rad och varje kolumn och vars övriga element är noll. Detta ger permutationsmatrisen följande egenskaper:

$$PP = I \tag{3.10}$$

och vid LU-faktorisering av A, med radpivotering

$$PA = LU \tag{3.11}$$

. Radpivotering används för att ge en permuterad matris som inte har några diagonalelement = 0. Resultatet blir en matris med rader bytta så att LU-faktorisering är möjlig.

3.2.4 Osäkerhet vid LU-faktorisering

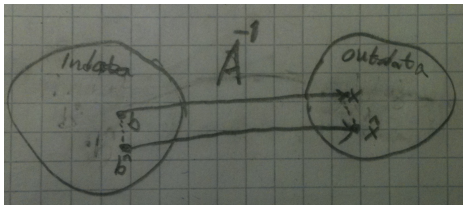
Osäkert b: $\hat{b}$, ger systemet en osäker ekvation:

$$A\hat{x} = \hat{b} \tag{3.12}$$

Vid lösning med hjälp av systemmatrisens invers (A^{-1}) fås:

$b - \hat{b}$ =Absoluta felet in	$x - \hat{x}$ =Absoluta felet ut
$\frac{b-\hat{b}}{b}$ =Relativa felet in	$\frac{x-\hat{x}}{x}$ =Relativa felet ut

Ur dessa fel fås sedan konditionstalet (Definition 3.17).



Figur 3.1: Geometrisk tolkning av felen som upkommer då A^{-1} används för att lösa Ax=b

3.3 Avrundningsfel, normer och konditionstal

För att analysera avrundningsfel i lösningar av ett ekvationssystem används normer.

3.3.1 Definition 3.7 Vektornorm

Antag ett lineärt rum $V \in \mathbb{R}$. Den positiva reella funktionen $||.||$ kallas en norm på rummet V förutsatt att följande villkor uppfylls:

1. $||v|| = 0$ om och endast om $v = 0 \in V$
2. $||\lambda v|| = |\lambda|||v||$ för alla $\lambda \in \mathbb{R}$ och alla v i V
3. $||u + v|| \leq ||u|| + ||v||$ för alla u och $v \in V$ (triangel olikheten)

Ett rum med en norm kallas ett normerat rum.

3.3.2 Definition 3.8 1-norm

1-normen till vektorn $\mathbf{v} = (v_1, ..., v_n) \in \mathbb{R}^n$ definieras

$$||\mathbf{v}||_1 = \sum_{i=1}^{n} |v_i| \tag{3.13}$$

3.3.3 Definition 3.9 2-norm

2-normen till vektorn $\mathbf{v} = (v_1, ..., v_n) \in \mathbb{R}^n$ definieras

$$||\mathbf{v}||_2 = \left\{ \sum_{i=1}^{n} |v_i|^2 \right\}^{\frac{1}{2}} \tag{3.14}$$

3.3.4 Definition 3.10 ∞-norm

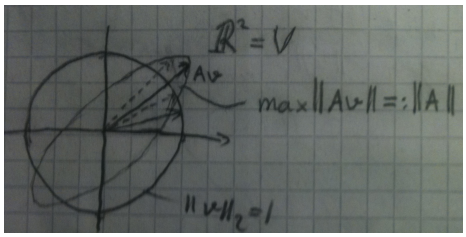
∞-normen till vektorn $\mathbf{v} = (v_1, ..., v_n) \in \mathbb{R}^n$ definieras

$$||\mathbf{v}||_\infty = \max_{i=1}^{n} |v_i| \tag{3.15}$$

3.3.5 Definition 3.11 Matrisnorm

Är vektornormen känd kan motsvarande matrisnorm härledas ur

$$||A|| = \max_{\mathbf{v} \neq 0} \frac{||A\mathbf{v}||}{||\mathbf{v}||} \tag{3.16}$$



Figur 3.2: Geometrisk tolkning av härledningen av 2-matrisnormen utgående från en vektor v av längd 1

3.3.6 Definition 3.12 1-matrisnorm

$$||A||_1 = \max_j \sum_{i=1}^{n} |a_{ij}| = max\ absolut\ kolonnsumma. \tag{3.17}$$

3.3.7 Definition 3.13 2-matrisnorm

$$||A||_2 = \max \sqrt{\lambda(A^T A)} \tag{3.18}$$

Där $\lambda(A^T A)$ är setet med alla egenvärden till $A^T A$.
Speciellt om A symmetrisk ($A^T = A$) fås

$$||A||_2 = \max \sqrt{\lambda(A)^2} = max|\lambda(A)| \tag{3.19}$$

3.3.8 Definition 3.14 ∞-matrisnorm

$$||A||_\infty = \max_i \sum_{j=1}^{n} |a_{ij}| = max\ absolut\ radsumma. \tag{3.20}$$

3.3.9 Definition 3.15 Residual

Låt x_c vara en ungefärlig lösning till ekvationssystemet $Ax = b$ då kallas vektorn

$$r = b - Ax_c \tag{3.21}$$

residualen till lösningen.

På motsvarande sätt kallas $g(x_c) - x_c$ residualen till fixpunktsproblemet.

3.3.10 Definition 3.16 bakåtfel - framåtfel

Bakåtfelet är normen till residualen, för ett fixpunktsproblem fås felet ur:

$$||g(x_c) - x_c|| \tag{3.22}$$

Och för ett ekvationssystem motsvarande:

$$||b - Ax_c||_\infty \tag{3.23}$$

Det relativa bakåtfelet fås ur

$$\frac{||b - Ax_c||_\infty}{||b||_\infty} \tag{3.24}$$

Framåtfelet är normen till $x - x_c$ alltså

$$||x - x_c||_\infty \tag{3.25}$$

Det relativa framåtfelet fås ur

$$\frac{||x - x_c||_\infty}{||x||_\infty} \tag{3.26}$$

.

3.3.11 Definition 3.17 Konditionstal

Konditionstalet till A_{nxn} är den maximala felförstärkningsfaktorn som är möjlig vid lösning av $Ax = b$ över alla högersidor b. Konditionstalet definieras som $cond(A) = ||A|| \cdot ||A^{-1}||$ för (ickesingulära) matriser.

