Numeriske algoritmer (BMECO1901E)

Eksamensopgave i Numeriske algoritmer

Studienummer: **S162378**

CBS - Copenhagen Business School

Erhvervsøkonomi og Matematik

Dato for aflevering: 31/05/2023

Vejleder: Patrick Kristiansen

Omfang:

44.056 tegn med mellemrum – 19.36 normalsider

27 sider i alt

CBS COPENHAGEN BUSINESS SCHOOL
HANDELSHØJSKOLEN

INDHOLDSFORTEGNELSE

1 PROBLEMFORMULERING:

I dette projekt går det ud på at tilpasse en kurve til et sæt støjfyldt data, som repræsenterer en funktion $f(x)$. Formålet er at finde en funktion $g(x)$, der bedst muligt tilnærmer sig funktionen $f(x)$. I dette projekt bruges B-splines til funktion $g(x)$, hvor B-Splines er sammensat af basissplines, der er polynomier. Denne implementeres i et C++-program, der giver brugeren lov til at indtaste værdierne $k\ og\ l$, der bruges til at genere B-Splines.

Til at løse problemet benyttes De Boors algoritme, der skaber en totalmatrix T^{mxn+1}, hvor hver række i totalmatricen repræsenterer funktionsværdierne for en given x-værdi. Derefter anvendes HouseHolders algoritme til mindste kvadraters metode for at opnå den bedst mulige tilpasning af kurven.

Metoderne skal implementeres i et C++-program, og programmet skal også være i stand til at indlæse data fra filer. For at sikre at implementering er udført korrekt, foretages der intern test af programmet. Derudover skal programmet testes eksternt for at vurdere egnetheden af metoderne, hvor der bl.a. vil blive sammenlignet med andre alternativer til mindste kvadraters metoder.

Dermed er formålet at kunne udvikle et velfungerende C++-program, der kan tilpasse regressionssplines til indlæst data ved hjælp af de beskrevne algoritmer til at levere pålidelige resultater.

2 METODEBESKRIVELSE:

2.1 DET GENERELLE PROBLEM

Dette projekt beskæftiger sig med at tilpasse regressionssplines til indlæst data, bestående af punkter $(x_1, y_1), \dots, (x_m, y_m)$, hvor $m \in R^2$, der stammer fra støjfyldte observationer af en funktion $f(x)$.

$$y_i = f(x_i) + \varepsilon_i \quad , \quad for\ i = 1, 2, \dots, m.$$

Hvor ε_i er tilfældig støj, og funktion $f(x)$ i dette projekt vælges til at være:

$$f(x) = \cos(x) + \frac{1}{2}\cos(2x) + \frac{1}{3}\cos(3x)$$

Projektet har fokus på kurvetilpasning, hvor målet er at finde en funktion $g(x)$, der tilnærmelsesvist ligger tæt på funktionen $f(x)$. Med andre ord ønskes det at bestemme $g(x)$, der minimerer udtrykket:

$$\sum_{i=1}^{m} (g(x_i) - y_i)^2$$

Hvor $g(x)$ betragtes som en approksimation af $f(x)$, baseret af de data der er blevet observeret. I dette projekt vil der være fokus på B-Splines. Hvis vi har et vektorrum V med basissplines for B-Splines, $B_1, B_2, \dots, B_n$, kan dette formuleres som et problem, der kan løses med mindste kvadraters metode med $m\ x\ n$ koefficientmatrix. Her bruges De Boors algoritme til at frembringe en totalmatrix, hvorefter Householders algoritme bruges til mindste kvadraters metode:

$$T = \begin{bmatrix} B_1(x_1) & B_2(x_1) & \cdots & B_{n-1}(x_1) & B_n(x_1) & y_1 \\ B_1(x_2) & B_2(x_2) & \cdots & B_{n-1}(x_2) & B_n(x_2) & y_2 \\ B_1(x_3) & B_2(x_3) & \cdots & B_{n-1}(x_3) & B_n(x_3) & y_3 \\ \vdots & \vdots & \cdots & \vdots & \vdots & \vdots \\ B_1(x_m) & B_2(x_m) & \cdots & B_{n-1}(x_m) & B_n(x_m) & y_m \end{bmatrix}$$

