

Copenhagen Buisness School

NAD 2016

Peter Nistrup Lind Larsen

090294 - 2491

## Indholdsfortegnelse

Problemformulering .....	4
Metodeafsnit .....	5
Overbestemte systemer af lineære ligninger.....	5
Mindste kvadraters løsning med 1) ortogonalprojektion og 2) minimering af fejlvektor.....	6
Tilnærmelse af gradient.....	7
Indkredsningsalgoritme og Golden Section .....	8
QR – faktorisering .....	9
Modificeret Gram-Schmidt .....	10
Tilvejebringelse af data i case a).....	12
Tilvejebringelse af data i case b) .....	12
Tilvejebringelse af data i case c).....	12
Test af program .....	13
Test af data dannelsen i case c) .....	13
Test af resultaterne fra case c) .....	14
Test af metode 1).....	16
Test af resultaterne fra metode 1) .....	16
Test af metode 2).....	19
Test af resultaterne fra metode 2) .....	19
Yderligere test af metode 2) .....	21
Test af metode 3).....	22
Vigtige funktioner .....	25
Konklusion.....	32

Bilag .....	33
Øvrige funktioner .....	33
Variabler .....	34
Datafiler.....	35
Brugervejledning .....	36
Rutediagram.....	37
Programstrukturen .....	37
Grundstruktur.cpp .....	38
Gennemkørsel af grundstruktur.cpp .....	41
Program.cpp.....	43

## Problemformulering

Hovedproblemet for denne opgave er bestemmelsen af mindste kvadraters løsning  $\underline{x}^*$  til et overbestemt system af lineære ligninger:  $\underline{A} \underline{x} = \underline{b}$  hvor:

$\underline{A}$  ( $n * m$ ) er koefficientmatricen, hvor  $n$  er antallet af rækker og  $m$  er antallet af søjler.

$\underline{b}$  ( $n * 1$ ) er resultatvektoren, der består af  $n$  antal rækker.

$\underline{x}$  ( $m * 1$ ) er vektoren af ubekendte, der består af  $m$  antal rækker.

Der gælder at  $n > m$  og der opstår derved et ligningssystem, hvor antallet af ligninger er større end antallet af ubekendte, og man får derved et overbestemt system af lineære ligninger.

Bestemmelsen af  $\underline{x}^*$  skal fremkomme efter 3 metoder i programmet, som følge:

$$1) \quad \underline{A} \underline{x} = \underline{b} \Leftrightarrow \underline{A}^T \underline{A} \underline{x}^* = \underline{A}^T \underline{b} \Leftrightarrow \underline{x}^* = (\underline{A}^T \underline{A})^{-1} \underline{A}^T \underline{b}$$

Som er normalligningerne og løses ved Gauss elimination, delvis pivotering og backwards substitution.

$$2) \quad \underline{A} \underline{x} = \underline{b} \Leftrightarrow \underline{R} \underline{x} = \underline{Q}^T \underline{b}$$

Hvor QR-faktoriseringen:  $\underline{A} = \underline{Q} \underline{R}$  er frembragt ved modificeret Gram-Schmidt ortogonalisering af  $\underline{A}$ .  $\underline{R} \underline{x} = \underline{Q}^T \underline{b}$  løses ved backwards substitution og giver derved  $\underline{x}^*$

### 3) Minimering af $f(\underline{x})$

Hvor  $f(\underline{x}) = |\underline{b} - \underline{A} \underline{x}|^2 = (\underline{b} - \underline{A} \underline{x})^T (\underline{b} - \underline{A} \underline{x})$  er en kvadratisk form og dermed en funktion af de  $m$  variable  $\underline{x}^T = (x_1, x_2, \dots, x_m)$  og hvor  $f(\underline{x}') = \min_{\underline{x}} f(\underline{x})$

Minimeringen af  $f(\underline{x})$  skal udføres ved hjælp af en specificeret iterativ optimeringsalgoritme.

Tilvejebringelse af  $n$ ,  $m$ ,  $A$  og  $b$  skal kunne udføres på 3 forskellige måder:

- a) Indlæsning fra en datafil med veldefineret, kendt struktur.
- b) Ved brugerindtastning af alle elementer af matrix  $A$  og vektor  $b$
- c) Fremstilling af matrix  $A$  og vektor  $b$  ud fra brugerspecificeret design af  $Q$ ,  $R$  og  $b$ .

## Metodeafsnit

### Overbestemte systemer af lineære ligninger

Et system af lineære ligninger

$$\underline{A} \underline{x} = \underline{b} \quad \underline{A} (n * m) \quad \underline{x} (m * 1) \quad \underline{b} (n * 1)$$

kaldes overbestemt hvis  $n > m$ : antallet af ligninger er større end antallet af ubekendte.

Hvis  $n = m$  er systemet kvadratisk.

Profilen for disse systemer er følgende

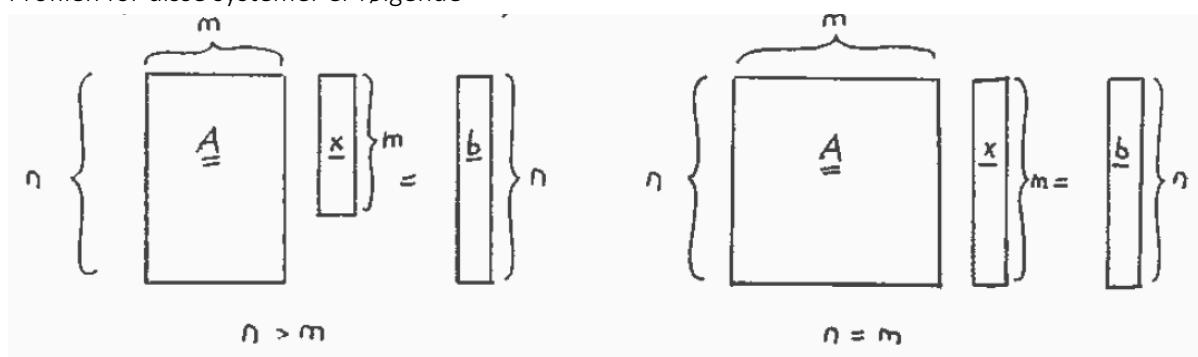


Illustration / eksempel med  $n = 3$   $m = 2$

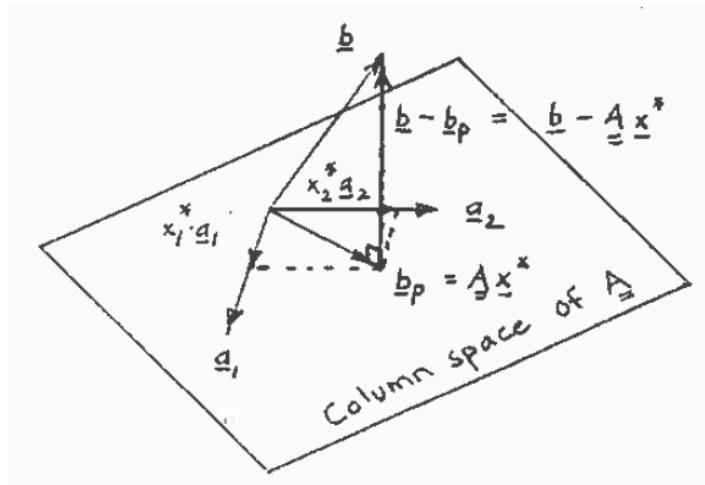
$$\underline{A} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix} \quad \underline{a}_1 = \begin{bmatrix} a_{11} \\ a_{21} \\ a_{31} \end{bmatrix} \quad \underline{a}_2 = \begin{bmatrix} a_{12} \\ a_{22} \\ a_{32} \end{bmatrix} \quad \underline{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \quad \underline{b} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

Problemet skrives som:  $\underline{A} \underline{x} = \underline{b} \quad <=> \quad x_1 * \underline{a}_1 + x_2 * \underline{a}_2 + \dots + x_m * \underline{a}_m = \underline{b}$

Altså skal skalarene  $x_1$  og  $x_2$  findes så den overstående lineære kombination resulterer i højre side vektoren  $\underline{b}$ , problematikken ligger i at  $\underline{A} \underline{x} = \underline{b}$  ikke har nogen løsning hvis  $\underline{b}$  ikke ligger i søjlerummet

$Col \{\underline{A}\}$  til matricen  $\underline{A}$ , og man må i stedet løse for  $\underline{A} \underline{x} = \underline{b}_p$  som resulterer i den ortogonale projktion af  $\underline{b}$  på  $Col \{\underline{A}\}$

Altså  $x_1^* * \underline{a}_1 + x_2^* * \underline{a}_2 + \dots + x_m^* * \underline{a}_m = \underline{b}_p$  hvilket fremgår af følgende skitsering:



Den eksakte løsning  $\underline{x}^*$  til  $\underline{A} \underline{x} = \underline{b}_p$  kaldes den mindste kvadratiske løsning til  $\underline{A} \underline{x} = \underline{b}$

Der er to overordnet metoder der bruges til at finde  $\underline{x}^*$  til  $\underline{A} \underline{x} = \underline{b}_p$ , ortogonal projektion og minimering af længden på fejlvektor, i dette program og denne rapport bruges begge.

Det overbestemte ligningssystem implementeres i C++, på tre forskellige måder.

- Data for A, b, n og m hentes fra en fil.
- Data for A, b, n og m indtastes manuelt af brugeren og kan vælges at blive skrevet til en fil.
- Data til Q dannes henholdsvis ud fra fil og indtastning af n. R indtastes og A beregnes,  $x^*$  og  $t^*$  indtastes.

**Mindste kvadraters løsning med 1) ortogonalprojektion og 2) minimering af fejlvektor**

En linearkombination

$$\underline{A} \underline{x}^* = x_1^* * \underline{a}_1 + x_2^* * \underline{a}_2 + \dots + x_m^* * \underline{a}_m = \underline{b}_p$$

Hvor  $\underline{b}_p$  er den ortogonale projektion af vektoren  $\underline{b}$  i søjlerummet  $Col \{\underline{A}\}$  til matricen  $\underline{A}$ .

Denne linearkombination  $\underline{A} \underline{x}^* = \underline{b}_p$  har 2 vigtige egenskaber der gør det muligt at finde  $\underline{x}^*$

$$1) \quad \underline{A} \underline{t} \perp \underline{b} - \underline{A} \underline{x}^*$$

Hvor  $\underline{t}^T = [t_1, t_2, \dots, t_m]$  så  $\underline{A} \underline{t}$  er en vilkårlig linearkombination af søjler i matrix  $\underline{A}$ , altså en tilfældig vektor der tilhører i søjlerummet  $Col \{\underline{A}\}$

$$\begin{aligned} (\underline{A} \underline{t}) \perp \underline{b} - \underline{A} \underline{x}^* &\iff (\underline{A} \underline{t})^T (\underline{b} - \underline{A} \underline{x}^*) = 0 \\ &\iff \underline{t}^T \underline{A}^T (\underline{b} - \underline{A} \underline{x}^*) = 0 \\ &\iff \underline{t}^T (\underline{A}^T \underline{b} - \underline{A}^T \underline{A} \underline{x}^*) = 0 \\ &\iff \underline{t}^T (\underline{A}^T \underline{b} - \underline{A}^T \underline{A} \underline{x}^*) = 0 \end{aligned}$$

Det fremgår her at eftersom  $\underline{t} (m * 1)$  er en tilfældig vektor med m koordinater og at  $\underline{A}^T \underline{b} - \underline{A}^T \underline{A} \underline{x}^* (m * 1)$  er fast en vektor med m koodinater, må det være nulvektoren. Den mindste kvadraters løsning  $\underline{x}^*$  findes ved følgende normalligning:

$$\begin{aligned} &\iff \underline{A}^T \underline{b} - \underline{A}^T \underline{A} \underline{x}^* = 0 \\ &\iff \underline{A}^T \underline{A} \underline{x}^* = \underline{A}^T \underline{b} \\ &\iff \underline{x}^* = (\underline{A}^T \underline{A})^{-1} \underline{A}^T \underline{b} \end{aligned}$$

Det vides endvidere at ortogonalprojektionen skrives som  $\underline{b}_p = \underline{A} \underline{x}^*$  altså:

$$\begin{aligned} &\iff \underline{b}_p = \underline{A} (\underline{A}^T \underline{A})^{-1} \underline{A}^T \underline{b} \\ &\iff \underline{b}_p = \underline{P} \underline{b} \end{aligned}$$

Hvor  $\underline{P} = \underline{A} (\underline{A}^T \underline{A})^{-1} \underline{A}^T$  er projektionsmatrixen.

$$2) \quad |\underline{b} - \underline{b}_p| = |\underline{b} - \underline{\underline{A}} \underline{x}^*| = \min_{\underline{x}} |\underline{b} - \underline{\underline{A}} \underline{x}|$$

$$|\underline{b} - \underline{b}_p|^2 = |\underline{b} - \underline{\underline{A}} \underline{x}^*|^2 = \min_{\underline{x}} |\underline{b} - \underline{\underline{A}} \underline{x}|^2$$

Eftersom  $|\underline{b} - \underline{\underline{A}} \underline{x}|^2 = (\underline{b} - \underline{\underline{A}} \underline{x})^T (\underline{b} - \underline{\underline{A}} \underline{x})$  skrives det som:

$$(\underline{b} - \underline{\underline{A}} \underline{x}^*)^T (\underline{b} - \underline{\underline{A}} \underline{x}^*) = \min_{\underline{x}} (\underline{b} - \underline{\underline{A}} \underline{x})^T (\underline{b} - \underline{\underline{A}} \underline{x})$$

Fejlvektoren defineres som  $\underline{e} = \underline{b} - \underline{\underline{A}} \underline{x}$  hvor  $\underline{x}^*$  vil minimere  $|\underline{e}|$  og  $|\underline{e}|^2$

Problemet stilles nu op som en funktion af  $f(\underline{x}) = (\underline{b} - \underline{\underline{A}} \underline{x})^T (\underline{b} - \underline{\underline{A}} \underline{x})$  hvor  $\underline{x}^*$  vil give os funktionen  $f(\underline{x})$ 's minimum som:

$$f(\underline{x}^*) = \min_{\underline{x}} f(\underline{x})$$

Vi finder at:

$$\begin{aligned} f(\underline{x}) &= (\underline{b} - \underline{\underline{A}} \underline{x})^T (\underline{b} - \underline{\underline{A}} \underline{x}) \iff \underline{x}^T (\underline{\underline{A}}^T \underline{\underline{A}}) \underline{x} - \underline{x}^T 2\underline{\underline{A}}^T \underline{b} + \underline{b}^T \underline{b} \\ \underline{\nabla} f(\underline{x}) &= 2\underline{\underline{A}}^T \underline{\underline{A}} \underline{x} - 2\underline{\underline{A}}^T \underline{b} \\ \underline{\nabla^2} f(\underline{x}) &= 2\underline{\underline{A}}^T \underline{\underline{A}} \end{aligned}$$

Man ser nu man har normaligningen fra:

$$\underline{\nabla} f(\underline{x}) = 0 \iff \underline{\underline{A}}^T \underline{\underline{A}} \underline{x} = \underline{\underline{A}}^T \underline{b}$$

I dette program bestemmes mindste kvadraters løsning via normaligninger og minimering af  $f(\underline{x})$ .

Ved minimering af  $f(\underline{x})$ , benyttes søgeretningsvektorer, indkredsning og det gyldne snits metode til at lave en tilnærmelse til  $\underline{x}^*$ , hvor  $f(\underline{x}^*)$  er et lokalt minimum for funktionen  $f(\underline{x})$ .

## Tilnærmelse af gradient

Tilnærmelsen af gradienten er ved minimering af fejlvektorens kvadrede længde tidligere udledt, som  $\underline{\nabla} f(\underline{x}) = 2\underline{\underline{A}}^T \underline{\underline{A}} \underline{x} - 2\underline{\underline{A}}^T \underline{b}$ , og den beregnes derved ved via en bestemt matrix  $A$  og vektor  $b$ , hvori  $x$  kan indsættes, og gradienten da kan udregnes. Ved bestemmelse af tilnærmelse af gradienten ved minimering af funktion, udregnes den gradient, der benyttes ifølge med BFGS søgeretningen via den overstående analytiske tilnærmelse, hvor matricen  $A$  og  $b$  er fundet tidligere, og hvor  $x$  da kan indsættes.

### Indkredsningsalgoritme og Golden Section

Det gyldne snit er en iterativ intervalreducerende metode til bestemmelse af en tilnærmelse til  $x^*$ . Den bruges til at bestemme  $\alpha_k$  der benyttes til minimeringen af  $f(x_k + \alpha_k \cdot S_k)$ . Metoden forudsætter, at intervallet  $[a, b]$  er kendt og at der for dette gælder følgende:

- 1)  $f(\alpha)$  er unimodal over  $[a, b]$ , betydende at der kun er et maks./min.
- 2)  $\alpha$  er et indre punkt i  $[a, b]$ , sådan at  $a < \alpha < b$ .

Før det gyldne snit kan benyttes, er det nødvendigt at bestemme et interval hvori minimum befinder sig. Hertil benyttes indkredsningsalgoritmen der netop har til formål at frembringe et startinterval  $[a, b]$  for det gyldne snit. Her indleder brugeren med at indtaste  $\alpha_1$ , hvorom det gælder at  $f'(\alpha_1) < 0$  og har en start-steplængde  $d > 0$ .

Algoritmen bestemmer først  $\alpha_2$  ved at summere  $\alpha_1$  og  $d$  som efterfølges af en løkke der indikerer at så længe  $f(\alpha_2) \geq f(\alpha_1)$ , vil steplængden blive multipliceret med 0,1 og en ny værdi vil blive gemt i  $\alpha_2$ .

Når  $f(\alpha_1) > f(\alpha_2)$  vil løkken da stoppe, grundet at  $\alpha_2$  har passeret  $\alpha_1$  og i stedet fordoble dog derved bestemme  $\alpha_3$  ved overføring af  $\alpha_2 + d$  i dennes værdi.

Efter dette påbegyndes en ny løkke der kører så længe det glæder at  $f(\alpha_2) > f(\alpha_3)$  og som overfører nye værdier i  $\alpha_1, \alpha_2$  og  $\alpha_3$ . Denne løkke afsluttes da når  $f(\alpha_3) > f(\alpha_2)$  da  $f(\alpha_3)$  altså nu må have bevæget sig over  $f(\alpha_2)$ . Herved er der da bestemt et interval  $[a, b]$  hvori minimum for funktionen befinder sig.