Vid LU faktorisering $(Ax = b \Longrightarrow [A = LU] \Longrightarrow LUx = b)$ fås konditionstalet K ur:

$$\underbrace{\frac{||x - \hat{x}||}{||x||}}_{\text{Relativ störning in}} \leq K \underbrace{\frac{||b - \hat{b}||}{||b||}}_{\text{Relativ störning ut}} \tag{3.27}$$

där approximationen $\hat{b} = b + \Delta b$ och Δb är störningen (perturbation).

3.3.12 Sats 3.5

Konditionstal är ett mått på hur svårt ett problem är att lösa. Ett problem som är lätt att lösa med god nogrannhet har lågt konditionstal. Olösliga problem har oändligt konditionstal. Om $cond(A) >> 1$ så kallas systemmatrisen för dåligt konditionerad.

3.4 Iterativa metoder

3.4.1 Definition 3.19 Planrotations matrisen

Antag att $n \geq 2$, $1 \leq p < q \leq n$ och $\varphi \in [-\pi, \pi]$. Matrisen $R^{(pq)}(\varphi) \in \mathbb{R}^{nxn}_{sym}$ vars element är samma som de i enhetsmatrisen $I \in \mathbb{R}^{nxn}$, förutom

$$\begin{aligned} r_{pp} &= c \quad r_{pq} = s \\ r_{qp} &= -s \quad r_{qq} = -c \end{aligned} \tag{3.28}$$

där $c = \cos(\varphi), \quad s = \sin(\varphi)$.

3.4.2 Definition 3.21 Jacobi matrisen (DF(x))

Matrisen

$$DF(x) = \begin{pmatrix} \frac{\partial f_1}{\partial u} & \frac{\partial f_1}{\partial v} & \frac{\partial f_1}{\partial w} \\ \frac{\partial f_2}{\partial u} & \frac{\partial f_2}{\partial v} & \frac{\partial f_2}{\partial w} \\ \frac{\partial f_3}{\partial u} & \frac{\partial f_3}{\partial v} & \frac{\partial f_3}{\partial w} \end{pmatrix} \tag{3.29}$$

kallas för Jacobi matrisen

3.4.3 Definition 3.21 Jacobi iteration

Låt $A \in \mathbb{R}^{nxn}_{sym}$ och sätt $A^{(0)} = A$. Givet ett $k \leq 0$och $A^{(k)} \in \mathbb{R}^{nxn}_{sym}$, räknar steget i Jacobi metoden ut $A^{(k+1)} \in \mathbb{R}^{nxn}_{sym}$ genom att först finna det största absolutvärdet i icke-diagonalelementen $(A^{(k)})_{pq} = a^k_{pq}$ till matrisen $A^{(k)}$ och sätta $A^{(k+1)} = R^{(pq)}(\varphi_k)^T A^{(k)} R^{(pq)}(\varphi_k)$ med φ_k vald så att $(A^{(k+1)})_{pq} = 0$. Detta upprepas till alla icke-diagonalelement är mindre än toleransen ϵ.

För begynnelse värdet x_0 definieras Jacobimetoden som

$$x_{k+1} = D^{-1}(b - (L + U)x_k)) \tag{3.30}$$

för k=0,1,2,..

3.4.4 Kod 3.2 Jacobi matrisen

```
function J=new_jac(F,Fx,x,h)
    h=size(x,1);
    n=length(F);
    E=eye(n);
for i=1:n
    J(i,:)=(F(x+E(i,:)*h)-Fx)/h
end
```

3.4.5 Kod 3.3 Jacobi iteration

```
%Program 3.2 Jacobi method
%Input: full or sparse matrix A, right hand side: b, number of Jacobi iterations: k
%Output: solution x
function x=jacobi(A,b,k)
n=length(b); % find n
d=diag(A); %take the diagnoal of A
r=A-diag(d); % remainder, r
x=zeros(n,1); %initialize vector x
for j=1:k %Jacobi iteration loop
  x= (b-r*x)./d
end
```

3.5 Olineära system

3.5.1 Definition 3.22 Newtons metod

Låt x_0 vara en begynnelse vektor till systemet. Då defineras Newtons metod

$$x_{k+1} = x_k - (DF(x_k))^{-1}F(x_k) \tag{3.31}$$

för k=0,1,2,....

3.5.2 Kod 3.4 Newtons metod (och förenklad)

```
%Program 3.3 Newton's method
%Input: ,initial vektor x0,
%Output: solution x
function [xst, flag]=newton_nD(F,x0, tol)
x=x0; %Set to initial vector
flag=1;
for i=1:100
  Fx=F(x);
  J=newJac(F,Fx,x,1E-8); %Place outside loop for simplified Newton's method
  dx=-J\Fx;
  x=x+dx;
  if abs(dx)<tol %rätt?
    flag=0;
    break;
  end
xst=x;
end
```

Kapitel 4

Interpolation

Polynominterpolation behövs för databehandling och till konstruktionen av andra nummeriska metoder. Interpolation går till på så vis att man till ett visst antal datapunkter anpassar ett polynom.

Funktionen $y = P(x)$ interpolerar datapunkterna $(x_1, y_1), ..., (x_n, y_n)$ om $P(x_i) = y_i$ för varje $1 \leq i \leq n$.

4.1 Lagrange interpolation

4.1.1 Definition 4.1 Lagrange polynom

Lagrangepolynomet defineras av

$$L_k(x) = \prod_{i=0, i \neq k}^{n} \frac{x - x_i}{x_k - x_i} \tag{4.1}$$

4.1.2 Definition 4.2 Lagrange interpolation

Till varje set data punkter $(x_1, y_1), ..., (x_n, y_n)$, där inga x_j är lika, kan ett interpolationspolynom av ordning n skrivas på Lagrangeform som lineärkombinationen

$$p_n(x) = \sum_{k=1}^{n} y_k L_k(x) \tag{4.2}$$

4.1.3 Kod 4.1 Lagrange interpolation

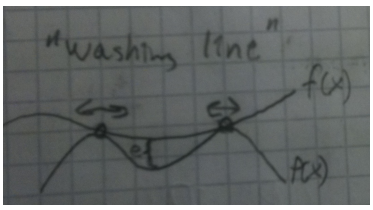
```
function y=lagrange(x,pointx,pointy)
%Input: Points pointx,pointy to interpolate and x to find corresponding y values to.
%Output: y points
n=size(pointx,2);
L=ones(n);
if (size(pointx,2)~=size(pointy,2))
   fprintf(1,'\nERROR!\nPOINTX and POINTY must have the same number of elements\n');
   y=NaN;
else
```

```
   for i=1:n
      for j=1:n
         if (i~=j)
            L(i)=L(i).*(x-pointx(j))/(pointx(i)-pointx(j));
         end
      end
   end
   y=0;
   for i=1:n
      y=y+pointy(i)*L(i);
   end
end
```