Dermed kan funktionen $g(x)$ bestemmes

$$g(x) = \sum_{j=i}^{n} \hat{c_j} B_j(x)$$

Hvor $\hat{c}$ er en løsningsvektor, der kan løse

$$\min_{\hat{\boldsymbol{c}} \in \boldsymbol{R^n}} ||\boldsymbol{y} - A\hat{\boldsymbol{c}}||$$

Dermed er formålet at minimere normen af vektoren $\boldsymbol{y} - A\boldsymbol{c}$. Dette medfører, at den vektor i $Col\ A$, som 'ligger' tættest på **y**, er den ortogonale projektion af **y** ned på A's søjlerum. Sådan et minimeringsproblem kaldes for mindste kvadraters metode.

2.1.1 Mindste kvadraters metode

I dette projekt beskæftiger sig med at finde løsninger til ligningssystemet af $A\boldsymbol{c} = \boldsymbol{y}$. Men her arbejdes der med ligningssystemer med $m \geq n$, og dermed ikke har en traditionel løsning. Dette vil sige, at $\boldsymbol{y} \notin \text{Col A}$, og kan derfor ikke rækkereduceres. Et sådant ligningssystem kaldes et overbestemt ligningssystem. Bestemmelsen af løsningsvektor **c** i dette projekt vil gøres ved hjælp af en QR-faktorisering, hvor der bruges Householders algoritme til at finde den bedste mulige løsning.

2.2 SPLINES

I dette projekt vil rummet af funktioner, V, nemlig bestå af splines.

En funktion g, der kaldes en $spline$, hvis der eksisterer knudepunkter $t_0 = a, t_1, t_2, \dots, t_1 = b$ og polynomier $p_1, p_2, \dots, p_l$, således:

$$g(x) = \begin{cases} p_1(x) & \text{for } t_0 \leq x < t_1 \\ p_2(x) & \text{for } t_1 \leq x < t_2 \\ \vdots & \\ p_l(x) & \text{for } t_{l-1} \leq x \leq t_l. \end{cases}$$

Bemærk at hvis g er den maksimale grad af polynomierne, siges g at være af orden $k = d + 1$. I dette projekt vil vi kræve at splines af orden k er C^{k-2}, altså at afledede funktioner op til og med $k - 2$ eksisterer og er kontinuerte.

Dermed kaldes en spline, der er bestemt ved kurvetilpasning med mindste kvadraters metode, for en regressionsspline.

2.3 KNUDEPUNKTER

I dette projekt skal vi danne en vektor **t** af knudepunkter, der har en særlig udvidelse. Nemlig at de første k skal være lig med det mindste element i **x** og de sidste k knudepunkter skal være lig med det maksimale element i **x**. De resterende skal være ækvidistant fordelt efter et brugervalgt antal interval l. Dette skyldes at vi ønsker at sikre os, at der er nok B-Splines defineret på de første og sidste intervaller. Det vil sige:

$$\underbrace{t_0 = \cdots = t_{k-1}}_{k} < t_k < t_{k+1} < \cdots < \underbrace{t_{l+k-1} = t_{l+k} = \cdots t_{l+2(k-1)}}_{k}$$

Dermed er der netop $n = k + l + 1$ forskellige B-Splines for en given opdeling l intervaller. Ifølge Curry og Schoenbergs sætning, siger de at disse udgør nemlig en basis for vektorrummet af splines af orden k, og dermed kan de derfor bruges til kurvetilpasning med mindste kvadraters metode.

2.4 B-SPLINES

For givne knudepunkter og orden k kan vi skabe B-Splines, der netop er basis for vektorrummet af Splines af orden k eller lavere med disse knudepunkter.