Da intervallet nu er bestemt kan der nu benyttes det gyldne snit.

I denne metode sættes det gyldne snits konstant til

$$c = \frac{\sqrt{5}-1}{2} \approx 0,618 \rightarrow 1 - c = \frac{3-\sqrt{5}}{2} \approx 0,382 .$$

Når de fire værdier  $\alpha_1, \alpha_2, \alpha_3, \alpha_4$  optræder gælder:

- 1)  $\alpha_1 < \alpha_2 < \alpha_3 < \alpha_4$
- 2)  $\alpha_2 \approx \alpha_1 + (1 - c)(\alpha_4 - \alpha_1)$  og  $\alpha_3 \approx \alpha_4 - (1 - c)(\alpha_4 - \alpha_1)$

### Indkredsningsalgoritme

```

 $\alpha_2 = \alpha_1 + d$ 
while  $f(\alpha_2) \geq f(\alpha_1)$ 
   $d = 0,1 \cdot d$ 
   $\alpha_2 = \alpha_1 + d$ 
end

```

```

 $d = 2 \cdot d$ 
 $\alpha_3 = \alpha_2 + d$ 

```

```

While  $f(\alpha_2) > f(\alpha_3)$ 
   $\alpha_1 = \alpha_2$ 
   $\alpha_2 = \alpha_3$ 
   $d = 2 \cdot d$ 
   $\alpha_3 = \alpha_2 + d$ 
end

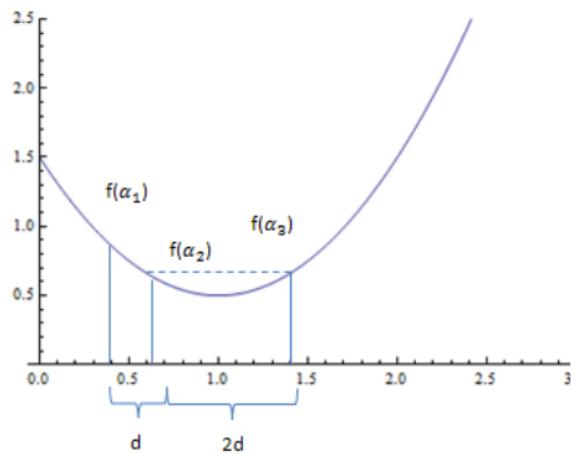
```

```

 $a = \alpha_1$ 
 $b = \alpha_3$ 

```

Indledningsvist overføres intervalendepunkterne  $a$  og  $b$  til  $\alpha_1$  og  $\alpha_4$  hvorefter  $\alpha_2$  og  $\alpha_3$  bestemmes ved brug af konstanten for det gyldne snit. Herefter påbegyndes en løkke der undersøger om  $f(\alpha_2) \geq f(\alpha_3)$ . Såfremt dette gælder, vides det at minimum for funktionen må befinde sig mellem  $\alpha_2$  og  $\alpha_4$ . Såfremt  $f(\alpha_3) \geq f(\alpha_2)$  må minimum da være mellem  $\alpha_1$  og  $\alpha_3$ . Løkken slutter da hvis forskellen mellem  $\alpha_4$  og  $\alpha_1$  bliver mindre end den angivne nul-tolerance,  $\text{eps}$ , og den nye tilnærmede  $x$ -værdi bestemmes, ved at være halvdelen af summen af  $\alpha_1$  og  $\alpha_4$ .

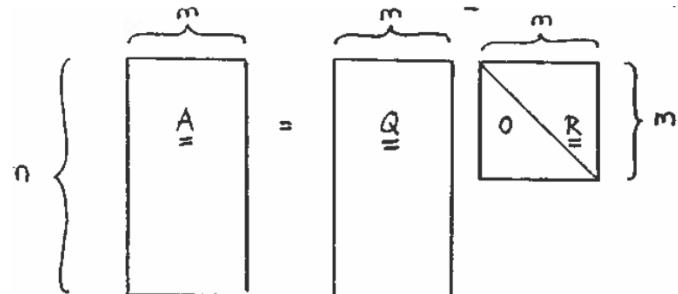


### QR – faktorisering

QR-faktorisering beskrives ud fra følgende:

$\underline{\underline{A}} = \underline{\underline{Q}} \underline{\underline{R}}$  hvor  $\underline{\underline{Q}}$  ( $n * m$ ) har kolonerne  $\underline{q}_1 + \underline{q}_2 + \dots + \underline{q}_m$  som en orto-nomal basis i søjlerummet  $\text{Col}\{\underline{\underline{A}}\}$  for en matrix  $\underline{\underline{A}}$  ( $n * m$ ) hvor det gælder at  $n \geq m$  og hvor  $\underline{\underline{R}}$  ( $m * m$ ) er en øvre trekantsmatrix.

Dette illustreres på følgende figur:



Hvor  $\underline{\underline{Q}} = \underline{\underline{Q}}_1$  og  $\underline{\underline{R}} = \underline{\underline{R}}_1$

Dette betegnes som smal QR – faktorisering hvilket er hvad der benyttes i dette program og rapport.

Det kan herfra ses at  $\underline{x}^*$  kan findes ved:

$$\begin{aligned}
 \underline{\underline{A}} \underline{x} = \underline{b} &\iff \underline{\underline{Q}}_1 \underline{\underline{R}}_1 \underline{x} = \underline{b} \\
 &\iff \underline{\underline{Q}}_1^T \underline{\underline{Q}}_1 \underline{\underline{R}}_1 \underline{x} = \underline{\underline{Q}}_1^T \underline{b} \\
 &\iff \underline{\underline{R}}_1 \underline{x} = \underline{\underline{Q}}_1^T \underline{b} \\
 &\iff \underline{x}^* = \underline{\underline{R}}_1^{-1} \underline{\underline{Q}}_1^T \underline{b}
 \end{aligned}$$

$\underline{x}^*$  er dermed løsningen til  $\underline{R}_1 \underline{x} = \underline{Q}_1^T \underline{b}$

Herfra kræver løsningen at man gør brug af backwards-substitution.

Smal QR-faktorisering kan løses ved Gram-Schmidt og modificeret Gram-Schmidt, i mit program og denne rapport bruges dog kun modificeret Gram-Schmidt.

### Modificeret Gram-Schmidt

Det der gør den modificeret Gram-Schmidt forskellig fra traditionel Gram-Schmidt er at den sikrer ortogonaliteten for den k'ne søjle med de øvrige søjler i den opdateret matrix A, samt at den modificeret Gram-Schmidt danner rækker i R for hver iteration i kontrast til den traditionelle hvori der dannes søjler.

#### The thin QR-factorization by modified Gram-Schmidt orthogonalization

- Purpose

Given:  $\underline{\underline{A}}(n \times m)$  where  $n \geq m$  and  $\text{rank}(\underline{\underline{A}}) = m$  is full,

The columns of  $\underline{\underline{A}}$  are  $\underline{a}_1, \underline{a}_2, \dots, \underline{a}_m$  which is ortho-normal.

Compute:  $\underline{\underline{Q}}(n \times m)$  The columns of  $\underline{\underline{Q}}$  are  $\underline{q}_1, \underline{q}_2, \dots, \underline{q}_m$  which is a set of mutually orthogonal unit vectors forming a basis of  $\text{Col}\{\underline{\underline{A}}\}$

$\underline{\underline{R}}(m \times m)$  which is upper triangular.

so that:  $\underline{\underline{A}} = \underline{\underline{Q}} \cdot \underline{\underline{R}}$  is a thin QR-factorization of  $\underline{\underline{A}}$

$$\underline{\underline{A}} = \underline{\underline{Q}} \cdot \underline{\underline{R}}$$

$$n \left\{ \begin{array}{|c|} \hline \underline{a}_1 & \dots & \underline{a}_m \\ \hline \end{array} \right. \underbrace{\quad}_{m} = \left[ \begin{array}{|c|} \hline \underline{q}_1 & \dots & \underline{q}_m \\ \hline \end{array} \right] \underbrace{\quad}_{m} \cdot \left[ \begin{array}{|c|} \hline \text{---} & \dots & \text{---} \\ \hline \end{array} \right] \underbrace{\quad}_{m}$$

Hele idéen bag modificeret Gram-Schmidt er at få konstueret en Q og R matrix der opfylder  $A = QR$ , gennem m-antal iterationer.

Hvor iterationsindeksene  $k = 1, 2, \dots, m$

Eftersom hver iteration af Gram-Schmidt transformerer den give matrix A kopieres denne over i en ny matrix,  $\underline{\underline{A}}_0$  før transformationen startes.

Strukturelt opbygget som følgende:

Søjlerne for  $\underline{\underline{A}} : \underline{a}_1, \underline{a}_2, \dots, \underline{a}_m$

Søjlerne for  $\underline{\underline{A}}_0 : \underline{a}_1^{(0)}, \underline{a}_2^{(0)}, \dots, \underline{a}_m^{(0)}$

Hvor

$$\underline{a}_1^{(1)} = \underline{a}_1$$

$$\underline{a}_2^{(2)} = \underline{a}_2 - r_{12} * \underline{q}_1$$

$$\underline{a}_3^{(3)} = \underline{a}_3 - r_{13} * \underline{q}_1 - r_{23} * \underline{q}_2$$

...

$$\underline{a}_m^{(m)} = \underline{a}_m - r_{1m} * \underline{q}_1 - r_{2m} * \underline{q}_2 - \dots - r_{mm} * \underline{q}_m$$

$$\underline{a}_1 = r_{11} * \underline{q}_1$$

$$\underline{a}_2 = r_{12} * \underline{q}_1 + r_{23} * \underline{q}_2$$

$$\underline{a}_3 = r_{13} * \underline{q}_1 + r_{23} * \underline{q}_2 + r_{33} * \underline{q}_3$$

...

$$\underline{a}_m = r_{1m} * \underline{q}_1 + r_{2m} * \underline{q}_2 + \dots + r_{mm} * \underline{q}_m$$

For k = 0,

Sæt  $\underline{\underline{A}}_0 = \underline{\underline{A}}$  med søjlerne  $\underline{a}_1^{(0)}, \underline{a}_2^{(0)}, \dots, \underline{a}_m^{(0)}$

For k = 1,

Udregn  $r_{11} = \|\underline{a}_1^{(0)}\|$

$$\underline{q}_1 = \frac{1}{r_{11}} * \underline{a}_1^{(0)}$$

Udregn  $\underline{\underline{A}}_1$  ved transformation af  $\underline{\underline{A}}_0$

Søjleopdatering  $\underline{a}_j^{(1)} = \underline{a}_j^{(0)} - (\underline{q}_1^T * \underline{a}_j^{(0)}) * \underline{q}_1$   $j = 2, \dots, m$

Gem  $r_{1j} = \underline{q}_1^T * \underline{a}_j^{(0)}$   $j = 2, \dots, m$

Søjleopdatering  $\underline{a}_1^{(1)} = \underline{a}_1^{(0)}$  (Søjle 1, kopi)

For k = m,

Udregn  $r_{mm} = \|\underline{a}_m^{(m-1)}\|$

$$\underline{q}_m = \frac{1}{r_{mm}} * \underline{a}_m^{(m-1)}$$

Udregn  $\underline{\underline{A}}_m$  ved transformation af  $\underline{\underline{A}}_{m-1}$

Søjleopdatering  $\underline{a}_m^{(m)} = \underline{a}_m^{(m-1)}$

Status  $\underline{q}_m$  er udregnet for  $\underline{\underline{Q}}$

$r_{mm}$  er udregnet for  $\underline{\underline{R}}$

$\underline{a}_m^{(m)} = r_{mm} * \underline{q}_m$  er sat i  $\underline{\underline{A}}_m$

### Tilvejebringelse af data i case a)

Hvis brugeren vælger at gøre brug at case a) vil data til matrix A og vektor b bliver indlæst fra en datafil 'HentFil.txt' på følgende format:

<b>n</b>					
<b>m</b>					
<b>a<sub>11</sub></b>	<b>a<sub>12</sub></b>	<b>a<sub>13</sub></b>	.....	<b>a<sub>1m</sub></b>	<b>b<sub>1</sub></b>
<b>a<sub>21</sub></b>	<b>a<sub>22</sub></b>	<b>a<sub>23</sub></b>	.....	<b>a<sub>2m</sub></b>	<b>b<sub>2</sub></b>
.	.	.		.	.
.	.	.		.	.
<b>a<sub>n1</sub></b>	<b>a<sub>n2</sub></b>	<b>a<sub>n3</sub></b>	.....	<b>a<sub>nm</sub></b>	<b>b<sub>n</sub></b>

### Tilvejebringelse af data i case b)

Hvis brugeren vælger at gøre brug at case b) vil data til matrix A og vektor b blive indtastet af brugeren selv, startende med indtastning af antal n-rækker og antal m-søjler hvor m < n, herefter indtastes matrix et element af gangen række for række.

Når matricen er udfyldt vil den blive udskrevet så brugeren kan se om der skal fortages rettelser i et eller flere elementer. Herefter indtastes vektor b og efter indtastning udskrives den og der bliver igen spurgt til rettelser. Til sidst bliver bruger spurgt om data skal gemmes til en fil.

### Tilvejebringelse af data i case c)

Hvis brugeren vælger at gøre brug at case b) vil data til matrix A og vektor b blive udregnet gennem en række funktioner, her startes med et brugervalgt n-rækker der frembringer An matricen udtrukket fra datafil 'DelC.txt' hvorefter matricen Dn dannes, eksempelvis hvis n = 4:  
D<sub>4</sub>(4x4):  $d_k = \frac{1}{|a_{kk}|}$  for  $k = 1, \dots, 4$  hvor  $|a_{kk}|$  er kvadratroden af søjlens indre værdi. Herefter indtastes det ønskede antal søjler m, og matricen Q dannes ud fra Dn og An. Brugeren taster så en ønsket R matrix som sammen med den fundne Q matrix danner matricen A.

Vektoren b findes efterfølgende ved indtastning af x\* samt t\* ved at finde ortogonalprojektionen bp og fejlvektoren e og trække dem fra hinanden.

## Test af program

Test af data dannelse i case c)

Vælg mellem a, b eller c: **c**

Du valgte c.

På dette sted indtastes n, m og R manuelt.

Frembringer matricen Q og beregner A = Q \* R.  
Indtastning x\* og t\*, derefter beregning bp, e og b

Angiv om du vil have (4) eller (8) rækker, n = **4**

← Her indtaster brugeren det ønskede antal rækker n.

Matrix A4 er givet ved:

1.00000	1.00000	1.00000	0.00000
1.00000	-1.00000	0.00000	1.00000
1.00000	1.00000	-1.00000	0.00000
1.00000	-1.00000	0.00000	-1.00000

Matrix D4 er givet ved:

0.50000	0.00000	0.00000	0.00000
0.00000	0.50000	0.00000	0.00000
0.00000	0.00000	0.70711	0.00000
0.00000	0.00000	0.00000	0.70711

Angiv antallet af søger hvor m < n, m = **2**

← Her indtaster brugeren det ønskede antal søger m.

Matrix Q er givet ved:

0.50000	0.50000
0.50000	-0.50000
0.50000	0.50000
0.50000	-0.50000

Angiv R's elementer:

Angiv elementet [1,1] = **2**

← Herfra indtaster brugeren de ønskede værdier til matricen R samt vektorerne x\* og t\*

Angiv elementet [1,2] = **1**

Angiv elementet [2,2] = **3**

Angiv elementerne i x\*

Angiv 1. element: **2**

Angiv 2. element: **1**

Angiv elementerne i t\*

Angiv 1. element: **3**

Angiv 2. element: **2**

Ortogonalprojektionen bp er givet ved:

[4.00000, 1.00000, 4.00000, 1.00000]

← Herfra udskrives henholdsvis

ortogonalprojektionen bp,  
fejlvektoren e, den dannede  
matrix A og vektor b, som under  
tjekkes i Mathematica.