4.1.4 Definition 4.3 Chebyshev polynomet

Chebyshev polynom kan användas för att hitta optimala interpolationspunkter till Lagrange interpolationen. Chebyshev polynomet T_n av ordning n defineras för $x \in [-1, 1]$ av

$$T_n(x) = cos(n \cdot arcos(x)) \tag{4.3}$$



Figur 4.1: Geometrisk tolkning av varför interpolationspunkters placering kan påverka felet e mellan interpolation $P(x)$ och verklig kurva $f(x)$

4.1.5 Sats 4.1 Chebyshev interpolationsnoder

På intervallet $[a, b]$ ges Chebyshev noderna av

$$x_i = \frac{b+a}{2} + \frac{b-a}{2} cos \frac{(2i-1)\pi}{2n} \tag{4.4}$$

4.1.6 Sats 4.2 Chebyshev punkter i Lagrangeinterpolation

När vi fått fram Chebychev noderna (x_i) matar vi in dem i den givna funktionen $f(x)$ och får så y-värden $y_i = f(x_i)$ till x och y anpassas sedan ett polynom av grad $n-1$, där n är antalet interpolationsnoder. Detta polynom är anpassat för att man ska slippa stora fel vid ändpunkterna (Runges fenomen).

4.1.7 Definition 4.4 Vandermondematrisen

$$\underbrace{\overbrace{\begin{pmatrix} x_0^n & \dots & x_0^1 & 1 \\ \vdots & \dots & \vdots & \vdots \\ x_i^n & \dots & x_i^1 & 1 \\ \vdots & \dots & \vdots & \vdots \\ x_n^n & \dots & x_n^1 & 1 \end{pmatrix}}^{V} \overbrace{\begin{pmatrix} a_n \\ a_{n-1} \\ \vdots \\ \vdots \\ a_0 \end{pmatrix}}^{a}}_{\begin{cases} a_n x_0^n + a_{n-1} x_0^{n-1} + ... + a_1 x_0^1 + a_0 = y_0 \\ \vdots \\ a_n x_i^n + a_{n-1} x_i^{n-1} + ... + a_1 x_i^1 + a_0 = y_i \\ \vdots \\ a_n x_n^n + a_{n-1} x_n^{n-1} + ... + a_1 x_n^1 + a_0 = y_n \end{cases}} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ \vdots \\ a_n \end{pmatrix} \tag{4.5}$$

4.1.8 Definition 4.5 Vandermondeinterpolation

$Va = y$ där V är Vandermondematrisen löses genom att ta ut $a = V \backslash y$

Kapitel 5

Polynom approximation

5.1 Lineära splines

5.1.1 Definition 5.1 Lineär spline

Antag att f är en realvärd funktion definerad och kontinuerlig på $[a, b]$. Låt $K = \{x_0, x_1, ..., x_m\}$ vara en delmängd av $[a, b]$, med $a = x_0 < x_1 < ... < x_m = b,\ m \geq 2$. En lineär spline s_L interpolerar f i punkterna x_i defineras av

$$s_L(x) = \frac{x_i - x}{x_i - x_{i-1}} f(x_{i-1}) + \frac{x - x_{i-1}}{x_i - x_{i-1}} f(x_i) \tag{5.1}$$

$x \in [x_{i-1}, x_i],\ \ i = 1, 2, ..., m.$
Punkterna x_i kallas knutar (knots) till splinen och K är setet av knutar.

5.1.2 Kod 5.1 Piecewise linear spline

5.2 Kubiska splines

5.2.1 Definition 5.1 Kubisk spline

En funktion $s \in C^2[a, b]$ kallas en kubisk spline på $[a, b]$, om det finns ett kubiskt polynom s_i i varje intervall $[x_i, x_{i+1}]$. Den kallas en kubiskt interpolerande spline om $s(x_i) = y_i$, för givna y_i.

5.2.2 Definition 5.2 Naturlig spline

En kubisk spline kallas för naturlig om den börjar och slutar i 0:

$$s''(x_0) = s''(x_m) = 0 \tag{5.2}$$

5.2.3 Kod 5.2 Cubicspline

```
function [ coeff ] = myCspline( x,y )
%Input an array x and array [y].
%Assume diff(x)= const.
%Output polynomial coefficients.
```

```
m = length(x);
if m==length(y)&& m>=4
  h=x(2)-x(1);
  %Construct the diagonal matrix A for A*sigma=6/(h^2)*diff(diff(y))
  %Natural:
  row=[4 1 zeros(1,m-4)];
  A=toeplitz(row);
  %h=diff(x); %define the deltas in h
  %not necessary since delta x is constant.
  HL=6/(h^2)*diff(diff(y));% diffdiff(y) gives m-1 elements
  sigma= A\HL';
  %Natural:
  sigma=[0 sigma' 0];
  for i=1:m-1
    coeff(1,i)=(sigma(i+1)-sigma(i))/(6*h);%a(i)
    coeff(2,i)=sigma(i)/2;%b(i)
    coeff(3,i)=(y(i+1)-y(i))/h-h/6*(2*sigma(i)+sigma(i+1));%c(i)
    coeff(4,i)=y(i);%d(i)
  end
  else
    disp('Vectors must be of the same length');
    coeff=0;
  end
end
```

5.2.4 Definition 5.2 Basis-spline

Alla splines är kombinationer av B-splines. Fördelen med B-splines är att de bara har lokalt inflytande.

$$F(x) = \sum U_i B_i(x) \tag{5.3}$$

Där U_i är förändringen och B_i den lokala påverkan i form av ett polynom.