Når Splines er af orden 1, vil den være en flad linje i højden 1 i et interval, og resten vil være 0. Hvis Splines er af orden 2, vil det være en lineær funktion, siden hvis orden k er 2, så er graden for Splines nemlig $2 - 1 = 1$. Hvis en spline derimod er af orden 4, så er den et tredjegradspolynomium.
Grafen for Splines af orden 1, 2, 3 og 4 med t knudepunkter, hvor $t = \{0, 0, 0\ ,0, 1, 2, 3, 3, 3, 3\}$ kan ses under Bilag 1.

For en ikke-aftagende følge af knudepunkter t_j defineres den $j'te$ B-Splines af orden 1 som:

$$B_{j,1}(x) = \begin{cases} 1 & for\ t_j \le x < t_{j+1} \\ 0 & ellers \end{cases}$$

Hvis orden k derimod er højere end 1, altså $k > 1$, defineres den $j'te$ B-Splines rekursivt som

$$B_{j,k} = \omega_{j,k} * B_{j,k-1} + \left(1 - \omega_{j+1,k}\right) * B_{j+1,k-1}$$

$$\omega_{j,k}(x) = \begin{cases} \dfrac{x - t_j}{t_{j+k-1} - t_j} & for\ t_{j+k-1} \neq t_j \\ 0 & ellers \end{cases}$$

Når ordenen k fremgår af sammenhængen kan vi blot skrive B_i i stedet for $B_{i,k}$. Dermed gælder det

$$B_{j,k}(x) = 0\ for\ x \neq [t_j, t_{j+k}]$$

2.5 TOTALMATRIX MED EVALUEREDE B-SPLINES

For at beregne $B_{j,k}$ opstiller man alle værdierne i et trekantsskema således:

$$\begin{matrix} B_{j,1}(x) & \dots & B_{j,k-2}(x) & B_{j,k-1}(x) & B_{j,k}(x) \\ B_{j+1,1}(x) & \dots & B_{j+1,k-2}(x) & B_{j+1,k-1}(x) & \\ B_{j+2,1}(x) & \dots & B_{j+2,k-2}(x) & & \\ \vdots & & & & \\ B_{j+k,1}(x) & & & & \end{matrix}$$

Men siden der kun er én af førsteordensfunktionerne i den første søjle er forskellige fra 0, og 2 i den anden søjle mv., og derfor kan man reducere udregningen ved at springe disse og andre tal, der er 0, over. Det vil sige at hvis vi har et int v, således at $t[v] \leq x\ t_{v+1}$ kan man danne en ny trekantsskema:

$$\begin{matrix} & & & B_{v-k+1,k}(x) \\ & & B_{v-k+2,k-1}(x) & B_{v-k+2,k}(x) \\ & & \vdots & \vdots \\ & B_{v-1,2}(x) & \dots\ B_{v-1,k-1}(x) & B_{v-1,k}(x) \\ B_{v,1}(x) & B_{v,2}(x) & \dots\ B_{v,k-1}(x) & B_{v,k}(x). \end{matrix}$$

Det vil sige, at vi nu kun får alle ikke-nulværdier B-Splines af orden til og med k. Dette er især nyttigt, når den skal bruges til at danne koefficientmatricer. Når vi skal bestemme værdierne, vil det foregå følgende:

Vi antager at vi har $j'te$ søjle i trappematricen foroven gemt i vektoren $\boldsymbol{b} = (b_0, b_1, \dots, b_{j-1})^T$, kan de bruges til at bestemme $\boldsymbol{b}' = (b'_0, b'_1, \dots, b'_j)^T$ i søjle j, der indeholder $B_{v-j,j+1}, B_{v-1,j+1}, \dots, B_{v-1,j+1}, B_{v,j+1}$