Fejlvektoren e er givet ved:

[3.00000, 2.00000, -3.00000, -2.00000]

Matrix A er givet ved:

1.00000	2.00000
1.00000	-1.00000
1.00000	2.00000
1.00000	-1.00000

Vektor b er givet ved:

[7.00000, 3.00000, 1.00000, -1.00000]

Test af resultaterne fra case c)

```
ClearAll["Global`*"]
```

### Test af case c)

Matrix A4 er givet ud fra datafil 'DelC.txt' med n = 4 og matrix R er indtastet af bruger til:

$$\mathbf{A4} = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & -1 & 0 & 1 \\ 1 & 1 & -1 & 0 \\ 1 & -1 & 0 & -1 \end{pmatrix}; \quad \mathbf{R} = \begin{pmatrix} 2 & 1 \\ 0 & 3 \end{pmatrix};$$

Matrix D4 regnes:

```
d1 = Transpose[{A4[[All, 1]]}];
d2 = Transpose[{A4[[All, 2]]}];
d3 = Transpose[{A4[[All, 3]]}];
d4 = Transpose[{A4[[All, 4]]}];
```

$$\mathbf{D4} = \begin{pmatrix} 1 / \text{Norm}[d1] & 0 & 0 & 0 \\ 0 & 1 / \text{Norm}[d2] & 0 & 0 \\ 0 & 0 & 1 / \text{Norm}[d3] & 0 \\ 0 & 0 & 0 & 1 / \text{Norm}[d4] \end{pmatrix} // \text{N}$$

Den beregnede D4 matrix bliver:

$$\begin{pmatrix} 0.5 & 0. & 0. & 0. \\ 0. & 0.5 & 0. & 0. \\ 0. & 0. & 0.707107 & 0. \\ 0. & 0. & 0. & 0.707107 \end{pmatrix}$$

Matrix S4 regnes:

```
S4 = A4 . D4
```

$$\begin{pmatrix} 0.5 & 0.5 & 0.707107 & 0. \\ 0.5 & -0.5 & 0. & 0.707107 \\ 0.5 & 0.5 & -0.707107 & 0. \\ 0.5 & -0.5 & 0. & -0.707107 \end{pmatrix}$$

```
Q = Join[Transpose[{S4[[All, 1]]}], Transpose[{S4[[All, 2]]}], 2]
```

Den beregnede Q matrix bliver:

$$\begin{pmatrix} 0.5 & 0.5 \\ 0.5 & -0.5 \\ 0.5 & 0.5 \\ 0.5 & -0.5 \end{pmatrix}$$

Matrix A regnes til:

```
A = Q . R
```

$$\begin{pmatrix} 1. & 2. \\ 1. & -1. \\ 1. & 2. \\ 1. & -1. \end{pmatrix}$$

Bruger indtaster  $x^*$  til:

$$\mathbf{x}^* = \begin{pmatrix} 2 \\ 1 \end{pmatrix};$$

Orthogonalprojektionen  $b_p$  regnes til:

$$\mathbf{b}_p = \mathbf{A} \cdot \mathbf{x}^*$$

$$\begin{pmatrix} 4. \\ 1. \\ 4. \\ 1. \end{pmatrix}$$

Bruger indtaster  $t^*$  til:

$$\mathbf{t}^* = \begin{pmatrix} 3 \\ 2 \end{pmatrix};$$

Matrix  $C$  regnes til:

$$\mathbf{C} = \text{Join}[\text{Transpose}[\{\mathbf{A}[[\mathbf{A}\mathbf{1}, 3]]\}], \text{Transpose}[\{\mathbf{A}[[\mathbf{A}\mathbf{1}, 4]]\}], 2]$$

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \\ -1 & 0 \\ 0 & -1 \end{pmatrix}$$

Fejlvektoren  $e$  regnes til:

$$\mathbf{e} = \mathbf{C} \cdot \mathbf{t}^*$$

$$\begin{pmatrix} 3 \\ 2 \\ -3 \\ -2 \end{pmatrix}$$

Vektoren  $b$  regnes til:

$$\mathbf{b} = \mathbf{b}_p + \mathbf{e}$$

$$\begin{pmatrix} 7. \\ 3. \\ 1. \\ -1. \end{pmatrix}$$

Det ses altså at case c) har udregnet de ønskede værdier nøjagtigt da:

Orthogonalprojektionen  $b_p$  er givet ved:  
 $[4.00000, 1.00000, 4.00000, 1.00000]$

Fejlvektoren  $e$  er givet ved:  
 $[3.00000, 2.00000, -3.00000, -2.00000]$

Matrix  $A$  er givet ved:  
 $\begin{matrix} 1.00000 & 2.00000 \\ 1.00000 & -1.00000 \\ 1.00000 & 2.00000 \\ 1.00000 & -1.00000 \end{matrix}$

Vektor  $b$  er givet ved:  
 $[7.00000, 3.00000, 1.00000, -1.00000]$

$$\begin{pmatrix} \text{Transpose}[b_p] \\ \text{Transpose}[e] \\ \mathbf{A} \\ \text{Transpose}[b] \end{pmatrix}$$

$$\begin{pmatrix} (4. & 1. & 4. & 1.) \\ (3 & 2 & -3 & -2) \\ \begin{pmatrix} 1. & 2. \\ 1. & -1. \\ 1. & 2. \\ 1. & -1. \end{pmatrix} \\ (7. & 3. & 1. & -1.) \end{pmatrix}$$

## Test af metode 1)

Her har bruger valgt metode 1)

Vælg mellem (1) eller (2/3): 1

Den transponeret matrix AT er givet ved:

$$\begin{matrix} 1.00000 & 1.00000 & 1.00000 & 1.00000 \\ 2.00000 & -1.00000 & 2.00000 & -1.00000 \end{matrix}$$

Matrix produktet ATA er givet ved:

$$\begin{matrix} 4.00000 & 2.00000 \\ 2.00000 & 10.00000 \end{matrix}$$

Matrix/vektor produktet ATb er givet ved:

$$[10.00000, 14.00000]$$

$x^*$  givet ved:

$$[2.00000, 1.00000]$$

hvor:

$$n = 4$$

$$m = 2$$

Matrix A:

$$\begin{matrix} 1.00000 & 2.00000 \\ 1.00000 & -1.00000 \\ 1.00000 & 2.00000 \\ 1.00000 & -1.00000 \end{matrix}$$

Vektor b:

$$[7.00000, 3.00000, 1.00000, -1.00000]$$

## Test af resultaterne fra metode 1)

```
ClearAll["Global`*"]
```

### Test af metode 1)

Matrix A og vektor b er fundet fra tidligere case a), b) eller c) til

$$A = \begin{pmatrix} 1 & 2 \\ 1 & -1 \\ 1 & 2 \\ 1 & -1 \end{pmatrix}; b = \begin{pmatrix} 7 \\ 3 \\ 1 \\ -1 \end{pmatrix};$$

AT = Transpose[A]

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 2 & -1 & 2 & -1 \end{pmatrix}$$

ATA = AT . A

$$\begin{pmatrix} 4 & 2 \\ 2 & 10 \end{pmatrix}$$

ATb = AT . b

$$\begin{pmatrix} 10 \\ 14 \end{pmatrix}$$

$x^* = (\text{Join}[ATA, ATb, 2] // \text{RowReduce}) [[All, 3]]$

{2, 1}

Det ses altså at metode 1) har udregnet de ønskede værdier nøjagtigt.

Jeg vil dog teste med nogle **høje tal, lave tal**, samt en **8x7 matrix** til sidst:

```
ClearAll["Global`*"]
```

### Test af metode I) med store tal

*Matrix A og vektor b er fundet fra tidligere case a), b) eller c) til*

$$A = \begin{pmatrix} 34625 & 234 & 235 & 163 & 463 \\ 34634 & 523 & 324 & 634 & 623 \\ 3657 & 674 & 754 & 756 & 475 \\ 234254 & 636 & 3453 & 245 & 234 \end{pmatrix}; b = \begin{pmatrix} 53252 & 342 \\ 64784 & 354 \\ 234646 & 768 \\ 23412 & 312 \end{pmatrix};$$

**AT = Transpose[A]**

$$\begin{pmatrix} 34625 & 234 & 3657 & 674 & 234254 & 636 \\ 235163 & 463 & 324634 & 623 & 754756 & 475 \\ 3453245 & 234 \end{pmatrix}$$

**ATA = AT . A**

$$\begin{pmatrix} 57287070079575057 & 831085513695859145 \\ 831085513695859145 & 12655249475478644736 \end{pmatrix}$$

**ATb = AT . b**

$$\begin{pmatrix} 10430354009427234 \\ 291503871811412672 \end{pmatrix}$$

```
x* = (Join[ATA, ATb, 2] // RowReduce)[[All, 3]] // N
{-3.21672, 0.23428}
```

```
ClearAll["Global`*"]
```

### Test af metode I) med små tal

*Matrix A og vektor b er fundet fra tidligere case a), b) eller c) til*

$$A = \begin{pmatrix} 0.0000042 & 0.000000345 \\ 0.0000012 & 0.00000064 \\ 0.5 & 0.000000214 \\ 0.00214 & 2 \end{pmatrix}; b = \begin{pmatrix} 0.00024 \\ 0.014 \\ 0.6 \\ 0.00001 \end{pmatrix};$$

**AT = Transpose[A]**

$$\begin{pmatrix} 4.2 \times 10^{-6} & 1.2 \times 10^{-6} & 0.5 & 0.00214 \\ 3.45 \times 10^{-7} & 6.4 \times 10^{-7} & 2.14 \times 10^{-7} & 2 \end{pmatrix}$$

**ATA = AT . A**

$$\begin{pmatrix} 0.250005 & 0.00428011 \\ 0.00428011 & 4 \end{pmatrix}$$

**ATb = AT . b**

$$\begin{pmatrix} 0.3 \\ 0.0000201374 \end{pmatrix}$$

```
x* = (Join[ATA, ATb, 2] // RowReduce)[[All, 3]] // N
{1.2, -0.001279}
```

Vælg mellem (1) eller (2/3): 1

Den transponeret matrix AT er givet ved:  
 $34625234.0000034634523.000003657674.00000234254636.00000$   
 $235163463.00000324634623.00000754756475.000003453245234.00000$

Matrix produktet ATA er givet ved:  
 $57287070079575056.00000831085513695859072.00000$   
 $831085513695859072.0000012655249475478644736.00000$

Matrix/vektor produktet ATb er givet ved:  
 $[10430354009427234.00000, 291503871811412672.00000]$

x\* givet ved:  
 $[-3.21672, 0.23428]$

Vælg mellem (1) eller (2/3): 1

Den transponeret matrix AT er givet ved:  
 $0.00000 \quad 0.00000 \quad 0.50000 \quad 0.00214$   
 $0.00000 \quad 0.00000 \quad 0.00000 \quad 2.00000$

Matrix produktet ATA er givet ved:  
 $0.25000 \quad 0.00428$   
 $0.00428 \quad 4.00000$

Matrix/vektor produktet ATb er givet ved:  
 $[0.30000, 0.00002]$

x\* givet ved:  
 $[1.20000, -0.001279]$

Det ses at der regnes rigtigt både med høje og lave tal, det eneste 'problem' ligger i afrundingen i programmet, det vil nemt kunne kommes til hjælp vil at fjerne den 'setprecision(5)' og 'fixed' funktion der danner matricerne

```
ClearAll["Global`*"]
```

### Test af metode 1) med 8x7

Matrix A og vektor b er fundet fra tidligere case a), b) eller c) til

$$\mathbf{A} = \begin{pmatrix} 5 & 3 & 2 & 4 & 5 & 7 & 1 \\ 8 & 2 & 3 & 1 & 5 & 7 & 2 \\ 3 & 5 & 1 & 2 & 3 & 5 & 7 \\ 2 & 3 & 7 & 2 & 1 & 2 & 8 \\ 8 & 1 & 2 & 9 & 2 & 3 & 6 \\ 2 & 1 & 6 & 4 & 7 & 2 & 1 \\ 2 & 7 & 2 & 3 & 7 & 8 & 4 \\ 2 & 1 & 2 & 7 & 9 & 5 & 6 \end{pmatrix}; \mathbf{b} = \begin{pmatrix} 7 \\ 2 \\ 3 \\ 1 \\ 6 \\ 2 \\ 3 \\ 7 \end{pmatrix};$$

```
AT = Transpose[A]
```

$$\begin{pmatrix} 5 & 8 & 3 & 2 & 8 & 2 & 2 & 2 \\ 3 & 2 & 5 & 3 & 1 & 1 & 7 & 1 \\ 2 & 3 & 1 & 7 & 2 & 6 & 2 & 2 \\ 4 & 1 & 2 & 2 & 9 & 4 & 3 & 7 \\ 5 & 5 & 3 & 1 & 2 & 7 & 7 & 9 \\ 7 & 7 & 5 & 2 & 3 & 2 & 8 & 5 \\ 1 & 2 & 7 & 8 & 6 & 1 & 4 & 6 \end{pmatrix}$$

```
ATA = AT . A
```

$$\begin{pmatrix} 178 & 78 & 87 & 138 & 138 & 164 & 128 \\ 78 & 99 & 62 & 71 & 110 & 132 & 107 \\ 87 & 62 & 111 & 89 & 113 & 98 & 109 \\ 138 & 71 & 89 & 180 & 163 & 143 & 148 \\ 138 & 110 & 113 & 163 & 243 & 208 & 145 \\ 164 & 132 & 98 & 143 & 208 & 229 & 154 \\ 128 & 107 & 109 & 148 & 145 & 154 & 207 \end{pmatrix}$$

```
ATb = AT . b
```

$$\begin{pmatrix} 134 \\ 79 \\ 74 \\ 158 \\ 165 \\ 161 \\ 132 \end{pmatrix}$$

```
x* = (Join[ATA, ATb, 2] // RowReduce)[[All, 8]] // N
```

```
{-0.551081, -0.77479, 0.0797171, 0.914148, -0.501052, 1.39861, -0.00616367}
```

Vælg mellem (1) eller (2/3): 1

Den transponerede matrix AT er givet ved:

$$\begin{matrix} 5.00000 & 8.00000 & 3.00000 & 2.00000 & 8.00000 & 2.00000 & 2.00000 & 2.00000 \\ 3.00000 & 2.00000 & 5.00000 & 3.00000 & 1.00000 & 1.00000 & 7.00000 & 1.00000 \\ 2.00000 & 3.00000 & 1.00000 & 7.00000 & 2.00000 & 6.00000 & 2.00000 & 2.00000 \\ 4.00000 & 1.00000 & 2.00000 & 2.00000 & 9.00000 & 4.00000 & 3.00000 & 7.00000 \\ 5.00000 & 5.00000 & 3.00000 & 1.00000 & 2.00000 & 7.00000 & 7.00000 & 9.00000 \\ 7.00000 & 7.00000 & 5.00000 & 2.00000 & 3.00000 & 2.00000 & 8.00000 & 5.00000 \\ 1.00000 & 2.00000 & 7.00000 & 8.00000 & 6.00000 & 1.00000 & 4.00000 & 6.00000 \end{matrix}$$

Matrix produktet ATA er givet ved:

$$\begin{matrix} 178.00000 & 78.00000 & 87.00000 & 138.00000 & 138.00000 & 164.00000 & 128.00000 \\ 78.00000 & 99.00000 & 62.00000 & 71.00000 & 110.00000 & 132.00000 & 107.00000 \\ 87.00000 & 62.00000 & 111.00000 & 89.00000 & 113.00000 & 98.00000 & 109.00000 \\ 138.00000 & 71.00000 & 89.00000 & 180.00000 & 163.00000 & 143.00000 & 148.00000 \\ 138.00000 & 110.00000 & 113.00000 & 163.00000 & 243.00000 & 208.00000 & 145.00000 \\ 164.00000 & 132.00000 & 98.00000 & 143.00000 & 208.00000 & 229.00000 & 154.00000 \\ 128.00000 & 107.00000 & 109.00000 & 148.00000 & 145.00000 & 154.00000 & 207.00000 \end{matrix}$$

Matrix/vektor produktet ATb er givet ved:

$$[134.00000, 79.00000, 74.00000, 158.00000, 165.00000, 161.00000, 132.00000]$$

x\* givet ved:

$$[-0.55108, -0.77479, 0.07972, 0.91415, -0.50105, 1.39861, -0.00616]$$

Igen ses det at der regnes helt præcist.

## Test af metode 2)

Her har bruger valgt metode 2)

Vælg mellem (1) eller (2/3): 2

Du har valgt 2 eller 3

QR-faktorisering af matrix A er udført ved hjælp af modificeret Gram-Schmidt til:

Matrix A:

$$\begin{matrix} 1.00000 & 2.00000 \\ 1.00000 & -1.00000 \\ 1.00000 & 2.00000 \\ 1.00000 & -1.00000 \end{matrix}$$

Matrix Q:

$$\begin{matrix} 0.50000 & 0.50000 \\ 0.50000 & -0.50000 \\ 0.50000 & 0.50000 \\ 0.50000 & -0.50000 \end{matrix}$$

Matrix R:

$$\begin{matrix} 2.00000 & 1.00000 \\ 0.00000 & 3.00000 \end{matrix}$$

Hvor:

$$\begin{matrix} n = 4 \\ m = 2 \end{matrix}$$

Matrix A:

$$\begin{matrix} 1.00000 & 2.00000 \\ 1.00000 & -1.00000 \\ 1.00000 & 2.00000 \\ 1.00000 & -1.00000 \end{matrix}$$

Vektor b:

$$[7.00000, 3.00000, 1.00000, -1.00000]$$

Du kan nu vælge imellem:

2) Bestemmelse af mindre kvadraters løsningen  $x^*$  ved at løse  $R * x = Q^T * b$ 3) Bestemmelse af mindre kvadraters løsningen  $x^*$  ved minimering af  $f(x) = (b - A * x)^2$ 

Vælg mellem 2 eller 3: 2

Du valgte 2, der løses efter ligningen  $R * x = Q^T * b$  $x^*$  givet ved:  
[2.00000, 1.00000]

## Test af resultaterne fra metode 2)

ClearAll["Global`\*"]

## Test af metode 2)

Matrix A og vektor b er fundet fra tidligere case a), b) eller c) til

$$A = \begin{pmatrix} 1 & 2 \\ 1 & -1 \\ 1 & 2 \\ 1 & -1 \end{pmatrix}; b = \begin{pmatrix} 7 \\ 3 \\ 1 \\ -1 \end{pmatrix},$$

Matrix Q og R defineres ved funktionen 'QR'

QR[x\_] := {{Transpose[QRDecomposition[x][[1]]], QRDecomposition[x][[2]]}}

QR[A] // N

$$\left( \begin{pmatrix} 0.5 & 0.5 \\ 0.5 & -0.5 \\ 0.5 & 0.5 \\ 0.5 & -0.5 \end{pmatrix} \begin{pmatrix} 2. & 1. \\ 0. & 3. \end{pmatrix} \right)$$

$$Q = \begin{pmatrix} 0.5 & 0.5 \\ 0.5 & -0.5 \\ 0.5 & 0.5 \\ 0.5 & -0.5 \end{pmatrix}; R = \begin{pmatrix} 2. & 1. \\ 0. & 3. \end{pmatrix};$$

Herefter bruges formelen  $R * x = Q^T * b$  hvilket giver os følgende  $x^*$ 

T = Join[R, Transpose[Q].b, 2] // RowReduce // N

$$\begin{pmatrix} 1. & 0. & 2. \\ 0. & 1. & 1. \end{pmatrix}$$

x\* = T[[All, 3]]

(2. 1.)

Det ses altså at metode 2) har udregnet de ønskede værdier nøjagtigt med modificeret G-S.