Bas: $B_0(x), B_1(x), ..., B_k(x)$

- B_i^1, styckvis konstant
- B_i^2, styckvis lineär
- B_i^3, styckvis kvadratisk
- B_i^4, styckvis kubisk

Iteration:

$$B_i^1 = N_{i1} = \begin{cases} 0 & \text{om } \xi_i = \xi_{i+1} \\ 1 & \text{om } x \in [\xi_i, xi_{i+1}) \\ 0 & \text{annars} \end{cases} \tag{5.4}$$

$$B_i^k = N_{i,k} = \frac{x - \xi_i}{\xi_{i+k-1} - \xi_i} N_{i,k-1} + \frac{\xi_{i+k} - x}{\xi_{i+k} - \xi_{i+1}} N_{i+i,k-1} \tag{5.5}$$

I iterationen ansätts först B_i^1 med hjälp enligt ovan därefter används den efterföljande ekvationen för B_i^k rekursivt där ξ_i är hjälpkonstruktionens punkter:

$$x_0 = \xi_1, \xi_2, \xi_3, \xi_4 \quad x_1 = \xi_5 \quad \dots \quad \dots \quad x_m = \xi_{m+5}, \xi_{m+6}, \xi_{m+7}, \xi_{m+8}$$

Resultatet av iterationen blir en spline med lokal påverkan på :

$$F(x) = \sum_{i=1}^{m+4} U_i B_i^4(x) \tag{5.6}$$

5.2.5 Kod 5.3 B-Spline

```
function s=bsplines(x,i,k,xi)
%INPUT: Point from x vector, iterations i: 1->length(de Boor point vector),
%       degree k(=4), node vector xi.
%OUTPUT: Spline s
N=0
if k==1
    if xi(i)<=x && x<xi(i+1)
        N=1;
    else
        N=0;
    end
else
    if xi(i+k-1)==xi(i);
        del1=0;
    else
        del1=((x-xi(i))/(xi(i+k-1)-xi(i))*bspline(x,i,k-1,xi);
    end
    if xi(i+k)==xi(i+1)
        del2=0;
    else
        del2=((xi(i+k)-x)/((xi(i+k)-xi(i+1))))*bspline(x,i+1,k-1,xi);
    end
    N=del1+del2;
end
```

Kapitel 6

Minsta kvadrat metoden

6.1 Definition 6.1 Minsta kvadrat metoden

Om vi har mer mätningar än okända så blir Vandermonde systemet överdefinierat. $Ax = b$ har lösning då $b = rang(A)$ (b i kolonnrummet till A). Vi löser $Ax = b$ med avseende på minsta kvadrat genom att ta $Ax - b = r$. Vektorn x som minimerar $||r||_2 = \sqrt{r^T r}$ kallas minsta kvadrat lösningen till $Ax = b$.

Kapitel 7

Numerisk integrering

Problemet att nummeriskt approximera $\int_a^b f(x)dx$ kan lösas med olika metoder, vardera med olika effektivitet med avseende på fel och hastighet.

7.1 Newton-Cotes formel

Tanken med Newton-Cotes metod för integration är att approximera funktionen som ska integreras med dess Lagrange interpolation eftersom denna är lättare att integrera. Newton-Cotes metoder av grad n har precision upp till grad $n+1$ för udda och n för jämna n (se även definition 7.9).

7.1.1 Definition 7.1 Newton-Cotes formel

$\int_a^b f(x)dx \approx \int_a^b p_n(x)dx$ Där p_n är lagrange-interpolationspolynomet (fås ur definition 4.2). Då fås den approximerade integralen

$$I_a^b(f) = \sum_{k=0}^{n} w_k f(x_k) \tag{7.1}$$

där vikten w_k fås ur

$$w_k = \int_a^b L_k(x)dx \tag{7.2}$$

för $k = 0, 1, ..., n$.

7.1.2 Definition 7.2 Trapetsregeln

$$\int_a^b f(x)dx \approx (b-a)\left(\frac{1}{2}f(a) + \frac{1}{2}f(b)\right) \tag{7.3}$$

7.1.3 Feluppskattning, Trapetsregeln

Felet i approximationen fås ur

$$|E_n(f)| \leq \frac{M_{n+1}}{(n+1)!}\int_a^b |(x-x_o)...(x-x_n)|dx \tag{7.4}$$

till

$$|E_1(f)| \leq \frac{(b-a)^3}{12} M_2 \tag{7.5}$$

7.1.4 Definition 7.3 Mittpunktsregeln

$$\int_a^b f(x)dx \approx (b-a) f(\frac{a+b}{2}) \tag{7.6}$$

7.1.5 Definition 7.4 Simpsons regel

$$\int_a^b f(x)dx \approx (b-a) \left(\frac{1}{6} f(a) + \frac{4}{6} f(\frac{b+a}{2}) + \frac{1}{6} f(b) \right) \tag{7.7}$$

7.1.6 Feluppskattning, Simpsons regel

Felet i approximationen fås ur

$$|E_n(f)| \leq \frac{M_{n+1}}{(n+1)!} \int_a^b |(x-x_o)...(x-x_n)|dx \tag{7.8}$$

till

$$|E_2(f)| \leq \frac{(b-a)^4}{196} M_3 \tag{7.9}$$

eller förbättrat till

$$|E_2(f)| \leq \frac{(b-a)^5}{2880} M_4 \tag{7.10}$$

7.2 Komposit regel (Composite rules)

För att Newton-Cotes ska ge ett resultat som är nära verkligheten så måste stegen (h) vara små, alltså intervallet $[a, b]$ måste vara litet. Detta stämmer oftast inte direkt utan man måste dela upp $[a, b]$ i delintervall och sedan applicera Newton-Cotes på dessa delintervall. Resultaten från de olika delintervallen summeras sedan ihop till ett resultat för hela intervallet. Detta kallas för enkomposit regel.