Dette kan gøres sådan:

$$b'_r = (x - t_{v-j+r}) * \frac{b_{r-1}}{t_{v+r} - t_{v-j+r}} + (t_{v+1+r} - x) * \frac{b_r}{t_{v+r+1} - t_{v-j+r+1}} \quad for\ r = 0, 1, \dots, j.$$

Hermed kan alle værdierne hos b'_r bestemmes ved hjælp af en kombination af $b_{r-1}\ og\ b_r$, dvs. værdierne fra søjlen før b_r

For at gøre det lidt mere overskuelige indføres en notation:

$$d_s^L = x - t_{v-s}, \quad d_s^R = t_{t+1+s} - x$$

Hvor L og R står hhv. for 'left' og 'right'.

Dermed kan b'_r nu skrives som:

$$b'_r = d_{j-r}^L * \frac{b_{r-1}}{d_{j-2-r}^L - d_{r-1}^L} + d_r^R * \frac{b_r}{d_{j-1-r}^L - d_r^R} \quad for\ r = 0, 1, \dots, j.$$

2.6 QR-FAKTORISERING

QR-faktorisering er en numerisk metode, der skriver en matrix A som et produkt af en ortogonal matrix Q og en øvre trekantsmatrix R. Dette kan nemlig bruges til at løse mindste kvadraters metode.

En par konklusioner om ortogonale matricer. Vi har Q-matrix: $Q \in \mathbb{R}^{nxn}$, hvor

$$q_l, \dots, q_2 q_1 = Q^T$$

Og Q er ortogonal, altså:

$$Q^T Q = I$$

Transformationen med ortogonale matricer ændrer ikke på normen, dermed er det en isometrisk transformation. Det vil sige at hvis en vektor y transformeres ved hjælp af en ortogonal matrix, fx Q, vil normen af y være lig med normen af den oprindelige vektor x:

$$\boldsymbol{y} = Q\boldsymbol{x} \rightarrow \left|\left|\boldsymbol{y}\right|\right| = ||\boldsymbol{x}||$$

$$\left|\left|\boldsymbol{y}\right|\right|^2 = \boldsymbol{y}^T\boldsymbol{y} = (Q\boldsymbol{x})^T(Q\boldsymbol{x}) = \boldsymbol{x}^T Q^T Q\boldsymbol{x} = \boldsymbol{x}^T\boldsymbol{x} = \left|\left|\boldsymbol{x}\right|\right|^2$$

Det vil sige, at når vi anvender isometriske transformationer, forstærkes afrundingsfejlene ikke, dermed opfører det sig numerisk bedre.

Disse konklusioner bruges i udledningen af løsningsligningen for **c**., hvor $A \in \mathbb{R}^{mxn}$

$$A\boldsymbol{c} = \boldsymbol{y}$$

Vi ønsker en QR-faktorisering baseret på ortogonale transformationer med en matrix A:

$$A = QR$$

$$QR\boldsymbol{c} = \boldsymbol{y}$$

$$Q^T QR\boldsymbol{c} = Q^T\boldsymbol{y}$$

$$R\boldsymbol{c} = Q^T\boldsymbol{y}$$

Og dette kan løses via.

$$R\hat{\boldsymbol{c}} = Q^T\boldsymbol{y}$$

Vi vil ikke bestemme Q her, da vi er kun interesseret i $Q^T\boldsymbol{y}$

Grunden til at denne kan bruges til at løse mindste kvadraters metode er at vi ønsker at minimere

$$\min_{\boldsymbol{c}\in\boldsymbol{R}^n} ||\boldsymbol{A}c - \boldsymbol{y}||$$

2.7 HOUSEHOLDER-TRANSFORMATION

Householder-transformation er en lineær transformation, der bruges på en totalmatrix, der kan transformere totalmatricen ved hjælp af spejlinger. Denne transformation kan bruges til QR-faktorisering af $m\ x\ n$ matricer med $m \geq n$, hvor den kan reducere matricen til en øvre trekantsmatrix R $(m\ x\ m)$ uden at beregne Q, hvor R skal bruges til mindste kvadraters metode. Fordelen ved at R er en øvre trekantsmatrix, er at løsningen $R\boldsymbol{c} = Q^T\boldsymbol{y}$ blot kan kræve baglæns substitution til at udregne løsningsvektor **c**.