Jeg vil dog teste metode 2) med **høje tal, lave tal**, samt en **8x7 matrix** også:

Du har valgt 2 eller 3

QR-faktorisering af matrix A er udført ved hjælp af modificeret Gram-Schmidt til:

Matrix A:

5.00000	3.00000	2.00000	4.00000	5.00000	7.00000	1.00000
8.00000	2.00000	3.00000	1.00000	5.00000	7.00000	2.00000
3.00000	5.00000	1.00000	2.00000	3.00000	5.00000	7.00000
2.00000	3.00000	7.00000	2.00000	1.00000	2.00000	8.00000
8.00000	1.00000	2.00000	9.00000	2.00000	3.00000	6.00000
2.00000	1.00000	6.00000	4.00000	7.00000	2.00000	1.00000
2.00000	7.00000	2.00000	3.00000	7.00000	8.00000	4.00000
2.00000	1.00000	2.00000	7.00000	9.00000	5.00000	6.00000

Matrix Q:

0.37477	0.10048	-0.09602	0.02606	0.09135	0.60135	-0.47839
0.59963	-0.18701	-0.04602	-0.59698	0.38732	-0.03009	0.20431
0.22486	0.45775	-0.23608	-0.04729	-0.13701	-0.35718	0.44097
0.14991	0.26376	0.67831	-0.17804	-0.47107	0.34079	0.25983
0.59963	-0.31121	-0.12778	0.43023	-0.50289	-0.21720	-0.09957
0.14991	0.01535	0.64423	0.11757	0.31866	-0.51698	-0.35706
0.14991	0.76059	-0.15962	0.10088	0.10699	-0.07398	-0.28566
0.14991	0.01535	0.12646	0.63237	0.48484	0.27173	0.49842

Matrix R:

13.34166	5.84635	6.52093	10.34354	10.34354	12.29232	9.59401
0.00000	8.05110	2.96561	1.30766	6.15172	7.46914	6.32337
0.00000	0.00000	7.72546	2.28757	3.53468	-0.55762	3.58368
0.00000	0.00000	0.00000	8.12824	4.90648	0.90578	3.97347
0.00000	0.00000	0.00000	0.00000	7.84857	4.13230	-3.22332
0.00000	0.00000	0.00000	0.00000	0.00000	1.97572	0.28150
0.00000	0.00000	0.00000	0.00000	nan	nan	5.98914

Du kan nu vælge imellem:

- 2) Bestemmelse af mindre kvadraters løsningen  $x^*$  ved at løse  $R * x = Qt * b$
- 3) Bestemmelse af mindre kvadraters løsningen  $x^*$  ved minimering af  $f(x) = (b - A * x)^2$

Vælg mellem 2 eller 3: 2

Du valgte 2, der løses efter ligningen  $R * x = Qt * b$

$x^*$  givet ved:

```
[-0.55108, -0.77479, 0.07972, 0.91415, -0.50105, 1.39861, -0.00616]
```

Det bemærkes her at Eclipse udskriver 2 værdier som "nan" eller NaN som betyder Not-a-Number, hvilket vil sige Eclipse højst sandsynligt har ramt en beregning hvor der eksempelvis divideres med 0 (eller en tilnærmelse med over 16 decimaler, hvilket er den præcision en data af typen double-precision har), der er fortaget kontrol med mathematica og  $x^*$  er stadig korrekt.

```
T = Join[R, Transpose[Q].b, 2] // RowReduce // N
{{1, 0, 0, 0, 0, 0, -0.551081},
 {0, 1, 0, 0, 0, 0, -0.77479},
 {0, 0, 1, 0, 0, 0, 0.0797171},
 {0, 0, 0, 1, 0, 0, 0.914148},
 {0, 0, 0, 0, 1, 0, -0.501052},
 {0, 0, 0, 0, 0, 1, 1.39861},
 {0, 0, 0, 0, 0, 0, -0.00616367}}
x* = {T[[All, 8]]}
(-0.551081 -0.77479 0.0797171 0.914148 -0.501052 1.39861 -0.00616367)
```

## Yderligere test af metode 2)

Input	Output	Kontrol	Kommentar
<pre>A = ( {     {0.0000042,      0.000000345},     {0.0000012,      0.00000064},     {0.5, 0.000000214},     {0.00214, 2}   }) b = ( {     {0.00024},     {0.014},     {0.6},     {0.00001}   })</pre>	<p>* givet ved:  <math>[1.2, 0.00128]</math></p>	$x^* =$ $(1.2, -0.001279)$	Korrekt udregnet resultat, med undtagelse af relevante decimaltal, afrundingen skyldes to funktioner i koden (setprecision(5) og fixed) hvis funktion ligger i at gøre programmet mere brugervenligt.
<pre>A = ( {     {34625234,      235163463},     {34634523,      324634623},     {3657674, 754756475},     {234254636,      3453245234}   }) b = ( {     {53252342},     {64784354},     {234646768},     {23412312}   })</pre>	<p><math>x^*</math> givet ved:  <math>[-3.21672, 0.23428]</math></p>	$x^* =$ $(-3.21672, 0.23428)$	Resultaterne er helt korrekte.

### Test af metode 3)

Her har bruger valgt metode 3) og QR faktorisering er fortaget før følgende beregninger.

Vælg mellem 2 eller 3: 3

Du valgte 3, der løses efter minimering af  $f(x) = (b - Ax)^2$

Angiv dit startpunkt, x:

Angiv 1. element: 1

Angiv 2. element: 2

Angiv maksimalt antal af iterationer, N = 1000

Angiv tolerancen, eps = 0.0001

Antal iterationer: 2

Vektoren er givet ved:

[2.00000, 1.00000]

Matricen dannet fra  $1/2 * R^{-1} * R^{-1}^T$  er givet ved:

0.13889 -0.02778  
-0.02778 0.05556

Iterationsmatricen  $H(m+1)$  er givet ved:

0.13889 -0.02778  
-0.02778 0.05556

Hvor:

n = 4  
m = 2

Matrix A:

1.00000	2.00000
1.00000	-1.00000
1.00000	2.00000
1.00000	-1.00000

Vektor b:

7.00000	3.00000	1.00000	-1.00000
---------	---------	---------	----------

### Test af metode 3)

Matrix A og vektor b er fundet fra tidligere case a), b) eller c) til

$$A = \begin{pmatrix} 1 & 2 \\ 1 & -1 \\ 1 & 2 \\ 1 & -1 \end{pmatrix}; b = \begin{pmatrix} 7 \\ 3 \\ 1 \\ -1 \end{pmatrix};$$

Matrix R og Q er fundet fra tidligere beregning efter valgt 2) eller 3) til

$$Q = \begin{pmatrix} 0.5 & 0.5 \\ 0.5 & -0.5 \\ 0.5 & 0.5 \\ 0.5 & -0.5 \end{pmatrix}; R = \begin{pmatrix} 2 & 1 \\ 0 & 3 \end{pmatrix};$$

1 / 2 \* Inverse[R].Transpose[Inverse[R]]

$$\begin{pmatrix} 0.138889 & -0.0277778 \\ -0.0277778 & 0.0555556 \end{pmatrix}$$

Her ses  $\frac{1}{2} * R^{-1} * (R^{-1})^T$

Og det ses den er identisk til C++

Inverse[2 \* Transpose[A].A] // N

$$\begin{pmatrix} 0.138889 & -0.0277778 \\ -0.0277778 & 0.0555556 \end{pmatrix}$$

Der tjekkes også direkte efter med  $(2A^T A)^{-1}$  hvilket også er korrekt

I tidligere test afsnit med samme A og b værdier konkluderede jeg også at  $x^*$  er korrekt.  
Jeg fortager yderligere test med en matrix A(8x7) (samme værdier som i 8x7 testen af metode 2)):

Vælg mellem 2 eller 3: 3

Du valgte 3, der løses efter minimering af  $f(x) = (b - Ax)^2$

Angiv dit startpunkt, x:

Angiv 1. element: 1

Angiv 2. element: 2

Angiv 3. element: 3

Angiv 4. element: 4

Angiv 5. element: 5

Angiv 6. element: 6

Angiv 7. element: 7

Angiv maksimalt antal af iterationer, N = 1000

Angiv tolerancen, eps = 0.0001

Antal iterationer: 7

Vektoren er givet ved:

[-0.55108, -0.77479, 0.07972, 0.91415, -0.50105, 1.39861, -0.00616]

Matricen dannet fra  $1/2 * RInv * RInv$  er givet ved:

0.04187	0.03836	-0.01920	-0.02036	0.03626	-0.06577	0.00248
0.03836	0.07323	-0.01797	-0.00711	0.03446	-0.08204	-0.01013
-0.01920	-0.01797	0.02088	0.01080	-0.02243	0.03321	-0.00654
-0.02036	-0.00711	0.01080	0.02416	-0.02400	0.02796	-0.01068
0.03626	0.03446	-0.02243	-0.02400	0.04691	-0.06840	0.00677
-0.06577	-0.08204	0.03321	0.02796	-0.06840	0.12837	-0.00199
0.00248	-0.01013	-0.00654	-0.01068	0.00677	-0.00199	0.01394

Iterationsmatricen  $H(m+1)$  er givet ved:

0.04187	0.03836	-0.01920	-0.02036	0.03626	-0.06577	0.00248
0.03836	0.07323	-0.01797	-0.00711	0.03446	-0.08204	-0.01013
-0.01920	-0.01797	0.02088	0.01080	-0.02243	0.03321	-0.00654
-0.02036	-0.00711	0.01080	0.02416	-0.02400	0.02796	-0.01068
0.03626	0.03446	-0.02243	-0.02400	0.04691	-0.06840	0.00677
-0.06577	-0.08204	0.03321	0.02796	-0.06840	0.12837	-0.00199
0.00248	-0.01013	-0.00654	-0.01068	0.00677	-0.00199	0.01394

$1 / 2 * \text{Inverse}[R].Transpose[\text{Inverse}[R]]$

$$\begin{pmatrix} 0.0418739 & 0.0383644 & -0.0191988 & -0.0203645 & 0.0362607 & -0.0657702 & 0.0024762 \\ 0.0383644 & 0.0732251 & -0.0179685 & -0.0071091 & 0.03446 & -0.0820399 & -0.0101333 \\ -0.0191988 & -0.0179685 & 0.0208769 & 0.0108001 & -0.0224342 & 0.0332066 & -0.00654484 \\ -0.0203645 & -0.0071091 & 0.0108001 & 0.0241635 & -0.024005 & 0.0279568 & -0.0106797 \\ 0.0362607 & 0.03446 & -0.0224342 & -0.024005 & 0.0469127 & -0.0684047 & 0.00677036 \\ -0.0657702 & -0.0820399 & 0.0332066 & 0.0279568 & -0.0684047 & 0.128374 & -0.00198604 \\ 0.0024762 & -0.0101333 & -0.00654484 & -0.0106797 & 0.00677036 & -0.00198604 & 0.0139393 \end{pmatrix}$$

$\text{Inverse}[2 * \text{Transpose}[A].A] // N$

$$\begin{pmatrix} 0.0418739 & 0.0383644 & -0.0191988 & -0.0203645 & 0.0362607 & -0.0657702 & 0.0024762 \\ 0.0383644 & 0.0732251 & -0.0179685 & -0.0071091 & 0.03446 & -0.0820399 & -0.0101333 \\ -0.0191988 & -0.0179685 & 0.0208769 & 0.0108001 & -0.0224342 & 0.0332066 & -0.00654484 \\ -0.0203645 & -0.0071091 & 0.0108001 & 0.0241635 & -0.024005 & 0.0279568 & -0.0106797 \\ 0.0362607 & 0.03446 & -0.0224342 & -0.024005 & 0.0469127 & -0.0684047 & 0.00677036 \\ -0.0657702 & -0.0820399 & 0.0332066 & 0.0279568 & -0.0684047 & 0.128374 & -0.00198604 \\ 0.0024762 & -0.0101333 & -0.00654484 & -0.0106797 & 0.00677036 & -0.00198604 & 0.0139393 \end{pmatrix}$$

$\text{Inverse}[2 * \text{Transpose}[A].A] == 1 / 2 * \text{Inverse}[R].Transpose[\text{Inverse}[R]]$

True

$T = \text{Join}[R, \text{Transpose}[\Omega].b, 2] // \text{RowReduce} // N$

$$\begin{pmatrix} 1. & 0. & 0. & 0. & 0. & 0. & -0.551081 \\ 0. & 1. & 0. & 0. & 0. & 0. & -0.77479 \\ 0. & 0. & 1. & 0. & 0. & 0. & 0.0797171 \\ 0. & 0. & 0. & 1. & 0. & 0. & 0.914148 \\ 0. & 0. & 0. & 0. & 1. & 0. & -0.501052 \\ 0. & 0. & 0. & 0. & 0. & 1. & 1.39861 \\ 0. & 0. & 0. & 0. & 0. & 1. & -0.00616367 \end{pmatrix}$$

$x^* = \{T[[All, 8]]\}$

(-0.551081 -0.77479 0.0797171 0.914148 -0.501052 1.39861 -0.00616367)

Det ses at der regnes helt korrekt. Det er også værd at påpege at antallet af iterationer er korrekte også, da antal iterationer = m og der er valgt en 8x7 matrix altså m = 7, ligeledes var tidligere test fortaget med en 4x2 matrix altså m = 2, hvor antallet af iterationer også var 2. Jeg fortager yderligere **eksterne** test:

Jeg tager udgangspunkt i den før testede 4x2 matrix:

```
n = 4
m = 2

Matrix A:
1.00000 2.00000
1.00000 -1.00000
1.00000 2.00000
1.00000 -1.00000

Vektor b:
[7.00000, 3.00000, 1.00000, -1.00000]
```

Input	Output	Kontrol	Kommentar
Eps = 0.0001 Iterationer = 1000 x = {1, 2}	1/2*Rinv*RinvT 0.13889 -0.02778 -0.02778 0.05556  Iterationsmatrix 0.13889 -0.02778 -0.02778 0.05556	1/2*Rinv*RinvT 0.138889 -0.0277778 -0.0277778 0.0555556  Invers(2ATA) 0.138889 -0.0277778 -0.0277778 0.0555556	Resultaterne er korrekt udregnet.
Eps = 0.0001 Iterationer = 1000 x = {3, 0}	1/2*Rinv*RinvT 0.13889 -0.02778 -0.02778 0.05556  Iterationsmatrix 0.13889 -0.02778 -0.02778 0.05556	1/2*Rinv*RinvT 0.138889 -0.0277778 -0.0277778 0.0555556  Invers(2ATA) 0.138889 -0.0277778 -0.0277778 0.0555556	Resultaterne er korrekt udregnet.
Eps = 0.0001 Iterationer = 1 x = {1, 2}	1/2*Rinv*RinvT 0.13889 -0.02778 -0.02778 0.05556  Iterationsmatrix 1.12409 -0.13148 -0.13148 0.06647	1/2*Rinv*RinvT 0.138889 -0.0277778 -0.0277778 0.0555556  Invers(2ATA) 0.138889 -0.0277778 -0.0277778 0.0555556	Det ses at den inverse hessematrice, altså iterationsmatricen er forkert udregnet grundet at den er blevet begrænset til kun 1 iteration.
Eps = 1 Iterationer = 2 x = {1, 2}	1/2*Rinv*RinvT 0.13889 -0.02778 -0.02778 0.05556  Iterationsmatrix 1.00000 0.00000 0.00000 1.00000	1/2*Rinv*RinvT 0.138889 -0.0277778 -0.0277778 0.0555556  Invers(2ATA) 0.138889 -0.0277778 -0.0277778 0.0555556	Da nultolerencen her er sat til 1 bliver iterationsmatricen helt forkert udregnet.
Jeg sætter herfra eps = 0.0001 og max iterationer = m, GS eps er eps i Golden Section			
Steplængde = 0.1 GS eps = 0.0000001	x* = {2, 1}	x* = {2, 1}	Resultaterne er korrekt udregnet.
Steplængde = 0.5 GS eps = 0.0000001	x* = {2, 1}	x* = {2, 1}	Resultaterne er korrekt udregnet.
Steplængde = 0.1 GS eps = 0.001	x* = {1.99998, 1.00003}	x* = {2, 1}	Resultaterne er altså ikke korrekte længere grundet ændret nultolerance

## Vigtige funktioner

<b>void BeregnAogb(double A[NMAX][NMAX], double b[NMAX], int &amp;n, int &amp;m);</b>	
Formål	At beregne matrix <u>A</u> , vektor <u>b</u> , ortogonalprojektionen <u>b<sub>p</sub></u> og fejlvektoren <u>e</u> ud fra en given An matrix og brugerdefineret R matrix
Input	double M[NMAX][NMAX] fra tekst fil 'DelC.txt'
Output	double A[NMAX][NMAX] og double b[NMAX]
Virkemåde	<p>Funktionen henter matricen A8 ( 8 x 8 ) fra tekstdlingen 'DelC.txt' og bruger angiver om der ønskes 4 x 4 eller 8 x 8 ved indtastning af n, herefter udskrives matricen An så bruger kan se hvad der er valgt.</p> <p>Herefter åbnes en for-løkke der danner matricen Dn bestående af de normaliserede søjlelængde fra matricen A i D matricens diagonal, eksempelvis for <u>D<sub>4</sub>(4x4)</u>: <math>d_k = \frac{1}{ a_k }</math> for <math>k = 1, \dots, 4</math> hvor <math> a_k </math> er kvadratroden af søjlets indre værdi. Dette gøres ved at trække søjlerne fra An matricen med funktionen 'Tagsoejle' og finde den indre værdi med funktionen 'Pprodukt' som herefter sættes som værdi i diagonalen på matricen Dn som udskrives efterfølgende.</p> <p>Bruger taster antallet af ønsket søjler og Q regnes og udskrives efterfølgende ud fra funktionen ProduktMogMT der laver produktet af en matrix og en transponeret matrix, her laves produktet af An og Dn matricerne.</p> <p>Herefter åbnes en for-løkke hvor bruger indtaster den øvre trekantsmatrix R, det fremgår af løkken at hvis rækketallet er lavere end søjletallet sættes værdien til 0, hvilket giver den ønskede øvre trekantsmatrix.</p> <p>Da Q og R nu er dannet kan den ønskede matrix <u>A</u> beregnes ud fra funktionen ProduktMogM for Q og R.</p> <p>Herefter indtaster bruger ønsket værdier for <u>x*</u> ved funktionen Angivv, hvorefter ortogonalprojektionen udreges ud fra funktionen ProduktMogv. Efterfølgende indtastes <u>t*</u> på identisk vis, og en for-løkke åbnes hvor matrix C dannes ud fra rækkerne hvor rækketal&gt;m i An, produktet af den nyfundne matrix C og vektor <u>t*</u> findes ud fra funktionen ProduktMogv. Det bemærkes at beregningen er begrænset til de n-m'de første søjler i den dannede matrix C, hvilket resulterer i den ønskede fejlvektor <u>e</u>.</p> <p>Til sidst findes den ønskede vektor <u>b</u> ved at fejlvektoren adderes til ortogonalprojektionen i den sidste for-løkke, herefter printes de fundne resultater internt i funktionen og matrix <u>A</u> og vektor <u>b</u> er givet som output og printes uden for funktionen.</p>
Afhængigheder	HentM, UdskrivM, Tagsoejle, Pprodukt, ProduktMogMT, ProduktMogM, Angivv, ProduktMogv, Udskrivv

```

void BeregnAogb(double A[NMAX][NMAX], double b[NMAX], int &n, int &m)
{
    double D[NMAX][NMAX], M[NMAX][NMAX], Q[NMAX][NMAX], C[NMAX][NMAX], R[NMAX][NMAX];
    double x[NMAX], v[NMAX], bp[NMAX], t[NMAX], e[NMAX];

    HentM(M, "DelC.txt");
    cout << "\nAngiv om du vil have (4) eller (8) rækker, n = ";
    cin >> n;
    cout << "\nMatrix A" << endl;
    UdskrivM(M, n, n);
}