7.2.1 Definition 7.5 Komposit regeln

Intervallet $[a, b]$ delas upp i m-stycken delintervall med bredd $h = (b-a)/m$ så att

$$\int_a^b f(x)dx = \sum_{i=1}^{m} \int_{x_{i-1}}^{x_i} f(x)dx \tag{7.11}$$

7.2.2 Definition 7.6 Komposit Trapetsregel

$$\int_a^b f(x)dx \approx h \left[\frac{1}{2} f(x_0) + f(x_1) + ... + f(x_{m-1}) + \frac{1}{2} f(x_m) \right] \tag{7.12}$$

7.2.3 Definition 7.7 Komposit Simpsons regel

$$\int_a^b f(x)dx \approx \frac{h}{3}\left[f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + ... + 2f(x_{2m-2}) + 4f(x_{2m-1}) + f(x_{2m})\right] \tag{7.13}$$

7.2.4 Kod 7.1 Komposit Simpsons metod

```
function I = simpsonsMethod(f,a,b,n)
%INPUT: function f, intervall [a,b], iterations n.
%OUTPUT: Integral value I
h = (b-a)/n;
x = linspace(a,b,2*n+1);
w = zeros(1,2*n+1);
w(1) = 1/6;
w(2:2:2*n+1) = 4/6;
w(3:2:2*n-1) = 2/6;
w(2*n+1) = 1/6;
I = h*sum(f(x).*w);
```

7.3 Gaussintegration

7.3.1 Definition 7.8 Gaussintegration

På nytt utgår vi från grunden som användes för att ta fram Newton-Cotes:

$$\int_{-1}^{1} f(x)dx \approx \sum_{i=1}^{n} c_i f(x_i) \tag{7.14}$$

Om man låter vikten vara Lagrangebaserad:

$$\left(c_i = \int_{-1}^{1} L_i(x)dx \quad i = 1, ..., n\right) \tag{7.15}$$

och sätter x med hjälp av Legendrepolynom, det vill säga låter x-punkterna vara rötterna till Legendrepolynomet fås Gauss-Legendre integration (se definition 7.10).

7.3.2 Kod 7.2 3-stegs Gauss integration

```
function I = mygauss3(f,a,b,n)
%INPUT: function f, intervall [a,b], knots n.
%OUTPUT: Integral value I
h = (b-a)/n;
left = linspace(a,b,n+1);
left = left(1:n);
d1 = h*(1/2-sqrt(15)/10);
d2 = h/2;
d3 = h*(1/2+sqrt(15)/10);
Iinterval = h*(5/18*f(left+d1) + 8/18*f(left + d2) + 5/18*f(left+d3));
I = sum(Iinterval);
```

7.3.3 Definition 7.9 Legendrepolynom

Legendrepolynom är ortogonala på [-1,1].

$$p_i(x) = \frac{1}{2^i i!}\frac{d^i}{dx^i}[(x^2-1)^i], \quad 0 \leq i \leq n \left(\Longrightarrow p_0 = 1, p_1 = x, p_2 = \frac{1}{2}(3x^2-1), p_3 = \frac{1}{2}(5x^3-3x) \right) \tag{7.16}$$

7.3.4 Definition 7.10 Gauss-Legendre integration

För att få högsta möjliga precision, ordning kan vi ta x-punkterna till integrationsmetoden från ett ortogonalt polynom. Det vill säga polynomen p_i med

$$\int_{-1}^{1} p_i(x)p_j(x)dx = \begin{cases} 1 \text{ om } i = j \\ 0 \text{ annars} \end{cases} \tag{7.17}$$

Evaluationspunkterna: rötterna till Legendrepolynomet, $p_n(1)_{normaliserad} = 1$ är x_i från:

$$\int_{-1}^{1} f(x)dx \approx \sum_{i=1}^{n} c_i f(x_i) = \sum_{i=1}^{n} \left(\int_{-1}^{1} c_i(x)dx \right) f(x_i) \tag{7.18}$$

Om gränserna på integralen är a, b så görs ett intervallbyte (Byta $f(x)dx \rightarrow f(\frac{(b-a)t+b+a}{2})\frac{b-a}{2}dt$ där $t = \frac{2x-a-b}{b-a}$).

7.3.5 Kod 7.3 Gauss-Legendre integration

```
function I=gausslege(f,a,b,n)
%INPUT: function f, intervall [a, b] and truncation order N.
%Aproximates integral using Gauss-Legendre quadrature method

%Legendre polynomial
p=polegende(n);
%Legendre roots
x=roots(p(n+1,:));
%Change of integration variable if it's needed
if a~=-1 | b~=1
   y=flege(f,a,b);
   G=subs(y,x);
else
   G=feval(f,x);
end

%Polynomial derivate
pn=polyder(p(n+1,:));

%Calculate the coeficients
for i=1:n
   C(i)=2./((1-x(i).^2).*((polyval(pn,x(i))).^2));
end
```

```
%Scalar product of the function at the nodes and the coeficients
I=dot(C,G);
end

function p=polegende(n)
% p=polegend(n)
% Saves on the rows of the p matrix the coeficients of the legendre polynomial.
p(1,1)=1;
p(2,1:2)=[1 0];
for k=2:n
   p(k+1,1:k+1)=((2*k-1)*[p(k,1:k) 0]-(k-1)*[0 0 p(k-1,1:k-1)])/k;
end
end

function y=flege(f,a,b)
%Performs variable change if a=!-1 y b=!1
syms x;
x=((b-a)./2).*x+(b+a)./2;
dx=(b-a)./2;
y=feval(f,x)*dx;
end
```

7.4 Integrationsmetoders ordning

7.4.1 Definition 7.11 Integrationsmetoders ordning

När steget $h \longrightarrow 0$ minskar felet med h^q där q kallas metodens ordning. En metod av ordning q är exakt upp till och med grad $q-1$.

Simpsons regel har $q = 4$, Trapetregeln har $q = 2$.

7.4.2 Sats 7.1 Kostnad integrations metoder

Kostnaden för en integrationsmetod är relaterad till antalet funktionsevalueringar och dess nogrannheten är relaterad till ordningen. När man väljer metod gör man en avvägning mellan dessa två.