Dette gøres ved hjælp af flere algoritmer. Den første algoritme er en, der danner vektor **v**, der repræsenterer en Householder-transformation, der kan frembringe nuller i en given søjle af matricen T. Så hvis vi har en standardbasis vektor: $\boldsymbol{e}_1 = (1,0,\dots,0)^T$, så ønsker vi en Householder transformation H, så $H\boldsymbol{a}$ er et multiplum af $\boldsymbol{e}_1$.

Hvis vi vælger

$$\boldsymbol{v} = \boldsymbol{a} - \big|\big|\boldsymbol{a}\big|\big|\boldsymbol{e}_1$$

hvor vektor **a** er en bestemt vektor i totalmatricen T, som vi ønskes at skabe nuller i.

Når vektor **v** sættes ind i matricen H, fås:

$$H = I - \frac{2}{\big|\big|\boldsymbol{v}\big|\big|}\boldsymbol{v}\boldsymbol{v}^T \rightarrow H\boldsymbol{a} = \big|\big|\boldsymbol{a}\big|\big|\boldsymbol{e}_1$$

Vores valg af vektor **v** har netop det samme koordinater som **a** bortset fra den første som er:

$$\boldsymbol{v}_1 = \boldsymbol{a}_1 - ||\boldsymbol{a}||$$

Men der kan være et problem med dette valg, nemlig at hvis vektor **a** er tæt på at være et positivt multiplum af $\boldsymbol{e}_1$**,** kan der forekomme nogle store afrundingsfejl. For at 'forebygge' dette, bruges

$$\boldsymbol{v}_1 = \frac{-(\boldsymbol{a}_2^2 + \boldsymbol{a}_3^2 + \cdots + \boldsymbol{a}_m^2)}{\boldsymbol{a}_1 + ||\boldsymbol{a}||}$$

Men for at gøre det nemmere er algoritmen formuleret således, at den arbejder direkte med totalmatricen T i stedet for en vektor **a**.
For at kunne skabe nuller i søjler i matricen, skal Householder matrix H ganges på matrix A. Efter flere Householder transformationer fås:

$$HA = H_{n-1} \dots H_2 H_1 A = [r_1\ r_2 \dots\ r_n] = Q^T A = R$$

Dermed får vi, når H ganges på $T = [A \quad \boldsymbol{y}] \rightarrow [R \quad Q^T\boldsymbol{y}]$

Householder-transformation er en bestemt type algoritme, der er en ortogonal transformation.
Hvis vi har en enhedsvektor **u**, altså $\big|\big|\boldsymbol{u}\big|\big| = 1$
Så kan transformationen resulterer i en matrix H, hvor H:

$$H = I - 2\boldsymbol{u}\boldsymbol{u}^{\boldsymbol{T}}$$

Matricen H har en særlig egenskab, nemlig at den er symmetrisk, $H^T = H$

$$H^T = I^T - 2(\boldsymbol{u}\boldsymbol{u}^T)^T = I - 2\boldsymbol{u}\boldsymbol{u}^T = H$$

Og H er også en ortogonal matrix: $H^T H = I$, hvilket kan ses her:

$$H^TH = H^2 = \left(I - 2\boldsymbol{uu^T}\right)\left(I - 2\boldsymbol{uu^T}\right) = I - 2\boldsymbol{uu^T} - 2\boldsymbol{uu^T} + 4\boldsymbol{u}\underbrace{\boldsymbol{u^Tu}}_{I}\boldsymbol{u^T} = I$$