```

```

for(int i=0;i<n;i++){
    Tagsoejle(M,v,n,i);
    D[i][i]=1/sqr(Pprodukt(v,v,n));
}
cout << "\nMatrix D" << n << " er givet ved: " << endl;
UdskrivM(D,n,n);
cout << "\nAngiv antallet af søjler hvor m < n, m = ";
cin >> m;
ProduktMogMT(D,M,Q,n,n);
cout << "\nMatrix Q er givet ved: " << endl;
UdskrivM(Q,n,m);
cout << "\nAngiv R's elementer:" << endl;
for (int i=0;i<m;i++){
    for (int j=0;j<m;j++){
        if (j>=i){
            cout << "Angiv elementet [" << i+1 << "," << j+1 << "] = ";
            cin >> R[i][j];
        }
        else R[i][j] = 0;
    }
}
ProduktMogM(Q,R,A,n,m);
cout << "\nAngiv elementerne i x*" << endl;
Angivv(x,m);
ProduktMogv(A,x,bp,n,m);
cout << "\nAngiv elementerne i t*" << endl;
Angivv(t,(n-m));
for (int i=0;i<n;i++){
    for (int j=0;j<n;j++){
        if (j>=m) C[i][(j-m)]=M[i][j];
    }
}
ProduktMogv(C,t,e,n,(n-m));
for (int i=0;i<n;i++){
{
    b[i] = bp[i] + e[i];
}
cout << "\nOrentogonalprojektionen bp er givet ved:" << endl;
Udskrивv(bp,n);
cout << "\nFejlvektoren e er givet ved:" << endl;
Udskrивv(e,n);
cout << endl;
}

```

<b>void BackwardsSubstitution(double TM[NMAX][NMAX+1],double x[NMAX],int n);</b>	
Formål	At beregne <u><math>x^*</math></u>
Input	double TM[NMAX][NMAX+1], int n
Output	double x[NMAX]
Virkemåde	<p>Den sidste position i løsningsvektoren findes først da den nederste m-1'te række i matricen er blevet reduceret til, at der kun er pivotpositionen tilbage med nuller på venstre siden. Det nederste element i løsningvektoren findes så ved at dividere det sidste element på højre siden med den sidste pivotposition.</p> <p>Efter man har beregnet værdien af den nederste position i løsningvektoren findes de overstående værdier ved en for-løkke som kører nedefra og op for hver række i matricen på nær m-1'te række hvor vi allerede kender løsningen.</p> <p>Vi starter med en startsum = 0, og en for-løkke der kører for hvert element i rækken. For-løkke ganger x på hvert element og lægger dette til den aktuelle sum, på den måde bliver summen opdateret for hvert element i rækken, indtil der er fundet en samlet sum for rækken. Når samlede sum er fundet for den aktuelle række benyttes den til at finde det aktuelle element i løsningvektoren x,</p>

	ved at tage elementet på højre siden og trække summen herfra, og derefter dividere det med det aktuelle diagonalelement. Herved bliver der i denne løkke fundet en løsning for hver række, der bliver opskrevet i vektoren x.
Afhængigheder	N / A

```
void BackwardsSubstitution(double TM[NMAX][NMAX+1],double x[NMAX],int n)
{
    double sum;

    x[n-1]=TM[n-1][n]/TM[n-1][n-1];
    for(int i=n-2;i>=0;i--)
    {
        sum=0;
        for(int j=i+1;j<n;j++) sum+=TM[i][j]*x[j];
        x[i]=(TM[i][n]-sum)/TM[i][i];
    }
}
```

<b>void DelvisPivoting(double TM[NMAX][NMAX+1],int j,int n);</b>	
Formål	Delvis pivoting indgår som en del af Gauss elimination, og benyttes til finde det numerisk største element og derefter gennemføre en rækkeombytning således at det numerisk største element placeres på pivotposition i hver søjle. Det benyttes til at undersøge singulariteten under Gauss eliminationen.
Input	double TM[NMAX][NMAX+1], int j, int n
Output	double TM[NMAX][NMAX+1]
Virkemåde	En for-løkke kører igennem alle elementer i den første søjle og tjekker om der er et element som er numerisk større end den tidligere. Hvis der er et element med numerisk større værdi defineres dette element nu som nyt maks. i variablen r. Herefter startes en if-forgrening som kører hvis der er fundet et nyt maks., if-forgreningen åbner en for-løkke, som kører for hver søjle i matricen, og som foretager en rækkeombytning, så det fundne r og den tilhørende række bliver placeret på pivotpositionen
Afhængigheder	N / A

```
void DelvisPivoting(double TM[NMAX][NMAX+1],int j,int n)
{
    double a;
    int r;

    r=j;
    for(int i=j+1;i<n;i++)
    {
        if(fabs(TM[r][j])<fabs(TM[i][j])) r=i;
    }
    if(r!=j){
        for(int k=j;k<=n;k++)
        {
            a=TM[j][k];
            TM[j][k]=TM[r][k];
            TM[r][k]=a;
        }
    }
}
```

<b>void Gauss(double TM[NMAX][NMAX+1],int n,int &amp;bs);</b>	
Formål	Gauss-eliminationen bruges til at undersøge totalmatricen, og derved bestemme om den er singulær eller regulær. Hvis totalmatricen er regulær klargøre

	funktionen matricen til backwards substitution, hvilket vil resultere i beregningen af $x^*$
Input	double TM[NMAX][NMAX+1], int m
Output	TM[NMAX][NMAX+1], hvor der er skabt en øvre trekantsmatrix, så der kan fortages backwards substitution på matricen. Værdien bs, som angiver om der kan fortages backwards substitution og dermed om der fremkommer singularitet.
Virkemåde	Funktionen sætter bs = 1, hvorefter en for-løkke kører for hver søjle i matricen, fra den 0'te til den n-2'te søjle, eftersom der ikke kan laves 0'er under pivoteringspositionen i den sidste søjle (n-1) da dette netop er det sidste element søjlen. For-løkken starter med at foretage delvis pivotering hvorefter der åbnes en if-forgrening der undersøger om pivotpositionen i den aktuelle søjle er numerisk mindre end den bestemte 0-tolerence. Hvis dette er tilfældet forekommer der et elementet med værdi 0 i diagonalen, der oplyses at matricen er singulær og funktionen indstilles. Hvis matricen ikke dømmes singulær efter overstående beregning åbnes en ny for-løkke der kører for hver række, med start i 2. række da der ikke foretages rækkeoperationer på 1. række. I denne for-løkke bestemmes den faktor der skal benyttes for at skabe elementer under diagonalen med værdi 0, herefter kommer en ny for-løkke der køres for hvert element i den pågældende række. Her ganges faktoren på det tilsvarende element i den første række og lægges da til elementet i den aktuelle række. På denne måde bliver der skabt en øvre trekantsmatrix med 0'er under diagonalen. Da den første for-løkke kun kørte til den m-2'te søjle, undersøges den m-1'te søjle herefter i en if-forgrening, for om denne er numerisk mindre end den indtastede 0-tolerance. Hvis dette er tilfældet vil der være et 0 i diagonalen, og totalmatricen vil da blive dømt singulær.
Afhængigheder	DelvisPivotering

```
void Gauss(double TM[NMAX][NMAX+1],int m,int &bs)
{
    int i,j,k;
    double factor,eps=0.00000001;
    bs=1;
    for(j=0;j<=m-2;j++)
    {
        DelvisPivotering(TM,j,m);
        if(fabs(TM[j][j])<eps)
        {
            bs=0;
            cout<<"Der forekommer singularitet.";
            break;
        }
        for(i=j+1;i<=m-1;i++)
        {
            factor=-TM[i][j]/TM[j][j];
            TM[i][j]=0;
            for(k=j+1;k<=m;k++)
            {
                TM[i][k]=TM[i][k]+factor*TM[j][k];
            }
        }
    }
    if(fabs(TM[m-1][m-1])<eps)
```

```

    {
        bs=0;
        cout<<"Der forekommer singularitet.";
    }
}

```

<b>void QRmodGS(double A[NMAX][NMAX], double Q[NMAX][NMAX], double R[NMAX][NMAX], int n, int m);</b>
--

<b>void QRmodGS(double A[NMAX][NMAX], double Q[NMAX][NMAX], double R[NMAX][NMAX], int n, int m);</b>	
Formål	At fortage QR – faktorisering med en given matrix A så $\underline{A} = \underline{Q}\underline{R}$
Input	double A[NMAX][NMAX], int n, int m
Output	double Q[NMAX][NMAX], double R[NMAX][NMAX]
Virkemåde	I de to første for-løkker kopieres matrix A over i matrix AA. Derefter begynder en for-løkke der opdaterer henholdsvis første søjle i Q og første række i R. Derudover udregner den diagonalen i R( $r_{kk}$ ) for-løkken kører indtil $k < m - 1$ Den indre for-løkke bestemmer alle elementer til højre for diagonalen i R i den k'te række og sætter alt under diagonalen = 0, samt matrix AA opdateres så $a_j$ er ortognal til $q_k$ og dermed også til $q_1, \dots, q_{k-1}$ , for-løkken kører til $j < m$ . Til sidst sættes diagonalen i matrix R til ved hjælp af den opdateret AA matrix og Q dannes ud fra de opdateret matricer AA og R.
Afhængigheder	Tagsøjle, DanQ, Lvek

```

void QRmodGS(double A[NMAX][NMAX], double Q[NMAX][NMAX], double R[NMAX][NMAX], int n, int m)
{
    double AA[NMAX][NMAX], v[NMAX];
    for (int i=0;i<n;i++)
    {
        for (int j=0;j<m;j++) AA[i][j] = A[i][j];
    }
    for(int k=0;k<(m-1);k++)
    {
        Tagsøjle(AA,v,n,k);
        R[k][k] = Lvek(v,n);
        DanQ(AA,Q,R,n,k);
        for (int j=k+1;j<m;j++)
        {
            R[k][j] = 0;
            for (int i=0;i<n;i++) R[k][j] += Q[i][k]*AA[i][j];
            for (int i=0;i<n;i++) AA[i][j] = AA[i][j]-(R[k][j]*Q[i][k]);
        }
        Tagsøjle(AA,v,n,(m-1));
        R[m-1][m-1] = Lvek(v,n);
        DanQ(AA,Q,R,n,(m-1));
    }
}

```

<b>void IndkredsningsAlg(double A[NMAX][NMAX], double vb[NMAX], double x[NMAX], double sk[NMAX], double d, int n, int m, double &amp;a, double &amp;b);</b>
---

<b>void IndkredsningsAlg(double A[NMAX][NMAX], double vb[NMAX], double x[NMAX], double sk[NMAX], double d, int n, int m, double &amp;a, double &amp;b);</b>	
Formål	Indkredser et interval der indeholder minimum til en funktion
Input	double A[NMAX][NMAX], double vb[NMAX], double x[NMAX], double sk[NMAX], double d, int n, int m
Output	double &a, double &b

Virkemåde	x1 sættes til 0 og x2 defineres som x1 + steplængden givet fra GoldenSection funktionen. Herefter åbnes en while-løkke der kører så længe $f(x_2) \geq f(x_1)$ og opdatere steplængden og x2 indtil $f(x_2) < f(x_1)$ hvorefter while-løkken stopper. Efter while-løkken opdateres steplængden d igen og x3 defineres som $x_2 + steplængden$ . Der åbnes en while-løkke der kører så længe $f(x_2) > f(x_3)$ , her sættes x1 til den opdateres x2 og x2 sættes til den forrige definerede x3, steplængden opdateres og x3 sættes som den opdateres x2 + den opdateret steplængde. Dette forsætter til $f(x_2) \leq f(x_3)$ hvorefter løkken stopper og de to ende intervaller fra indkredsningsalgoritmen er fundet i form af x1 og x3 som defineres som a og b og referenceoverføres.
Afhængigheder	falpha

```
void IndkredsningsAlg(double A[NMAX][NMAX], double vb[NMAX], double x[NMAX], double sk[NMAX], double d, int n, int m, double &a, double &b)
{
    double x1, x2, x3;

    x1=0;
    x2=x1+d;
    while (falpha(A,vb,x,sk,x2,n,m)>=falpha(A,vb,x,sk,x1,n,m))
    {
        d=0.1*d;
        x2=x1+d;
    }
    d=2*d;
    x3=x2+d;
    while (falpha(A,vb,x,sk,x2,n,m)>falpha(A,vb,x,sk,x3,n,m))
    {
        x1=x2;
        x2=x3;
        d=2*d;
        x3=x2+d;
    }
    a=x1;
    b=x3;
}
```

void GoldenSection(double A[NMAX][NMAX], double b[NMAX], double x[NMAX],double sk[NMAX],int n,int m,double &alph);	
Formål	At finde minimum i det givne interval
Input	double A[NMAX][NMAX], double b[NMAX], double x[NMAX], double sk[NMAX], int n, int m
Output	double &alph
Virkemåde	eps (nultolerancen) og d (steplængden) defineres og funktionen IndkredsningsAlg udregner intervallet hvori minimum findes som a1 og a4 og den gyldne snit konstant c defineres. Herefter udformes a2 og a3 ud fra henholdsvis a1 og a4 samt brug af konstanten c. Der åbnes derefter en do-while-løkke der kører så længe de ydre punkter (a1 og a4) i intervallet har en differens på mere end den givne nultolerance. I denne løkke ligger en if/else-løkke der opdatere a1, a2 og a3 så længe $f(a_2) \geq f(a_3)$ , når dette ikke længere er tilfældet ryger beregningerne ned i else-ledet hvor a4, a3 og a2 opdateres, herefter tjekker do-while-løkken og nultolerancen stadig overholdes hvorefter den køres igen indtil tolerancen er nået.

	Til sidst sættes alph til gennemsnittet af de endelige værdier for a4 og a1 og funktionen er færdig og referenceoverføre alph.
Afhængigheder	IndkredsningsAlg, falpha

```

void GoldenSection(double A[NMAX][NMAX], double b[NMAX], double x[NMAX],double sk[NMAX],int n,int m,double &alph)
{
    double a1, a2, a3, a4, c, eps=0.00000001,d=0.25;

    IndkredsningsAlg(A,b,x,sk,d,n,m,a1,a4);
    c=(sqrt(5.0)-1)/2;
    a2=a1+(1-c)*(a4-a1);
    a3=a4-(1-c)*(a4-a1);
    do
    {
        if (falpha(A,b,x,sk,a2,n,m)>=falpha(A,b,x,sk,a3,n,m))
        {
            a1=a2;
            a2=a3;
            a3=a4-(1-c)*(a4-a1);
        }
        else
        {
            a4=a3;
            a3=a2;
            a2=a1+(1-c)*(a4-a1);
        }
    } while((a4-a1)>=eps);
    alph=0.5*(a4+a1);
}

```

## Konklusion

Jeg kan konkludere at problemformuleringen er blevet løst til fulde eftersom programmet udfører alle de ønskede beregninger på en fornuftig og korrekt måde, endvidere er programstrukturen opbygget efter den ønskede grundstruktur og alle testkørsler viser at det virker som ønsket.

Den største usikkerhed der fremkommer i programmet er i Metode 3) hvor indkræsningsalgoritmen laver en tilnærmelse som følge af en defineret eps nultolerance, hvis denne tolerance ikke er tilstrækkeligt lavt sat (eks. 0.00000001) vil der forekomme afrundingsfejl.

Som fejlkilder vil jeg påpege double-precision datatypsens mangel i form af en præcisionsgrænse på 15/16 decimaler der gør beregning i denne boldgade vanskelig og der vil i sådanne tilfælde fremkomme afrundingsfejl.

Bestemmelsen af  $\underline{x}^*$  skulle fremkomme efter 3 metoder i programmet, som følge:

$$1) \quad \underline{A} \underline{x} = \underline{b} \Leftrightarrow \underline{A}^T \underline{A} \underline{x}^* = \underline{A}^T \underline{b} \Leftrightarrow \underline{x}^* = (\underline{A}^T \underline{A})^{-1} \underline{A}^T \underline{b}$$

Som er normalligningerne og løses ved Gauss elimination, delvis pivotering og backwards substitution.

### OPFYLDT

$$2) \quad \underline{A} \underline{x} = \underline{b} \Leftrightarrow \underline{R} \underline{x} = \underline{Q}^T \underline{b}$$

Hvor QR-faktoriseringen:  $\underline{A} = \underline{Q} \underline{R}$  er frembragt ved modificeret Gram-Schmidt ortogonalisering af  $\underline{A}$ .  $\underline{R} \underline{x} = \underline{Q}^T \underline{b}$  løses ved backwards substitution og giver derved  $\underline{x}^*$

### OPFYLDT

### 3) Minimering af $f(\underline{x})$

Hvor  $f(\underline{x}) = \left| \underline{b} - \underline{A} \underline{x} \right|^2 = (\underline{b} - \underline{A} \underline{x})^T (\underline{b} - \underline{A} \underline{x})$  er en kvadratisk form og dermed en funktion af de m variable  $\underline{x}^T = (x_1, x_2, \dots, x_m)$  og hvor  $f(\underline{x}') = \min_{\underline{x}} f(\underline{x})$

Minimeringen af  $f(\underline{x})$  skal udføres ved hjælp af en specificeret iterativ optimeringsalgoritme.

### OPFYLDT

Tilvejebringelse af data i Case A), B) og C) er også opfyldt hvilket er dokumenteret i de pågældende afsnit om tilvejebringelse af data. Her ligges der primært fokus på Case C) hvor der faktisk fortages beregninger. Disse er også konkluderet korrekte som følge af testkørsler.

Som afsluttende kommentar kan jeg også konkludere at opgave 5) i sættet er løst til fulde eftersom der printes antal iterationer som  $N = m$ , hvilket er korrekt samt dannelsen af den inverse hessematrice (iterationsmatricen) som er identisk til  $\frac{1}{2} * Rinv * RinvT$  matricen såfremt eps nultolerancen er sat tilstrækkeligt lavt (0.00000001).