Kapitel 8

Differentialekvationer

Begynnelsevärdesproblemet $y'(t) = f(t, y(t))$ med $y(t_0) = y_0$ löses numeriskt genom användning av någon approximerande metod. I det här kapitlet presenteras några sådana metoder. Problemet $y'(t) = f(t, y(t))$ kan vidare utökas till att omfatta system av differentialekvationer, till exempel av andra ordningen typ

$$\begin{array}{rcl} y'' & = & f(t, y(t), y'(t)) \\ y(t_0) & = & y_0 \\ y'(t_0) & = & y'_0 \end{array} \tag{8.1}$$

Detta problem återförs till första ordningen genom ansatsen

$$\begin{array}{rcl} y' & = & w \\ w' & = & f(t, y(t), w) \end{array} \tag{8.2}$$

Specialfall fås i tre olika former

- då $f(t, y(t)) = f(t)$, vilken enkelt löses genom integrering av vänster- och högerled.
- då $f(t, y(t)) = f(y(t))$ vilken är en autonom differentialekvation.
- då $x'(t) = Ax(t) + By(t)$ för $y(t) = cx(t)$, $x(t_0) = x_0$, $(A_{nxn}, x(t) \in \mathbb{R}^n)$ vilken är en lineär ODE.

För att lösa en differentialekvation måste vi först diskretisera det kontinuerliga tidsintervall i vilket differentialekvationen har sitt förlopp: $t_0, t_1, ..., \underbrace{t_i, t_{i+1}}_{h_i}, ..., t_e$.

Vi kallar den approximerade lösningen i punkten i för u_i och den tillhörande exakta lösningen $y_i = y(t_i)$

8.1 Enkelstegsmetoder

De två enkelstegsmetoderna som vi tar upp kallas implicit- och explicit Euler. Dessa tas fram genom följande resonemang: Givet att $y(x_0) = y_0$, antag att vi

redan räknat ut y_n där $0 \leq n \leq N-1$ och $N \geq 1$

$$y'(t) = f(t, y(t)) \Longrightarrow \frac{y(t+h) - y(t)}{h} \approx \begin{cases} y'(t) & \text{Framåt differentierad} \\ y'(t+h) & \text{Bakåt differentierad} \end{cases} \tag{8.3}$$

Ur dessa fås approximationerna för explicit Euler:

$$\frac{u_{i+1} - u_i}{h_i} = f(t_i, u_i) \Longrightarrow u_{i+1} = u_i + h_i f(t_i, u_i) \tag{8.4}$$

och implicit Euler:

$$\frac{u_{i+1} - u_i}{h_i} = f(t_{i+1}, u_{i+1}) \Longrightarrow u_{i+1} = u_i + h_i f(t_{i+1}, u_{i+1}) \tag{8.5}$$

Där skillnaden är att den implicita varianten har den okända u_{i+1} i både höger- och vänsterled.

8.1.1 Definition 8.1 Explicit Euler

$$\frac{u_{i+1} - u_i}{h_i} = f(t_i, u_i) \Longrightarrow u_{i+1} = u_i + h_i f(t_i, u_i) \tag{8.6}$$

Initialisera först genom att ansätta u_0 iterera därefter u_i:

$$\begin{aligned} u_0 &= y_0 = y(t_0) \\ u_{i+1} &= u_i + h \underbrace{f(t_i, u_i)}_{u_i'}, \quad i = 0, 1, ..., N-1 \end{aligned} \tag{8.7}$$

8.1.2 Kod 8.1 Explicit Euler

```
function [t,y] = expeuler(f,tint,y0,h)
%   [T,Y] = EXPEULER(F,TINT,Y0,H) integrates the system of ODEs defined
%       by the function F. F must be a function handle with two input
%       arguments F(t,y) where t is the time and y is the state vector. It
%       must also return a column vector containg the derivatives of the
%       state variables. TINT is the interval of integration [T0 TEND], Y0
%       is a column vector with the initial states and H is the time step
%       length to be used in the integration. The outputs are T as a column
%       vector containing time points and Y is a matrix with the states
%       as columns.

t = tint(1):h:tint(2);  % The time vector (note the tint(2) may be lost)
N = length(t);          % Nbr of elements in t
M = length(y0);         % Nbr of states
y = zeros(M,N);         % Matrix to store computed states
y(:,1) = y0;            % Initial state

for n = 1:N-1
    y(:,n+1) = y(:,n)+h*f(t(n),y(:,n));
end

% Perform a last step if tint(2) was lost in t %
```

```
if t(N)<tint(2)
    t(N+1) = tint(2);
    h      = t(N+1)-t(N);
    y(:,N+1) = y(:,N)+h*f(t(N),y(:,N));
end

% Transpose for agreement with outputs from Matlabs ODE-solvers %
y = y';
t = t';
```

8.1.3 Definition 8.2 Implicit Euler

$$\frac{u_{i+1}-u_i}{h_i} = f(t_{i+1},u_{i+1}) \Longrightarrow u_{i+1} = u_i + h_i f(t_{i+1},u_{i+1}) \tag{8.8}$$

Initialisera först genom att ansätta u_0 iterera därefter u_i:

$$\begin{aligned} u_0 &= y_0 = y(t_0) \\ u_{i+1} &= u_i + h\underbrace{f(t_{i+1},u_{i+1})}_{u'_{i+1}}, \quad i = 0,1,...,N-1 \end{aligned} \tag{8.9}$$

u'_{i+1} Löses för u_{i+1} med FPI eller Newtons metod. Detta gör implicit Euler mer tidskrävande än den explicita. Dock är den implicita mer stabil när man löser stiff-ekvationer, detta medger användning av en större steglängd (h).

8.1.4 Kod 8.2 Implicit Euler med FPI

```
function [t,y] = impeuler_fpi(f,tint,y0,h)
%   [T,Y] = IMPEULER_FPI(F,TINT,Y0,H) integrates the system of ODEs defined
%       by the function F. F must be a function handle with two input
%       arguments F(t,y) where t is the time and y is the state vector. It
%       must also return a column vector containg the derivatives of the
%       state variables. TINT is the interval of integration [T0 TEND], Y0
%       is a column vector with the initial states and H is the time step
%       length to be used in the integration. The outputs are T as a column
%       vector containing time points and Y is a matrix with the states
%       as columns.
t = tint(1):h:tint(2);  % The time vector (note the tint(2) may be lost)
N = length(t);          % Nbr of elements in t
M = length(y0);         % Nbr of states
y = zeros(M,N);         % Matrix to store computed states
y(:,1) = y0;            % Initial state

for n = 1:N-1
    G  = @(x) y(:,n)+h*f(t(n+1),x);
    x0 = y(:,n)+h*f(t(n),y(:,n));   % Explicit euler predictor to compute
                                    % starting point in FPI
    y(:,n+1) = fpi(G,x0);           % Perform FPI to update states.
end

% Perform a last step if tint(2) was lost in t %
```

```
if t(N)<tint(2)
    t(N+1) = tint(2);
    h      = t(N+1)-t(N);
    G  = @(x) y(:,N)+h*f(t(N+1),x);
    x0 = y(:,N)+h*f(t(N),y(:,N));
    y(:,N+1) = fpi(G,x0);
end

% Transpose for agreement with outputs from Matlabs ODE-solvers %
y = y';
t = t';

function x = fpi(G,x0)
%X = FPI(G,X0) performes fixed point iteration on the function G with
%the starting point X0. X is the computed fixed point.
TOL     = 1e-9; % Tolerance on step length
MAXITER = 30;   % Maximal Nbr of iterations

% Perform the first iteration %
xold    = x0;
x       = G(xold);
iter    = 1;

% Perform the rest %
while norm(x-xold)>TOL && iter<MAXITER
    xold = x;
    x    = G(xold);
    iter = iter+1;
end
```