Så hvis vi ganger matrix H med en enhedsvektor **u**:

$$H\boldsymbol{u} = \boldsymbol{u} - 2\boldsymbol{uu^Tu} = -\boldsymbol{u}$$

Og vektor **x** står vinkelret på vektor **u**, altså $\boldsymbol{x} \perp \boldsymbol{u}$

$$H\boldsymbol{x} = \boldsymbol{x} - 2\boldsymbol{uu^Tx} = \boldsymbol{x}$$

Hvis vi bestemme en vektor **v**, der …

$$\left|\left|v\right|\right| \neq 1 \; og \; \left|\left|v\right|\right| \neq 0$$

Så kan vektor **u** bestemmes:

$$\boldsymbol{u} = \frac{1}{||\boldsymbol{v}||}\boldsymbol{v}$$

Så matrix H kan nu også bestemmes på en ny måde:

$$H = I - \frac{2}{\left|\left|\boldsymbol{v}\right|\right|}\boldsymbol{vv}^T$$

Når man har foretaget en HouseQR, vil totalmatricen T være blevet reduceret til

$$\begin{bmatrix} R & Q^Ty \\ 0 & * \end{bmatrix}$$

Hvor $*$ er nogle vilkårlige tal.

2.8 BAGLÆNS SUBSTITUTION

Baglæns substitution er en numerisk metode, der bruges til at løse et lineært ligningssystem, hvor matricen er en øvre trekant-matrix, R $(m\ x\ m)$, som fx $R\hat{\boldsymbol{c}} = Q^T\boldsymbol{y}$ i dette projekt**.** Metoden går ud på at man beregner fra bunden af ligningssystemet, altså at løse den nederste ligning af matricen, hvor man så substituerer den ind i den næstsidste ligning og dette fortsætter indtil man har udregnet alle c-værdier.

Det vil sige, at man bruger en totalmatrix som argument. Men her deles totalmatrix op i to dele, R (øvre trekantmatrix) og **v** (konstantsøjlen)

Nedenunder kan man se virkemåden ved baglæns substitution:

$$\begin{aligned} r_{11}c_1 + r_{12}c_2 + r_{13}c_3 + \cdots + r_{1n}c_n &= v_1 \\ r_{22}c_2 + r_{23}c_3 + \cdots + r_{2n}c_n &= v_2 \\ r_{33}c_3 + \cdots + r_{3n}c_n &= v_3 \\ &\vdots \\ r_{nn}c_n &= v_n \end{aligned}$$

Og løsningsvektor **c** kan løses således:

$$\begin{aligned} c_n &= \frac{v_n}{r_{nn}} \\ c_{n-1} &= \frac{v_{n-1} - r_{n-1,n}x_n}{r_{n-1,n-1}} \\ c_1 &= \frac{v_1 - r_{12}x_2 - \cdots - r_{1n}x_n}{r_{11}} \end{aligned}$$

Men metoden baglæns substitution i dette projekt skal modificeres lidt, da vi ikke har en $R\hat{\boldsymbol{c}} = Q^T \boldsymbol{y}$, men derimod en totalmatrix: $\begin{bmatrix} R & Q^T y \\ 0 & * \end{bmatrix}$

Derfor skal den modificeres lidt, så man starter ved R's sidste række, og dermed droppes ligningerne, der ligger under R. Det vil sige at 'm' initialiseres til $n-1$, hvor n angiver antal søjler i totalmatricen. Men fordi R er en $n\ x\ n$, vil det sige at m blot skal være $n-1$.

3 DOKUMENTATION AF KODE:

3.1 DIAGRAM OVER PROGRAMMETS STRUKTUR:

Programmets grundstruktur er beskrevet i et flowdiagram. Flowdiagrammet kan ses til højre. Herunder kommer der en kort beskrivelse af, hvordan programmet fungerer.