## Bilag

### Øvrige funktioner

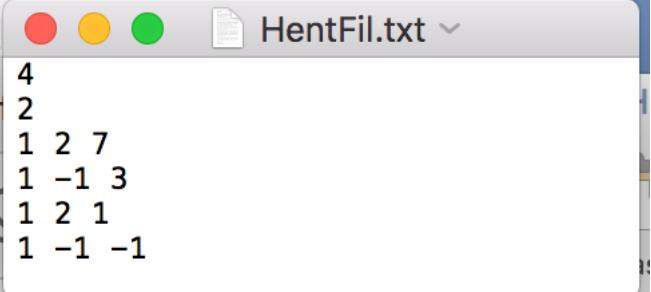
Funktion	Type	Formål
AngivM	void	Angiver en matrix M
Angivv	void	Angiver en vektor v
HentM	void	Henter matrix M fra DelC.txt
UdskrivM	void	Udskriver en matrix
Udskrивv	void	Udskriver en matrix
GemBogC	void	Gemmer n, m, matrix A og vektor b til tekstfil
Introduktion	void	Inderholder introduktionstekst til programmet
CaseA	void	Overordnet funktion til dannelsel af data i Case A)
CaseB	void	Overordnet funktion til dannelsel af data i Case B)
CaseC	void	Overordnet funktion til dannelsel af data i Case C)
ProduktMogv	void	Danner et produkt ud fra en matrix og vektor
ProduktMogM	void	Danner et produkt ud fra 2 matricer
TransponerM	void	Transponerer en matrix
ProduktMogMT	void	Danner et produkt ud fra en matrix og en transponeret matrix
Totalmatrix	void	Danner en totalmatrix
Tagsoejle	void	Trækker en søjle fra en given matrix
RetMatrix	void	Funktion til at fortage rettelser i en matrix
Indprodukt	void	Udregner indreprodukt
RetVektor	void	Funktion til at fortage rettelser i en vektor
Metode1	void	Overordnet funktion til udregning af data i Metode 1)
DanQ	void	Danner matrix Q
Metode2	void	Overordnet funktion til udregning af data i Metode 2)
ParameterMetode3	void	Funktion til indtasting af data til Metode 3)
Gradient	void	Danner en vektor som gradient
DanHesse	void	Danner en tom hessematrix
Hesse	void	Fylder den tomme hessematrix med beregnet værdier
BFGS	void	Danner vektor ud fra BFGS søgeretning
Metode3	void	Overordnet funktion til udregning af data i Metode 3)
Danenhedsvek	void	Danner en enhedsvektor
DanRinv	void	Danner den inverse R matrix
RinvogRinvT	void	Laver produktet af den inverse R matrix og den inverste R matrix transponeret
AnalyseMetode3	void	Overordnet funktion til analyse af data i metode 3)
f	double	Indholder funktionen f(x)
Pprodukt	double	Laver prikproduktet mellem to vektorer

## Variabler

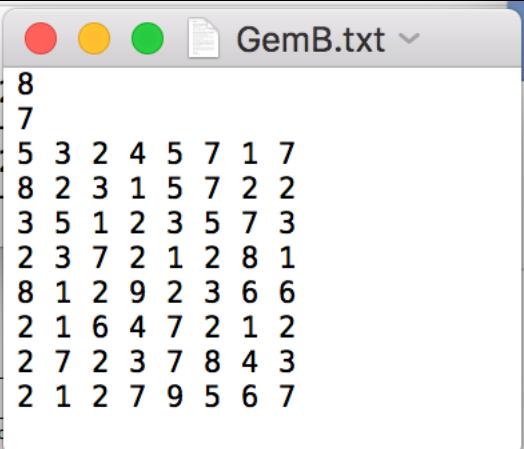
Variabel	Type	Formål
a	double	Indeholder første referenceoverførte værdi for indkrædsningsalgoritmen
A[NMAX][NMAX]	double	Indeholder en matrix <u>A</u>
alph	double	Indeholder en konstant til udregning af $x^*$ i metode 3)
alpha	double	Indeholder en konstant til dannelse af en vektor v i funktion double falpha
AT[NMAX][NMAX]	double	Indeholder den transponeret matrix <u>AT</u> til en given matrix <u>A</u>
b	double	Indeholder anden referenceoverførte værdi for indkrædsningsalgoritmen
b[NMAX]	double	Indeholder en vektor <u>b</u>
B[NMAX][NMAX]	double	Indeholder en matrix B
eps	double	Indeholder nultolerencen, brugervalgt eller bestemt af funktion
grad[NMAX]	double	Indeholder den vektor der findes i funktionen double gradient
H[NMAX][NMAX]	double	Indeholder hessematricen <u>H</u>
i, j, k	double	Iterationsvariabler
m	double	Indeholder antal søjler
M[NMAX][NMAX]	double	Indeholder en matrix <u>M</u>
n	double	Indeholder antal rækker
N	double	Indeholder maks. antal iterationer
P[NMAX][NMAX]	double	Indeholder matrixproduktet <u>P</u> til en matrix <u>A</u> og matrix <u>B</u>
Q[NMAX][NMAX]	double	Indeholder den fundne matrix <u>Q</u> som led af QR-faktorisering
R[NMAX][NMAX]	double	Indeholder den fundne eller brugerdefineret matrix <u>R</u> som led af QR-faktorisering
Rinv[NMAX][NMAX]	double	Indeholder den inverse matrix til <u>R</u>
sk[NMAX]	double	Indeholder en vektor sk
TM[NMAX][NMAX+1]	double	Indeholder en totalmatrix <u>TM</u> til en matrix <u>A</u> og vektor y
v[NMAX]	double	Indeholder en vektor v
vb[NMAX]	double	Indeholder en vektor vb
x[NMAX]	double	Indeholder en vektor $x^*$
xgl[NMAX]	double	Indeholder gammel x
xny[NMAX]	double	Indeholder ny x
y[NMAX]	double	Indeholder en vektor y

## Datafiler

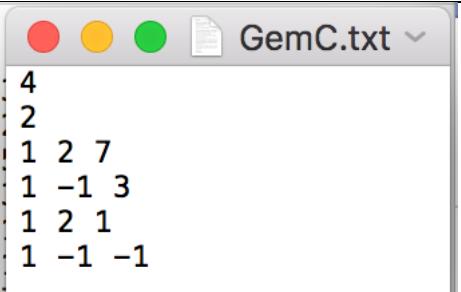
### HentFil.txt

Anvendes af:	Case A)
Indeholder:	n og m, samt en matrix ( $n \times m$ ) hvor $n > m$ og en vektor b på højresiden
Eksempel:	

### GemB.txt

Anvendes af:	Case B)
Indeholder:	n og m, samt en matrix ( $n \times m$ ) hvor $n > m$ og en vektor b på højresiden
Eksempel:	

### GemC.txt

Anvendes af:	Case C)
Indeholder:	n og m, samt en matrix ( $n \times m$ ) hvor $n > m$ og en vektor b på højresiden
Eksempel:	

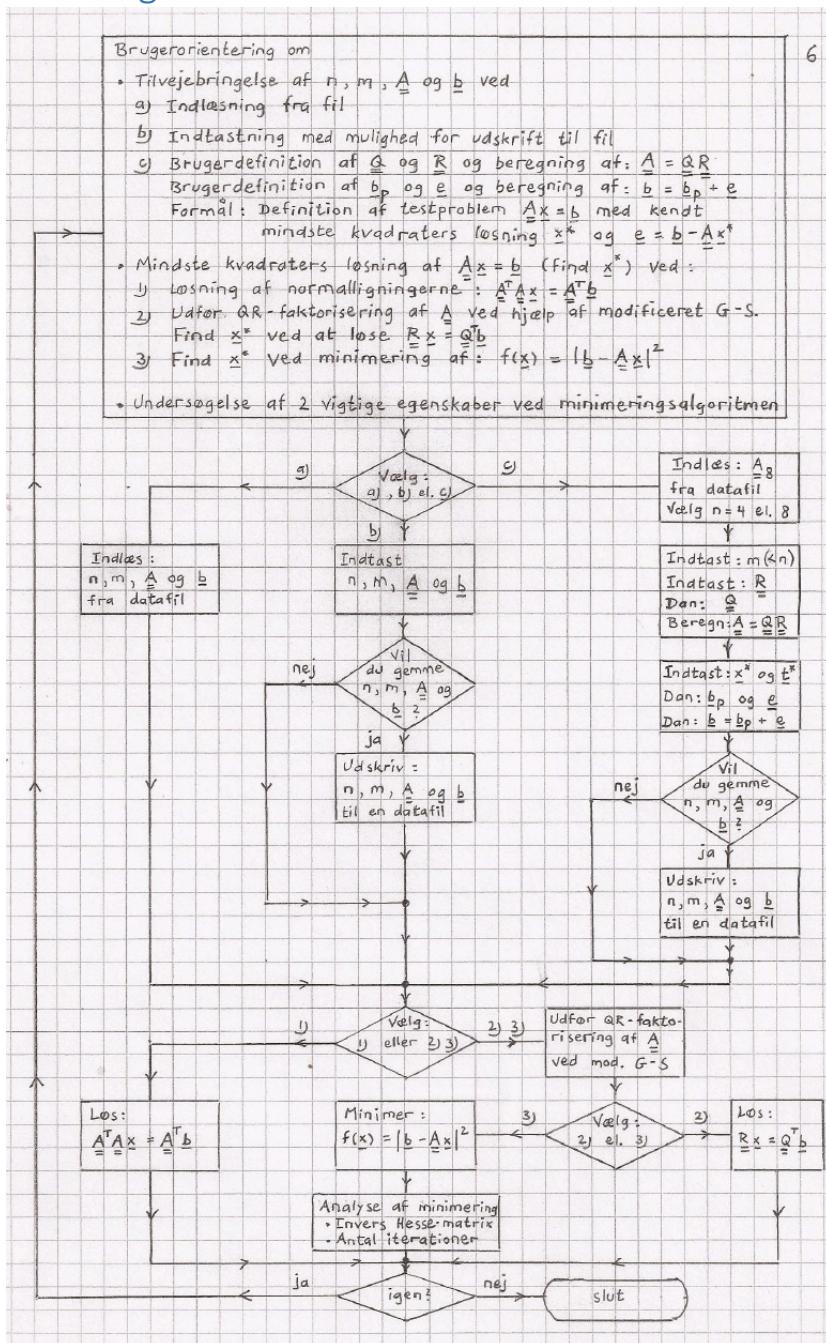
**DelC.txt**

Anvendes af:	Case C)
Indeholder:	Indeholder en forudbestemt 8x8 matrix
Eksempel:	 The screenshot shows a terminal window titled "DelC.txt" containing the following 8x8 matrix: <pre>1 1 1 0 1 0 0 0 1 -1 0 1 0 1 0 0 1 1 -1 0 0 0 1 0 1 -1 0 -1 0 0 0 1 1 1 1 0 -1 0 0 0 1 -1 0 1 0 -1 0 0 1 1 -1 0 0 0 -1 0 1 -1 0 -1 0 0 0 1</pre>

**Brugervejledning**

Brugervejledning forekommer løbende gennem programmet hvilket også fremgår af gennemkørslen af venstre og højre side i grundstruktur testkørslen længere nede.

## Rutediagram



## Programstrukturen

Programstrukturen er skrevet og dannet ud fra det give rutediagram, og fremgår i rapporten under afsnittet 'Grundstruktur.cpp' hvori der ikke fortages nogle beregninger men blot viser implementering af grundstruktur samt kort forklaring om hvad der sker i programmet på det pågældende punkt.

## Grundstruktur.cpp

```

#define NMAX 20

#include <iostream>
#include <iomanip>
#include <math.h>
#include <fstream>

using namespace std;

void Introduktion();

//-----
//----- MAIN -----
//-----

int main() {

    char Valgabc, Valg1eller23, Valg2eller3, Valglgen, ValgGemB, ValgGemC;
    int retry;

    do{
        Introduktion();
        do{
            cout<<"\nVælg mellem a, b eller c: "; cin>>Valgabc;
            retry=0;

            switch (Valgabc)
            {

                case 'a' :{
                    cout<<"\nDu valgte a.";
                    cout<<"\nPå dette sted indlæses n, m, A og B fra fil."<<endl;

                    // CaseA(A,b,n,m);
                }
                break;

                case 'b' :{
                    cout<<"\nDu valgte b.";
                    cout<<"\nPå dette sted indtastes n, m, A og B manuelt."<<endl;

                    // CaseB(A,b,n,m);

                    cout<<"Ønsker du at skrive n, m, A og b til fil? J/N: "; cin>>ValgGemB;
                    if(ValgGemB=='J')
                    {
                        //GemBogC(A,b,n,m,"GemB.txt");
                        cout<<"Data er skrevet til fil.";
                    }
                }
                break;

                case 'c' :{
                    cout<<"\nDu valgte c.";
                }
            }
        }
    }
}

```

```

cout<<"\nPå dette sted indtastes n, m og R manuelt."<<endl;
cout<<"\nFrembringer matricen Q og beregner A = Q * R.";
cout<<"\nIndtastning x* og t*, derefter beregning bp, e og b"<<endl;

//CaseC(A,b,n,m);

cout<<"\nØnsker du at skrive n, m, A og b til fil? J/N: "; cin>>ValgGemC;
if(ValgGemC=='J')
{
    //GemBogC(A,b,n,m,"GemC.txt");
    cout<<"Data er skrevet til fil.";
}
}

break;

default :{
    cout<<"Du kan kun vælge a, b eller c"<<endl;
    retry=1;
}

}// Switch [SLUT]
}while(retry==1);
do{
    retry=0;

    cout<<"\nDu kan nu vælge følgende fremgangsmåder:"<<endl;
    cout<<"\n1) Bestemmelse af x* ved normalligningerne: At * A * x = At * b"<<endl;
    cout<<"\n2) Eller efter udførelse af QR-faktorisering af matrix A med modificeret Gram-Schmidt:"<<endl;
    cout<<"\n3) Bestemmelse af x* ved minimering af f(x) = (b - A * x)^2"<<endl;
    cout<<"\nVælg mellem (1) eller (2/3): "; cin>>Valg1eller23;

    if(Valg1eller23=='1')
    {
        //Metode1(A, b, x, n, m);
        cout<<"\nNormalligningerne løses og løsning til x* udskrives her"<<endl;
    }

    else if (Valg1eller23 == '2' || Valg1eller23 == '3')
    {
        //QRmodGS(A,Q,R,n,m);
        do{
            cout<<"\nDu har valgt 2 eller 3"<<endl;
            cout<<"\nQR-faktorisering af matrix A er udført ved hjælp af modificeret Gram-Schmidt
til:"<<endl;cout<<"\nMatrix A udskrives her: "<<endl;
            //UdskrivM(A,n,m);
            cout<<endl<<"Matrix Q udskrives her:<<endl;
            //UdskrivM(Q,n,m);
            cout<<endl<<"Matrix R udskrives her:<<endl;
            //UdskrivM(R,m,m);
            cout<<"\nDu kan nu vælge imellem:"<<endl;
            cout<<"\n2) Bestemmelse af mindre kvadraters løsningen x* ved at løse R * x = Qt * b";
            cout<<"\n3) Bestemmelse af mindre kvadraters løsningen x* ved minimering af f(x) = (b - A *
x)^2"<<endl;
            cout<<"\nVælg mellem 2 eller 3: "; cin>>Valg2eller3;
            retry=0;
        switch (Valg2eller3) // Switch [START]
        {
            case 1:
                cout<<"\nNormalligningerne løses og løsning til x* udskrives her:<<endl;
                //Metode1(A, b, x, n, m);
                cout<<endl<<"Matrix Q udskrives her:<<endl;
                //UdskrivM(Q,n,m);
                cout<<endl<<"Matrix R udskrives her:<<endl;
                //UdskrivM(R,m,m);
                cout<<"\nDu kan nu vælge imellem:"<<endl;
                cout<<"\n2) Bestemmelse af mindre kvadraters løsningen x* ved at løse R * x = Qt * b";
                cout<<"\n3) Bestemmelse af mindre kvadraters løsningen x* ved minimering af f(x) = (b - A *
x)^2"<<endl;
                cout<<"\nVælg mellem 2 eller 3: "; cin>>Valg2eller3;
                retry=0;
            break;
        }
    }
}
}

```

```

{
    case '2':
    {
        cout<<"\nDu valgte 2, der løses efter ligningen R * x = Qt * b"<<endl;
        //Metode2(A,b,Q,R,n,m); //Udskriver QR-faktorisering
    }
    break;

    case '3':
    {
        cout<<"\nDu valgte 3, der løses efter minimering af f(x) = (b - Ax)^2"<<endl;
        //Metode3(A,H,b,n,m);
        //AnalyseMetode3(R,P,H,m);
    }
    break;
    default :
    {
        cout<<"Du kan kun skrive 2 eller 3, prøv igen: "<<endl;
        retry=1;
    }
    } //Switch [SLUT]
}while(retry==1);
}
else{
    cout<<"\nDu kan kun skrive 1, 2 eller 3, prøv igen: "<<endl;
    retry=1;
}
}while(retry == 1);

cout << "\nSkal programmet startes fra begyndelsen igen? Tast J/N: ";
cin >> Valgigen;

}while(Valgigen=='J');
}

//----- FUNKTIONER TIL INTRODUKTIONEN -----
void Introduktion()
{
    cout<<"Velkommen til programmet!"<<endl<<endl;
    cout<<"Formål: Definition af testproblem A * x = b med kendt"<<endl;
    cout<<"    mindste kvadraters løsning x* og e = b - A * x*"<<endl<<endl;
    cout<<"Du har følgende muligheder for tilvejebringelse af n, m, A og b:"<<endl;
    cout<<"(na) Indlæsning af data fra fil.";
    cout<<"(nb) Indtastning af n, m, A og b, med mulighed for udskrift til fil.";
    cout<<"(nc) Dannelse af Q og indtastning af R til beregning af A = Q * R";
    cout<<"\n Beregning af af bp og e ud fra indtastet x* og t* samt beregning af b = bp + e"<<endl;
}

```

## Gennemkørsel af grundstruktur.cpp

### Venstresiden

Velkommen til programmet!