8.1.5 Definition 8.3 Stiff equation

En ekvation kallas för stiff (på svenska: stel eller styv) då den har:

- Hög dämpning och Höga frekvenser
- En systemmatris A vars egenvärden har negativa realdelar ($\Re(\lambda) < 0$ för alla λ) med stor storleks skillnad mellan den största och minsta negativa realdelen ($\Re_{max}(\lambda_i) >>> \Re_{min}(\lambda_j)$).

8.1.6 Sats 8.1 Stabilitet hos Eulers metod

Det kontinuerliga fallet med testekvationen $y'(t) = \lambda y(t)$ är stabilt om:

$$\begin{matrix} \lambda \in \mathbb{C} \\ \Re(\lambda) < 0 \end{matrix} \tag{8.10}$$

Diskretisering av kontinuerliga fallet ger explicit Euler :

$$u_{i+1} = u_i + \lambda h u_i = (1 + \lambda h) u_i \tag{8.11}$$

Lösningen är stabil om $|1 + h\lambda| < 1$

Diskretisering av kontinuerliga fallet ger även implicit Euler :

$$\begin{aligned} u_{i+1} &= u_i + \lambda h u_{i+1} \\ &\Longrightarrow u_{i+1}(1-\lambda h) = u_i \\ &\Longrightarrow u_{i+1} = \tfrac{u_i}{(1-\lambda h)} \end{aligned} \tag{8.12}$$

Lösningen är stabil om $\frac{1}{|1+h\lambda|} < 1$

8.1.7 Sats 8.2 Precision och fel med Eulers metoder

Det globala felet fås ur $e_n = y(x_n) - y_n$.

Den Lokala residualen eller Trunkeringsfelet fås ur $T_n = \frac{y(x_{n+1})-y(x_n)}{h} - f(x_n, y(x_n))$

8.1.8 Sats 8.3 Val av Eulermetod

Beroende på vilken typ av differentialekvation som ska approximeras finns en tummregel för vilken Lösningsmetod som passar:

	Enkelstegsmetod	Flerstegsmetod
Stiff ODE	Implicit Euler med Newtoniteration	BDF
Nonstiff ODE	Implicit Euler med FPI eller Explicit Euler	Adams metoder

8.2 Flerstegsmetoder

Skillnaden mellan flerstegs- och enkelstegsmetoder är att flerstegs metoderna approximerar differentialekvationen med hjälp av flera av punkterna i det diskretiserade tidsintervall i vilket differentialekvationen har sitt förlopp: $t_0, t_1, ..., t_i, t_{i+1}, ..., t_e$.

8.2.1 Initiera en flerstegsmetod

En flerstegsmetod initieras enligt:

- Ansätt begynnelsevärde
- Ansätt en enkelstegsmetod
- Ansätt en två-stegs metod
- ...

8.2.2 Definition 8.4 Adams-Bashforths metoder (Explicit)

Generell form

$$u_{n+1} = u_n + h\sum_{i=1}^{k} \beta_{k-i} f(t_{n+1-i}, u_{n+1-i}) \tag{8.13}$$

Andra ordningens metod (Tvåstegsmetod)

$$u_{i+1} = u_i + h\left[\frac{3}{2}f(t_i, u_i) - \frac{1}{2}f(t_{i-1}, u_{i-1})\right] \tag{8.14}$$

Tredje ordningens metod (Trestegsmetod)

$$u_{i+1} = u_i + \frac{h}{12}\left[23f(t_i, u_i) - 16f(t_{i-1}, u_{i-1}) + 5f(t_{i-2}, u_{i-2})\right] \tag{8.15}$$

Fjärde ordningens metod (Fyrstegsmetod)

$$u_{i+1} = u_i + \frac{h}{24}\left[55f(t_i, u_i) - 59f(t_{i-1}, u_{i-1}) + 37f(t_{i-2}, u_{i-2}) - 9f(t_{i-3}, u_{i-3})\right] \tag{8.16}$$

8.2.3 Definition 8.5 Adams-Moultons metoder (Implicit)

Adams-Moultons metod är implicit eftersom man i den använder värdet i t_{i+1} trots att det är okänt.

Generell form

$$u_{n+1} = u_n + h\sum_{i=0}^{k}\beta_{k-i}f(t_{n+1-i}, u_{n+1-i}) \tag{8.17}$$

Första ordningens metod (k=0)

$$u_{i+1} = u_i + hf(t_{i+1}, u_{i+1})$$

(Implicit Euler)

Andra ordningens metod (k=1)

$$u_{i+1} = u_i + \tfrac{h}{2}\left[f(t_{i+1}, u_{i+1}) + f(t_i, u_i)\right]$$

(Trapetsregeln)

Tredje ordningens metod (Tvåstegsmetod)(k=2)

$$u_{i+1} = u_i + \frac{h}{12}\left[5f(t_{i+1}, u_{i+1}) + 8f(t_i, u_i) - f(t_{i-1}, u_{i-1})\right] \tag{8.18}$$

Fjärde ordningens metod (Trestegsmetod)

$$u_{i+1} = u_i + \frac{h}{24}\left[9f(t_{i+1}, u_{i+1}) - 19f(t_i, u_i) - 5f(t_{i-1}, u_{i-1}) + f(t_{i-2}, u_{i-2})\right] \tag{8.19}$$

Femte ordningens metod

$$u_{i+1} = u_i + \frac{h}{720}\left[251f(t_{i+1}, u_{i+1}) - 646f(t_i, u_i) - 264f(t_{i-1}, u_{i-1}) + 106f(t_{i-2}, u_{i-2}) - 19f(t_{i-3}, u_{i-3})\right] \tag{8.20}$$

8.2.4 Definition 8.7 Område med absolut stabilitet

Området med absolut stabilitet hörande till en lineär flerstegsmetod är det set av alla punkter λh i det komplexa talplanet för vilken metoden är absolut stabil, det vill säga där varje rot till stabilitetspolynomet π: $z_r = z_r(\lambda h) < 1$ (se Süli s. 347).