Programmet startes ved at kalde på funktionen *'velkomsttekst()'*, hvor der kommer en kort introduktion til programmet.
Derefter indlæses programmet data fra filen *'x.txt'* og *'y.txt'*, hvor de hhv. er x- og y-punkter fra støjfyldte observationer af en funktion.

Der kaldes på funktionen *'minmax()'*, der returnerer den mindste og største værdi fra vektor **x** som et par.
Derefter beder programmet brugeren om at indtaste orden k og antal intervaller l.

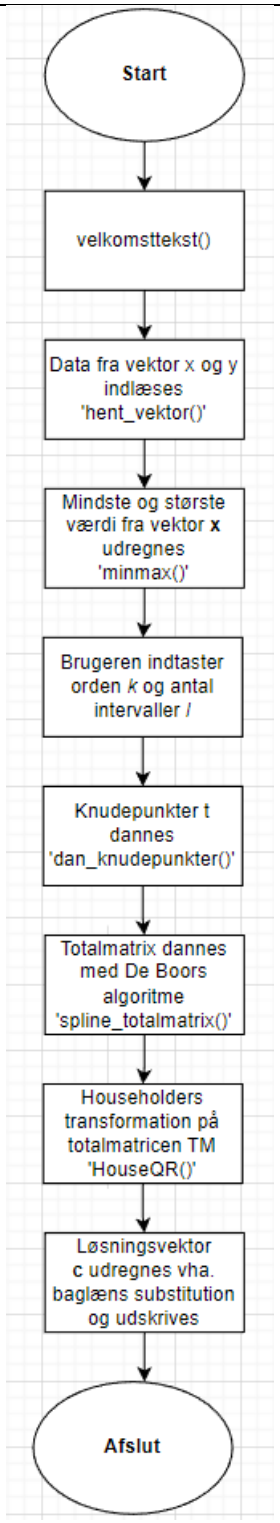
Nu kan der dannes knudepunkter t vha. mindste og største værdi fra vektor **x** og k, l ved en funktionskald på *'dan_knudepunkter()'*.

Derefter dannes der en totalmatrix ved hjælp af De Boors algoritme ved en funktionskald på *'splines_totalmatrix()'*.
Derefter løses kvadratminimeringsproblemet med Householder-transformation med funktionen *'HouseQR'*, hvor der returneres en ny totalmatrix:

$$\begin{bmatrix} R & Q^T y \\ 0 & * \end{bmatrix}$$

Til sidst udregnes $R\boldsymbol{c} = \boldsymbol{y}$ med baglæns substitution under den antagelse af at R er en øvre trekantsmatrix, og løsningsvektoren **c** udskrives.

Programmet afsluttes.



3.2 OVERSIGT OVER DE IMPLEMENTEREDE FUNKTIONER:

I dette projekt er der nogle bestemte funktioner, der skal implementeres. Nedenfor gennemgås de forskellige implementerede funktioner.

3.2.1 Funktion minmax

`pair<double, double> minmax(const Vektor& x)`	
Formål:	Formålet er at kunne returnere den mindste og største værdi fra parameteren x.
Input:	const Vektor& x: En vektor **x**, der indeholder x-punkterne.
Output:	Returnerer den mindste og største værdi i vektoren **x** som et par.
Virkemåde:	Variablene 'xmin' og 'xmax' initialiseres med den første værdi i vektor **x**. Derefter gennemgås den alle elementerne i vektoren **x** vha. en **i** for-løkke, der køres fra i=0 til x.size(). Hvis elementet er mindre end "xmin", opdateres "xmin" med den nye mindste værdi. Hvis elementet derimod er større end "xmax", opdateres "xmax" med den nye største værdi. Efter for-løkken har gennemkørt hele vektoren **x,** returneres funktionen et par (pair), hvor den mindste værdi og den største værdi er hhv. repræsenteret ved "xmin" og "xmax".