Formål: Definition af testproblem  $A * x = b$  med kendt mindste kvadraters løsning  $x^*$  og  $e = b - A * x^*$

Du har følgende muligheder for tilvejebringelse af  $n$ ,  $m$ ,  $A$  og  $b$ :

- Indlæsning af data fra fil.
- Indtastning af  $n$ ,  $m$ ,  $A$  og  $b$ , med mulighed for udskrift til fil.
- Dannelse af  $Q$  og indtastning af  $R$  til beregning af  $A = Q * R$   
Beregning af af  $b_p$  og  $e$  ud fra indtastet  $x^*$  og  $t^*$  samt beregning af  $b = b_p + e$

Vælg mellem a, b eller c: **a**

Du valgte a.

På dette sted indlæses  $n$ ,  $m$ ,  $A$  og  $B$  fra fil.

Du kan nu vælge følgende fremgangsmåder:

- Bestemmelse af  $x^*$  ved normalligningerne:  $At * A * x = At * b$

Eller efter udførelse af QR-faktorisering af matrix  $A$  med modificeret Gram-Schmidt:

- Bestemmelse af  $x^*$  ved at løse  $R * x = Qt * b$ :
- Bestemmelse af  $x^*$  ved minimering af  $f(x) = (b - A * x)^2$

Vælg mellem (1) eller (2/3): **1**

Normalligningerne løses og løsning til  $x^*$  udskrives her

Skal programmet startes fra begyndelsen igen? Tast J/N:

## Højresiden

---

Velkommen til programmet!

Formål: Definition af testproblem  $A * x = b$  med kendt  
mindste kvadraters løsning  $x^*$  og  $e = b - A * x^*$

Du har følgende muligheder for tilvejebringelse af  $n$ ,  $m$ ,  $A$  og  $b$ :

- Indlæsning af data fra fil.
- Indtastning af  $n$ ,  $m$ ,  $A$  og  $b$ , med mulighed for udskrift til fil.
- Dannelse af  $Q$  og indtastning af  $R$  til beregning af  $A = Q * R$   
Beregning af af  $bp$  og  $e$  ud fra indtastet  $x^*$  og  $t^*$  samt beregning af  $b = bp + e$

Vælg mellem a, b eller c: **c**

Du valgte c.

På dette sted indtastes  $n$ ,  $m$  og  $R$  manuelt.

Frembringer matricen  $Q$  og beregner  $A = Q * R$ .  
Indtastning  $x^*$  og  $t^*$ , derefter beregning  $bp$ ,  $e$  og  $b$

Ønsker du at skrive  $n$ ,  $m$ ,  $A$  og  $b$  til fil? J/N: **J**

Data er skrevet til fil.

Du kan nu vælge følgende fremgangsmåder:

- Bestemmelse af  $x^*$  ved normalligningerne:  $At * A * x = At * b$

Eller efter udførelse af QR-faktorisering af matrix  $A$  med modificeret Gram-Schmidt:

- Bestemmelse af  $x^*$  ved at løse  $R * x = Qt * b$ :
- Bestemmelse af  $x^*$  ved minimering af  $f(x) = (b - A * x)^2$

Vælg mellem (1) eller (2/3): **2**

Du har valgt 2 eller 3

QR-faktorisering af matrix  $A$  er udført ved hjælp af modificeret Gram-Schmidt til:

Matrix  $A$  udskrives her:

Matrix  $Q$  udskrives her:

Matrix  $R$  udskrives her:

Du kan nu vælge imellem:

- Bestemmelse af mindre kvadraters løsningen  $x^*$  ved at løse  $R * x = Qt * b$
- Bestemmelse af mindre kvadraters løsningen  $x^*$  ved minimering af  $f(x) = (b - A * x)^2$

Vælg mellem 2 eller 3: **2**

|  
Du valgte 2, der løses efter ligningen  $R * x = Qt * b$

Skal programmet startes fra begyndelsen igen? Tast J/N:

## Program.cpp

```
#define NMAX 20

#include <iostream>
#include <iomanip>
#include <math.h>
#include <fstream>

using namespace std;

//-----
//----- DEKLARATION AF ANVENDTE FUNKTIONER -----
//-----

//----- FUNKTIONER TIL INDHENTNING OG UDSKRIVNING AF DATA -----
//Indhentning:
void AngivM(double M[NMAX][NMAX], int n, int m);
void Angivv(double v[NMAX], int n);
void HentM(double M[NMAX][NMAX], const char* filename);
void HentAogb(double A[NMAX][NMAX], double b[NMAX], int &n, int &m, const char* filename);

//Udskrivning:
void UdskrivM(double M[NMAX][NMAX], int n, int m);
void Udskriva(double v[NMAX], int n);
void GemBogC(double A[NMAX][NMAX], double b[NMAX], int n, int m, const char* filename);

//----- FUNKTIONER TIL INTRODUKTIONEN -----
void Introduktion();
void CaseA(double A[NMAX][NMAX], double b[NMAX], int &n, int &m);
void CaseB(double A[NMAX][NMAX], double b[NMAX], int &n, int &m);
void BeregnAogb(double A[NMAX][NMAX], double b[NMAX], int &n, int &m);
void CaseC(double A[NMAX][NMAX], double b[NMAX], int &n, int &m);

//----- FUNKTIONER TIL MATRIX OG VEKTOR REGNING -----
//Matrix:
void ProduktMogv(double A[NMAX][NMAX], double x[NMAX], double b[NMAX], int n, int m);
void ProduktMogM(double A[NMAX][NMAX], double B[NMAX][NMAX], double P[NMAX][NMAX], int n, int m);
void TransponerM(double A[NMAX][NMAX], double AT[NMAX][NMAX], int n, int m);
void ProduktMogMT(double A[NMAX][NMAX], double AT[NMAX][NMAX], double P[NMAX][NMAX], int n, int m);
void Totalmatrix(double TM[NMAX][NMAX+1], double A[NMAX][NMAX], double y[NMAX], int m);
void Tagsoejle(double A[NMAX][NMAX], double v[NMAX], int n, int k);
void RetMatrix(double M[NMAX][NMAX], int n, int m);

//Vektor:
void Indprodukt(double a[NMAX], double b[NMAX], double p[NMAX], int n);
double Pprodukt(double a[NMAX], double b[NMAX], int n);
double Ldiffvek(double a[NMAX], double b[NMAX], double n);
double Lvek(double v[NMAX], int n);
void RetVektor(double v[NMAX], int n);

//----- FUNKTIONER TIL DE ENKELTE METODER -----
//Vigtige funktioner:
void BackwardsSubstitution(double TM[NMAX][NMAX+1], double x[NMAX], int n);

//Metode 1:
void DelvisPivotering(double TM[NMAX][NMAX+1], int j, int n);
```

```

void Gauss(double TM[NMAX][NMAX+1],int n,int &bs);
void Metode1(double A[NMAX][NMAX],double b[NMAX],double x[NMAX],int n, int m);

//Metode 2:
void DanQ(double A[NMAX][NMAX], double Q[NMAX][NMAX], double R[NMAX][NMAX], int n, int k);
void QRmodGS(double A[NMAX][NMAX], double Q[NMAX][NMAX], double R[NMAX][NMAX], int n, int m);
void Metode2(double A[NMAX][NMAX], double b[NMAX], double Q[NMAX][NMAX], double R[NMAX][NMAX],
int n, int m);

//Metode 3:
void ParameterMetode3(double x[NMAX], double &eps, int &N, int m);
double f(double A[NMAX][NMAX], double b[NMAX], double x[NMAX], int n, int m);
double falpha(double A[NMAX][NMAX], double b[NMAX], double x[NMAX], double sk[NMAX], double alpha, int
n, int m);
void Gradient(double A[NMAX][NMAX], double b[NMAX], double x[NMAX], double grad[NMAX], int n, int m);
void DanHesse(double H[NMAX][NMAX], int m);
void Hesse(double A[NMAX][NMAX], double b[NMAX], double H[NMAX][NMAX], double xny[NMAX], double
xgl[NMAX], int n, int m);
void BFGS(double H[NMAX][NMAX], double grad[NMAX], double sk[NMAX], int m);
void IndkredsningsAlg(double A[NMAX][NMAX], double vb[NMAX], double x[NMAX], double sk[NMAX], double
d, int n, int m, double &a, double &b);
void GoldenSection(double A[NMAX][NMAX], double b[NMAX], double x[NMAX],double sk[NMAX],int n,int
m,double &alph);
void Metode3(double A[NMAX][NMAX], double H[NMAX][NMAX], double b[NMAX], int n, int m);

//Analyse af Metode 3:
void Danenhedsvek(double e[NMAX], int m, int k);
void DanRinv(double R[NMAX][NMAX], double Rinv[NMAX][NMAX], int m);
void RinvogRinvT(double Rinv[NMAX][NMAX], double P[NMAX][NMAX], int m);
void AnalyseMetode3(double R[NMAX][NMAX], double P[NMAX][NMAX], double H[NMAX][NMAX], int m);

//-----
//----- MAIN -----
//-----

int main() {

    char Valgabc, Valg1eller23, Valg2eller3, Valglgen, ValgGemB, ValgGemC;
    double A[NMAX][NMAX], Q[NMAX][NMAX], R[NMAX][NMAX];
    double H[NMAX][NMAX], P[NMAX][NMAX];
    double b[NMAX], x[NMAX];
    int n,m,retry;

    do{
        Introduktion();
        do{
            cout<<"\nVælg mellem a, b eller c: "; cin>>Valgabc;
            retry=0;

            switch (Valgabc)
            {

                case 'a' : {
                    cout<<"\nDu valgte a.";
                    cout<<"\nPå dette sted indlæses n, m, A og B fra fil."<<endl;
                    CaseA(A,b,n,m);
                }
            }
        }
    }
}

```

```

        }

        break;

case 'b' : {
    cout<<"\nDu valgte b.";
    cout<<"\nPå dette sted indtastes n, m, A og B manuelt."<<endl;

    CaseB(A,b,n,m);

    cout<<"\nØnsker du at skrive n, m, A og b til fil? J/N: "; cin>>ValgGemB;
    if(ValgGemB=='J')
    {
        GemBogC(A,b,n,m,"GemB.txt");
    }
}
break;

case 'c' : {
    cout<<"\nDu valgte c.";
    cout<<"\nPå dette sted indtastes n, m og R manuelt."<<endl;
    cout<<"\nFrembringer matricen Q og beregner A = Q * R.";
    cout<<"\nIndtastning x* og t*, derefter beregning bp, e og b"<<endl;

    CaseC(A,b,n,m);

    cout<<"\nØnsker du at skrive n, m, A og b til fil? J/N: "; cin>>ValgGemC;
    if(ValgGemC=='J')
    {
        GemBogC(A,b,n,m,"GemC.txt");
    }
}
break;

default :{
    cout<<"Du kan kun vælge a, b eller c"<<endl;
    retry=1;
}

}// Switch [SLUT]
}while(retry==1);
do{
    retry=0;

    cout<<"\nDu kan nu vælge følgende fremgangsmåder:"<<endl;
    cout<<"\n1) Bestemmelse af x* ved normaliseringerne: At * A * x = At * b"<<endl;
    cout<<"\n2) Eller efter udførelse af QR-faktorisering af matrix A med modificeret Gram-Schmidt:"<<endl;
    cout<<"\n3) Bestemmelse af x* ved løse R * x = Qt * b:";
    cout<<"\n3) Bestemmelse af x* ved minimering af f(x) = (b - A * x)^2"<<endl;
    cout<<"\nVælg mellem (1) eller (2/3): "; cin>>Valg1eller23;

    if(Valg1eller23=='1')
    {
        Metode1(A, b, x, n, m);
    }

    else if (Valg1eller23 == '2' || Valg1eller23 == '3')
    {

```

```

QRmodGS(A,Q,R,n,m);
do{
    cout<<"\nDu har valgt 2 eller 3"<<endl;
    cout<<"\nQR-faktorisering af matrix A er udført ved hjælp af modificeret Gram-Schmidt
til:"<<endl;cout<<"\nMatrix A: "<<endl;
    UdskrivM(A,n,m);
    cout<<endl<<"Matrix Q:"<<endl;
    UdskrivM(Q,n,m);
    cout<<endl<<"Matrix R:"<<endl;
    UdskrivM(R,m,m);
    cout<<"\nDu kan nu vælge imellem:"<<endl;
    cout<<"\n2) Bestemmelse af mindre kvadraters løsningen  $x^*$  ved at løse  $R * x = Q^T * b$ ";
    cout<<"\n3) Bestemmelse af mindre kvadraters løsningen  $x^*$  ved minimering af  $f(x) = (b - Ax)^2$ "<<endl;
    cout<<"\nVælg mellem 2 eller 3: "; cin>>Valg2eller3;
    retry=0;
    switch (Valg2eller3) // Switch [START]
    {
        case '2':
        {
            cout<<"\nDu valgte 2, der løses efter ligningen  $R * x = Q^T * b"$ <<endl;
            Metode2(A,b,Q,R,n,m); //Udskriver QR-faktorisering
        }
        break;

        case '3':
        {
            cout<<"\nDu valgte 3, der løses efter minimering af  $f(x) = (b - Ax)^2$ "<<endl;
            Metode3(A,H,b,n,m);
            AnalyseMetode3(R,P,H,m);
        }
        break;
        default :
        {
            cout<<"nej, du skal skriv 2 eller 3"<<endl;
            retry=1;
        }
    } //Switch [SLUT]
}while(retry==1);
}
else{
    cout<<"\nDu kan kun skrive 1, 2 eller 3, prøv igen: "<<endl;
    retry=1;
}
}while(retry == 1);

cout << "\nSkal programmet startes fra begyndelsen igen? Tast J/N: ";
cin >> Valgigen;

}while(Valgigen=='J');
}

// -----
// ----- KODE TIL FUNKTIONER -----
// -----



//----- FUNKTIONER TIL INDHENTNING OG UDSKRIVNING AF DATA -----

```

```

//Indhentning:
void AngivM(double M[NMAX][NMAX], int n, int m)
{
    for(int i=0;i<n;i++){
        for (int j=0;j<m;j++){
            cout << "Angiv elementet [" << i+1 << "," << j+1 << "] = ";
            cin>>M[i][j];
        }
    }
}
void Angivv(double v[NMAX], int n)
{
    for(int i=0;i<n;i++){
        cout << "Angiv " << i+1 << ". element: ";
        cin>>v[i];
    }
}
void HentM(double M[NMAX][NMAX], const char* filename)
{
    ifstream Fil;
    Fil.open(filename);
    for (int i=0;i<8;i++){
        for (int j=0;j<8;j++) Fil >> M[i][j];
    }
    Fil.close();
}
void HentAogb(double A[NMAX][NMAX], double b[NMAX], int &n, int &m, const char* filename)
{
    ifstream Fil;
    Fil.open(filename);
    Fil >> n;
    Fil >> m;
    for (int i=0;i<n;i++){
        for (int j=0;j<=m;j++){
            if(j<m) Fil >> A[i][j];
            else Fil >> b[i];
        }
    }
    Fil.close();
}

//Udskrivning:
void UdskrivM(double M[NMAX][NMAX], int n, int m)
{
    for (int i=0;i<n;i++){
        for (int j=0;j<m;j++) cout << setw(10) << M[i][j];
        cout << endl;
    }
}
void Udskrivv(double v[NMAX], int n)
{
    cout << setw(4) << "[" << v[0];
    for (int i=1;i<n;i++) cout << ", " << v[i];
    cout << "]" << endl;
}
void GemBogC(double A[NMAX][NMAX], double b[NMAX], int n, int m, const char* filename)

```

```

{
    ofstream Fil;
    Fil.open(filename);
    Fil << n << endl;
    Fil << m << endl;
    for (int i=0;i<n;i++){
        for (int j=0;j<=m;j++){
            if(j<m) Fil << A[i][j] << " ";
            else Fil << b[i] << endl;
        }
    }
    Fil.close();
}

//----- FUNKTIONER TIL INTRODUKTIONEN -----
void Introduktion()
{
    cout<<"Velkommen til programmet!"<<endl<<endl;
    cout<<"Formål: Definition af testproblem  $A * x = b$  med kendt"<<endl;
    cout<<"    mindste kvadraters løsning  $x^*$  og  $e = b - A * x^*$ "<<endl<<endl;
    cout<<"Du har følgende muligheder for tilvejebringelse af n, m, A og b:"<<endl;
    cout<<"\na) Indlæsning af data fra fil.";
    cout<<"\nb) Indtastning af n, m, A og b, med mulighed for udskrift til fil.";
    cout<<"\nc) Dannelse af Q og indtastning af R til beregning af  $A = Q * R$ ";
    cout<<"\n Beregning af af bp og e ud fra indtastet  $x^*$  og  $t^*$  samt beregning af  $b = bp + e$ "<<endl;
}
void CaseA(double A[NMAX][NMAX],double b[NMAX], int &n, int &m){
    int i,j;

    ifstream IndFil;

    IndFil.open("HentFil.txt");

    IndFil>>n;
    IndFil>>m;

    for (i=0;i<n;i++){
        for (j=0;j<=m;j++){
            if(j<m) IndFil >> A[i][j];
            else IndFil >> b[i];
        }
    }
    IndFil.close();

    cout<<"\nn = "<<n;
    cout<<"\nm = "<<m<<endl;

    cout<<"\nMatrix A: "<<endl;
    UdskrivM(A,n,m);

    cout<<"\nVektor b: "<<endl;
    Udskrivv(b,n);

    RetMatrix(A,n,m);
    RetVektor(b,n);
}
void CaseB(double A[NMAX][NMAX], double b[NMAX], int &n, int &m)