8.2.5 Definition 8.7 A-stabil

En lineär flerstegsmetod kallas A-stabil om dess område av absolut stabilitet innehåller vänstra sidan av det komplexa talplanet.

8.2.6 Definition 8.6 Backwards differential formula (BDF)

BDF metoden konstrueras direkt från differentialekvationen. Man söker ett polynom som löser ODE i en punkt t_{n+1} och interpolerar k tidigare punkter. Använd den generella lineära k-stegs metoden (som även trivialt kan ge Adams-Bashforth och - Moulton

$$\sum_{j=0}^{k} \alpha_j y_{n+j} = h \sum_{j=0}^{k} \beta_j f(x_{n+j}, y_{n+j}) \tag{8.21}$$

För BDF metoderna gäller då följande:
$\beta_j = 0$
$0 \leq j \leq k-1$
$k \geq 1$
$\beta_k \neq 0$. Dessa villkor ger en metod som ser ut enligt:

$$\alpha_k y_{n+k} + ... + \alpha_0 y_n = h\beta_k f_{n+k} \tag{8.22}$$

Koefficienterna fås där efter ur 12.47 Süli genom att sätta $C_j = 0$ för $j = 0, 1, ..., k$ För k=1,2,3,4,5 fås metoderna nedan:

BDF1, $k = 1$: fås Implicit Euler.

BDF2, $k = 2$: $3y_{n+2} - 4y_{n+1} + y_n = 2hf_{n+2}$

BDF3, $k = 3$: $11y_{n+3} - 18y_{n+2} + 9y_{n+1} - 2y_n = 6hf_{n+3}$

BDF4, $k = 4$: $25y_{n+4} - 48y_{n+3} + 36y_{n+2} - 16y_{n+1} + 3y_n = 12hf_{n+4}$

BDF5, $k = 5$: $137y_{n+5} - 300y_{n+4} + 300y_{n+3} - 200y_{n+2} + 75y_{n+1} - 12y_n = 60hf_{n+5}$

BDF7, $k = 7$: $\Longrightarrow$ Området med absolut stabilitet kollapsar!

Kapitel 9

Appendix. Tidseffektivitet

Tidseffektivitet för kodalgoritmer mäts i enheten flop (floating point operations per second). En flop motsvarar en addition och en multiplikation. Notationen som används kallas ordonotation ($O(n)$) och visar hur många flops en algoritm kräver beroende på antal element, n.

9.1 Ekvationssystem

9.1.1 LU-faktorisering

Vid lösning av ekvationssystem med hjälp av LU faktorisering utförs följande steg:

1. $A = LU$ själva faktoriseringen, $O(n^3)flops$
2. $L \underbrace{Ux}_{y} = b$ framsubstitution, $O(n^2)flops$
3. $Ux = y$ återsubstitution, $O(n^2)flops$

Matlabkommandot $x = A\backslash b$ utför de nödvändiga av dessa steg (vid triangulär A behövs inte alla steg) för att lösa systemet

Kapitel 10

Appendix. Hjälpsatser

10.1 Medelvärdessatsen

Antag att f är en realvärd deriverbar funktion som är definerad och kontinuerlig på ett intervall $[a, b]$. då finns ett $\xi \in (a, b)$ så att

$$f(b) - f(a) = f'(\xi)(b - a). \tag{10.1}$$

10.2 Symmetrisk matris

En matris A kallas symmetrisk om $A^T = A$.

10.3 Ortogonalmatris

En matris A kallas för ortogonal om $A^T A = I$, det vill säga om transponatet till A är en invers till A.

10.4 Överbestämt system

Ett system som har fler ekvationer än okända kallas överbestämt.

10.5 Kod tips:

10.5.1 for- eller while-loop

For-loopen är antagligen att föredra då den avslutas även om man glömt implementera ett villkor för avslut. Om man glömmer implementera räknaren i while fås en oändlig loop.

10.5.2 Kod 10.1 :Numerisk approximation av funktionsderivata

```
fprime_numerical=@(x) (f(x+1.e-8)-f(x))/(1.e-8)
```

10.5.3 Ortogonalt polynom

Setet av funktioner $\{p_0, ..., p_n\} \forall p_i \neq 0$ på intervallet $[a, b]$ är ortogonalt på $[a, b]$ om

$$\int_a^b p_j(x)p_k(x)dx = \begin{cases} & 0 \text{ om } j \neq k \\ \neq & 0 \text{ om } j = k \end{cases} \tag{10.2}$$

Kapitel 11

Assignments

11.1 Assignment 7, pendel:

Pendelns ekvation fås ur:

$$f'(t) = \frac{-g}{l} sin(\alpha) \tag{11.1}$$

Denna differentialekvation ska lösas med hjälp av BDF-2 Skapa ett interpolations polynom Finn ett nollställe till $f(t) = p(t) - u_{obstacle}$

Val: BDF-2

- Hur starta $\longrightarrow$ begynnelsevärde 1-stegs metod, Impl.-/ Expl.- Euler.
- Hur rätta till $\longrightarrow$ FPI, Lätt att implementera, eller Newton iteration (kräver bestämning av jacobianen).

Polynomial

- varför polynom och inte spline
- Hur många interpolationspunkter $\longrightarrow$ Var restriktiv med punkter för att undvika Runges-Fenomen, 3 punkter.
- Vilken metod $\longrightarrow$ polyfit eller Lagrange (av lathet).

Finn nollställen:

- Hur ser vi konvergens