```

```

{
    cout<<"\nMatricen er en (n x m) matrix hvor n > m"<<endl;
    cout<<"\nIndtast n: "; cin>>n;
    cout<<"\nIndtast m: "; cin>>m;
    AngivM(A,n,m);
    cout<<"\nMatrix A er givet ved:"<<endl;
    UdskrivM(A,n,m);
    RetMatrix(A,n,m);

    cout<<"\nIndtast vektor b: "<<endl;
    Angivv(b,n);
    cout<<"\nVektor b er givet ved:"<<endl;
    Udskrivv(b,n);
    RetVektor(b,n);
}
void BeregnAogb(double A[NMAX][NMAX], double b[NMAX], int &n, int &m)
{
    double D[NMAX][NMAX], M[NMAX][NMAX], Q[NMAX][NMAX], C[NMAX][NMAX], R[NMAX][NMAX];
    double x[NMAX], v[NMAX], bp[NMAX], t[NMAX], e[NMAX];

    HentM(M,"DelC.txt");
    cout << "\nAngiv om du vil have (4) eller (8) rækker, n = ";
    cin >> n;
    cout << "\nMatrix A"<<n<<" er givet ved: "<<endl;
    UdskrivM(M,n,n);

    for(int i=0;i<n;i++){
        Tagsoejle(M,v,n,i);
        D[i][i]=1/sqrt(Pprodukt(v,v,n));
    }
    cout << "\nMatrix D"<<n<<" er givet ved: "<<endl;
    UdskrivM(D,n,n);
    cout << "\nAngiv antallet af søjler hvor m < n, m = ";
    cin >> m;
    ProduktMogMT(D,M,Q,n,n);
    cout << "\nMatrix Q er givet ved: "<<endl;
    UdskrivM(Q,n,m);
    cout << "\nAngiv R's elementer: " << endl;
    for (int i=0;i<m;i++){
        for (int j=0;j<m;j++){
            if (j>=i){
                cout << "Angiv elementet [" << i+1 << "," << j+1 << "] = ";
                cin >> R[i][j];
            }
            else R[i][j] = 0;
        }
    }
    ProduktMogM(Q,R,A,n,m);
    cout << "\nAngiv elementerne i x*" << endl;
    Angivv(x,m);
    ProduktMogv(A,x,bp,n,m);
    cout << "\nAngiv elementerne i t*" << endl;
    Angivv(t,(n-m));
    for (int i=0;i<n;i++){
        for (int j=0;j<n;j++){
            if (j>=m) C[i][(j-m)]=M[i][j];
        }
    }
}

```

```

}

ProduktMogv(C,t,e,n,(n-m));
for (int i=0;i<n;i++)
{
    b[i] = bp[i] + e[i];
}
cout << "\nOrtogonalprojektionen bp er givet ved:" << endl;
Udskrivv(bp,n);
cout << "\nFejlvektoren e er givet ved:" << endl;
Udskrivv(e,n);
cout << endl;
}
void CaseC(double A[NMAX][NMAX], double b[NMAX], int &n, int &m)
{
    BeregnAogg(A,b,n,m);
    cout << "Matrix A er givet ved:" << endl;
    UdskrivM(A,n,m);
    cout << "\nVektor b er givet ved:" << endl;
    Udskrivv(b,n);

    RetMatrix(A,n,m);
    RetVektor(b,n);
}

//----- FUNKTIONER TIL MATRIX OG VEKTOR REGNING -----
//Matrix:
void RetMatrix(double M[NMAX][NMAX],int n, int m)
{
    int i,j;
    char Svar='n';

    cout << "\nSkal der rettes i matricen? J/N: ";
    cin >> Svar;

    while(Svar=='J') {
        cout << "\nIndtast række nr: ";
        cin >> i; i--;
        cout << "\nIndtast søjle nr: ";
        cin >> j; j--;
        cout << "\nIndtast ny værdi: ";
        cin >> M[i][j];

        cout << "\nNy matrix givet ved: " << endl;
        UdskrivM(M,n,m);

        cout << "\nSkal der rettes i matricen igen? J/N: ";
        cin >> Svar;
    }
}
void ProduktMogv(double A[NMAX][NMAX], double x[NMAX], double b[NMAX], int n, int m)
{
    for (int i=0;i<n;i++)
    {
        b[i] = 0;
        for (int j=0;j<m;j++) b[i] += A[i][j]*x[j];
    }
}

```

```

    }
}
void ProduktMogM(double A[NMAX][NMAX],double B[NMAX][NMAX], double P[NMAX][NMAX], int n, int m)
{
    for (int i=0;i<n;i++)
    {
        for (int j=0;j<m;j++)
        {
            P[i][j] = 0;
            for (int k=0;k<m;k++) P[i][j] = P[i][j] + A[i][k]*B[k][j];
        }
    }
}
void TransponerM(double A[NMAX][NMAX],double AT[NMAX][NMAX],int n, int m)
{
    for(int i=0;i<m;i++)
    {
        for(int j=0;j<n;j++) AT[i][j]=A[j][i];
    }
}
void ProduktMogMT(double A[NMAX][NMAX],double AT[NMAX][NMAX], double P[NMAX][NMAX], int n, int m)
{
    for(int i=0;i<m;i++)
    {
        for (int j=0;j<m;j++)
        {
            P[i][j]=0;
            for(int k=0;k<n;k++) P[i][j] += AT[i][k]*A[k][j];
        }
    }
}
void Totalmatrix(double TM[NMAX][NMAX+1],double A[NMAX][NMAX],double y[NMAX],int m)
{
    for(int i=0;i<m;i++)
    {
        for(int j=0;j<m;j++) TM[i][j]=A[i][j];
        TM[i][m]=y[i];
    }
}
void Tagsoejle(double A[NMAX][NMAX], double v[NMAX], int n, int k)
{
    for (int i=0;i<n;i++) v[i]=A[i][k];
}

//Vektor:
void RetVektor(double v[NMAX],int n)
{
    int i;
    char Svar;

    cout<<"\nSkal der rettes i vektoren? J/N: ";
    cin>>Svar;

    while(Svar=='J') {
        cout<<"\nIndtast række nr: ";
        cin>>i;i--;
}

```

```

cout<<"\nIndtast ny værdi: ";
cin>>v[i];
cout<<"\nNy vektor givet ved: "<<endl;
Udskrivv(v,n);
cout<<"\nSkal der rettes i vektoren igen? J/N: ";
cin>>Svar;
}
}

void Indprodukt(double a[NMAX], double b[NMAX], double p[NMAX], int n)
{
    for (int i=0;i<n;i++) p[i] = a[i]*b[i];
}
double Pprodukt(double a[NMAX], double b[NMAX], int n)
{
    double sum = 0;
    for (int i=0;i<n;i++) sum += a[i]*b[i];

    return sum;
}
double Ldiffvek(double a[NMAX], double b[NMAX], double n)
{
    double sum = 0;
    for (int i=0;i<n;i++)
    {
        sum += pow((a[i]-b[i]),2);
    }

    return sqrt(sum);
}
double Lvek(double v[NMAX], int n)
{
    double sum=0;

    for (int i=0;i<n;i++)
    {
        sum += v[i]*v[i];
    }

    return sqrt(sum);
}

//----- FUNKTIONER TIL DE ENKELTE METODER -----
//Vigtige funktioner:
void BackwardsSubstitution(double TM[NMAX][NMAX+1],double x[NMAX],int n)
{
    double sum;

    x[n-1]=TM[n-1][n]/TM[n-1][n-1];
    for(int i=n-2;i>=0;i--)
    {
        sum=0;
        for(int j=i+1;j<n;j++) sum+=TM[i][j]*x[j];
        x[i]=(TM[i][n]-sum)/TM[i][i];
    }
}

//Metode 1:

```

```

void DelvisPivotering(double TM[NMAX][NMAX+1],int j,int n)
{
    double a;
    int r;

    r=j;
    for(int i=j+1;i<n;i++)
    {
        if(fabs(TM[r][j])<fabs(TM[i][j])) r=i;
    }
    if(r!=j){
        for(int k=j;k<=n;k++)
        {
            a=TM[j][k];
            TM[j][k]=TM[r][k];
            TM[r][k]=a;
        }
    }
}
void Gauss(double TM[NMAX][NMAX+1],int m,int &bs)
{
    int i,j,k;
    double factor,eps=0.00000001;
    bs=1;
    for(j=0;j<=m-2;j++)
    {
        DelvisPivotering(TM,j,m);
        if(fabs(TM[j][j])<eps)
        {
            bs=0;
            cout<<"Der forekommer singularitet.";
            break;
        }
        for(i=j+1;i<=m-1;i++)
        {
            factor=-TM[i][j]/TM[j][j];
            TM[i][j]=0;
            for(k=j+1;k<=m;k++)
            {
                TM[i][k]=TM[i][k]+factor*TM[j][k];
            }
        }
        if(fabs(TM[m-1][m-1])<eps)
        {
            bs=0;
            cout<<"Der forekommer singularitet.";
        }
    }
}
void Metode1(double A[NMAX][NMAX],double b[NMAX],double x[NMAX],int n, int m)
{
    double TM[NMAX][NMAX+1], AT[NMAX][NMAX], P[NMAX][NMAX], p[NMAX];
    int bs;

    TransponerM(A,AT,n,m);
    cout<<"\nDen transponeret matrix AT er givet ved: "<<endl;
    UdskrivM(AT,m,n);
}

```

```

ProduktMogMT(A,AT,P,n,m);
cout<<"\nMatrix produktet ATA er givet ved: "<<endl;
UdskrivM(P,m,m);
ProduktMogv(AT,b,p,m,n);
cout<<"\nMatrix/vektor produktet ATb er givet ved: "<<endl;
Udskrивv(p,m);
Totalmatrix(TM,P,p,m);
Gauss(TM,m,bs);
if(bs==1) BackwardsSubstitution(TM,x,m);
else cout << "Det lineære ligningssystem kunne ikke bestemmes." << endl;
cout<<"\nx* givet ved: "<<endl;
Udskrивv(x,m);
}

//Metode 2:
void Metode2(double A[NMAX][NMAX], double b[NMAX], double Q[NMAX][NMAX], double R[NMAX][NMAX],
int n, int m)
{
    double TM[NMAX][NMAX+1], QT[NMAX][NMAX];
    double y[NMAX], x[NMAX];
    TransponerM(Q,QT,n,m);
    ProduktMogv(QT,b,y,m,n);
    Totalmatrix(TM,R,y,m);
    BackwardsSubstitution(TM,x,m);
    cout<<"\nx* givet ved: "<<endl;
    Udskrивv(x,m);
}
void DanQ(double A[NMAX][NMAX], double Q[NMAX][NMAX], double R[NMAX][NMAX], int n, int k)
{
    for (int i=0;i<n;i++)
    {
        Q[i][k] = (1/R[k][k])*A[i][k];
    }
}
void QRmodGS(double A[NMAX][NMAX], double Q[NMAX][NMAX], double R[NMAX][NMAX], int n, int m)
{
    double AA[NMAX][NMAX], v[NMAX];

    for (int i=0;i<n;i++)
    {
        for (int j=0;j<m;j++) AA[i][j] = A[i][j];
    }
    for(int k=0;k<(m-1);k++)
    {
        Tagsoejle(AA,v,n,k);
        R[k][k] = Lvek(v,n);
        DanQ(AA,Q,R,n,k);
        for (int j=k+1;j<m;j++)
        {
            R[k][j] = 0;
            for (int i=0;i<n;i++) R[k][j] += Q[i][k]*AA[i][j];
            for (int i=0;i<n;i++) AA[i][j] = AA[i][j]-(R[k][j]*Q[i][k]);
        }
        Tagsoejle(AA,v,n,(m-1));
        R[m-1][m-1] = Lvek(v,n);
        DanQ(AA,Q,R,n,(m-1));
    }
}

```

```

}

//Metode 3:
void Metode3(double A[NMAX][NMAX], double H[NMAX][NMAX], double b[NMAX], int n, int m)
{
    double grad[NMAX], sk[NMAX], x[NMAX], xgl[NMAX];
    double eps, alph=0;
    int k=-1, N;

    ParameterMetode3(x,eps,N,m);
    do
    {
        k++;
        if (k == 0) DanHesse(H,m);
        else Hesse(A,b,H,x,xgl,n,m);
        Gradient(A,b,x,grad,n,m);
        BFGS(H,grad,sk,m);
        GoldenSection(A,b,x,sk,n,m,alph);
        for (int i=0;i<m;i++){
            xgl[i] = x[i];
            x[i] = x[i] + alph*sk[i];
        }
    } while (Ldiffvek(x,xgl,m)>eps && Lvek(grad,m)>eps && k<N);
    cout<<"\nAntal iterationer: "<<k<<endl;
    cout<<"\nVektoren er givet ved:"<<endl;
    Udskrivv(x,m);
}
void ParameterMetode3(double x[NMAX], double &eps, int &N, int m)
{
    cout << "\nAngiv dit startpunkt, x:" << endl;
    Angivv(x,m);
    cout << "\nAngiv maksimalt antal af iterationer, N = ";
    cin >> N;
    cout << "\nAngiv tolerancen, eps = ";
    cin >> eps;
}
double f(double A[NMAX][NMAX], double b[NMAX], double x[NMAX], int n, int m)
{
    double AT[NMAX][NMAX], P[NMAX][NMAX], p[NMAX];
    double led1 = 0, led2 = 0, led3 = 0;

    for (int i=0;i<n;i++) led1 += b[i]*b[i];
    ProduktMogv(A,x,p,n,m);
    for (int i=0;i<n;i++) led2 += 2*b[i]*p[i];
    TransponerM(A,AT,n,m);
    ProduktMogMT(A,AT,P,n,m);
    ProduktMogv(P,x,p,n,m);
    for (int i=0;i<n;i++) led3 += x[i]*p[i];

    return led1-led2+led3;
}
double falpha(double A[NMAX][NMAX], double b[NMAX], double x[NMAX], double sk[NMAX], double alpha, int n, int m)
{
    double v[NMAX];

    for(int i=0;i<n;i++) v[i]=x[i]+alpha*sk[i];
}

```

```

    return f(A,b,v,n,m);
}
void Gradient(double A[NMAX][NMAX], double b[NMAX], double x[NMAX], double grad[NMAX], int n, int m)
{
    double AT[NMAX][NMAX], P[NMAX][NMAX];
    double p1[NMAX], p2[NMAX];

    TransponerM(A,AT,n,m);
    ProduktMogMT(A,AT,P,n,m);
    ProduktMogv(P,x,p1,n,m);
    ProduktMogv(AT,b,p2,m,n);
    for (int i=0;i<m;i++) grad[i] = 2*p1[i] - 2*p2[i];
}
void DanHesse(double H[NMAX][NMAX], int m)
{
    for (int i=0;i<m;i++)
    {
        for (int j=0;j<m;j++)
        {
            if (i==j) H[i][j] = 1;
            else H[i][j] = 0;
        }
    }
}
void Hesse(double A[NMAX][NMAX], double b[NMAX], double H[NMAX][NMAX], double xny[NMAX], double
xgl[NMAX], int n, int m)
{
    double q[NMAX], t[NMAX], g[NMAX], gradgl[NMAX], gradny[NMAX], p[NMAX];

    Gradient(A,b,xgl,gradgl,n,m);
    Gradient(A,b,xny,gradny,n,m);
    for (int i=0;i<m;i++)
    {
        t[i] = xny[i] - xgl[i];
        g[i] = gradny[i] - gradgl[i];
    }
    ProduktMogv(H,g,p,m,m);
    for (int i=0;i<m;i++) q[i] = t[i]-p[i];
    for (int i=0;i<m;i++)
    {
        for (int j=0;j<m;j++) H[i][j] = H[i][j] + (q[i]*t[j]+t[i]*q[j])/Pprodukt(g,t,m) -
(Pprodukt(q,g,m)/pow(Pprodukt(g,t,m),2))*t[i]*t[j];
    }
}
void BFGS(double H[NMAX][NMAX], double grad[NMAX], double sk[NMAX], int m)
{
    double v[NMAX];

    for (int i=0;i<m;i++)
    {
        v[i] = 0;
        for (int j=0;j<m;j++) v[i] += -H[i][j]*grad[j];
    }
    for (int i=0;i<m;i++) sk[i] = v[i]/Lvek(v,m);
}

```

```

void IndkredsningsAlg(double A[NMAX][NMAX], double vb[NMAX], double x[NMAX], double sk[NMAX], double d, int n, int m, double &a, double &b)
{
    double x1, x2, x3;

    x1=0;
    x2=x1+d;
    while (falpha(A,vb,x,sk,x2,n,m)>=falpha(A,vb,x,sk,x1,n,m))
    {
        d=0.1*d;
        x2=x1+d;
    }
    d=2*d;
    x3=x2+d;
    while (falpha(A,vb,x,sk,x2,n,m)>falpha(A,vb,x,sk,x3,n,m))
    {
        x1=x2;
        x2=x3;
        d=2*d;
        x3=x2+d;
    }
    a=x1;
    b=x3;
}
void GoldenSection(double A[NMAX][NMAX], double b[NMAX], double x[NMAX],double sk[NMAX],int n,int m,double &alph)
{
    double a1, a2, a3, a4, c, eps=0.00000001,d=0.1;

    IndkredsningsAlg(A,b,x,sk,d,n,m,a1,a4);
    c=(sqrt(5.0)-1)/2;
    a2=a1+(1-c)*(a4-a1);
    a3=a4-(1-c)*(a4-a1);
    do
    {
        if (falpha(A,b,x,sk,a2,n,m)>=falpha(A,b,x,sk,a3,n,m))
        {
            a1=a2;
            a2=a3;
            a3=a4-(1-c)*(a4-a1);
        }
        else
        {
            a4=a3;
            a3=a2;
            a2=a1+(1-c)*(a4-a1);
        }
    } while((a4-a1)>=eps);
    alph=0.5*(a4+a1);
}

//Analyse af Metode 3:
void AnalyseMetode3(double R[NMAX][NMAX], double P[NMAX][NMAX], double H[NMAX][NMAX], int m)
{
    double Rinv[NMAX][NMAX];
    DanRinv(R,Rinv,m);
    RinvogRinvT(Rinv,P,m);
}

```

```

cout << endl;
cout << "Matricen dannet fra 1/2 * RInv * RInvt er givet ved: " << endl;
UdskrivM(P,m,m);
cout << endl;
cout << "Iterationsmatricen H(m+1) er givet ved: " << endl;
UdskrivM(H,m,m);
}
void Danenhedsvek(double e[NMAX], int m, int k)
{
    for (int i=0;i<m;i++)
    {
        if (i == k) e[i] = 1;
        else e[i] = 0;
    }
}
void DanRinv(double R[NMAX][NMAX], double Rinv[NMAX][NMAX], int m)
{
    double TM[NMAX][NMAX+1], x[NMAX], e[NMAX];

    for (int i=0;i<m;i++)
    {
        Danenhedsvek(e,m,i);
        Totalmatrix(TM,R,e,m);
        BackwardsSubstitution(TM,x,m);
        for (int j=0;j<m;j++) Rinv[j][i] = x[j];
    }
}
void RinvogRinvT(double Rinv[NMAX][NMAX], double P[NMAX][NMAX], int m)
{
    double RinvT[NMAX][NMAX];

    TransponerM(Rinv,RinvT,m,m);
    ProduktMogMT(RinvT,Rinv,P,m,m);
    for (int i=0;i<m;i++)
    {
        for (int j=0;j<m;j++) P[i][j] = P[i][j]/2;
    }
}

```