3.2.2 Funktion dan_knudepunkter

`Vektor dan_knudepunkter(double tmin, double tmax, int l, int k)`	
Formål:	Formålet er at danne en vektor med knudepunkter, der har denne udvidelse, at de første k knudepunkter skal være lig med det mindste element fra vektor x, samt de sidste k knudepunkter skal være lig med det maksimale element i vektor x.
Input:	double tmin: Det mindste element fra vektor **x**. double tmax: Det maksimale element fra vektor **x**. int l: Antal intervaller int k: Orden på splines.
Output:	Returnerer t med knudepunkter
Virkemåde:	Vektor t initialiseres med størrelsen $n = k * 2 + l - 1$, da der skal være $2 * k$ samt $l - 1$. Derefter udføres en **i** for-løkke, hvor de første k sættes lig med tmin, og de sidste k knudepunkter sættes lig med tmax. Derefter beregnes ækvidistant dx, som er afstanden mellem knudepuntkerne i intervallet. En **j** for-løkke køres for at danne de midterste knudepunkter, der er ækvidistant fordelt. Til sidst returneres vektor **t.**

3.2.3 Funktion vknude

`int vknude(const Vektor& t, double x)`	
Formål:	Formålet er at kunne returnere indekset v, der er mindre eller lig med x-værdi, mens den næste v-indeks er større end x. Mao. $t[v] \leq x < t[v+1]$
Input:	const Vektor& t: Vektor t, der indeholder knudepunkter

	double x: x-værdi, der stammer fra x-vektor.
Output:	Returnerer int v, der repræsenterer den knudepunkt i vektoren t, som er tættest på eller mindre end værdien "x".
Virkemåde:	Først tjekkes det, om x er mindre end den første knudepunkt: $x < t[0]$ – Hvis det er sandt, returneres der 0. Derefter køres en **v** for-løkke fra 0 til antallet af knudepunkter 't.size()'. I løkken gennemgås hvert element i vektoren t, hvor der tjekkes om det aktuelle element er større end 'x'. Hvis dette er sandt, returneres indekset's for det foregående knudepunkt (v-1) som resultat. $$if(t[v] > x)\ \{return\ v-1\}$$ Hvis for-løkken er færdig, returneres det sidste indeks i vektor **t** (n-1).

3.2.4 Funktion eval_spline

`Vektor eval_spline(const Vektor& t, int k, int v, double x)`	
Formål:	Formålet er at evaluere alle ikke-nul B-Splines af orden k (grad k-1) med knudepunkter i t fra vektor x.
Input:	const Vektor& t: Vektor t, der indeholder knudepunkter int k: orden k int v: Parameteren v er valgt ud fra $t[v] \leq x < t[v+1]$ double x: x-værdi
Output:	Returnerer en vektor, der indeholder B-Splines af orden k.
Virkemåde:	Først initialiseres tre vektorer, $b(k), dL(k-1), dR(k-1)$, hvor k er en given parameter af orden k. Derefter indføres notationen: $$d_s^L = x - t_{v-s}$$ $$d_s^R = t_{v+1+s} - x$$ Dette gøres vha. en for-løkke, der køres fra 0 til $k-2$ Derefter sættes $b[0]\ til\ 1$ En ny for-løkke køres til $k-1$, hvor denne beregner 'saved', der er en ny værdi til vektor b. Først initalieres $saved$ med 0, og derefter køres en indre for-løkke. Denne for-løkke køres fra $0\ til\ j-1$ og bruges til at beregne de to variabler $'term'\ og\ 'saved'$. Hvor af værdien i $b[r]$ opdateres ved at tilføje 'saved' og $dR[r] * term$; Til sidst opdateres $b[j]$ med 'saved' Til sidst returneres **b** vektor, der indeholder de beregnede B-Splines basisfunktioner.

3.2.5 Funktion spline_totalmatrix

```
Matrix spline_totalmatrix(const Vektor& t, int k, const Vektor& x,
                          const Vektor& y)
```

Formål:	Formålet er at kunne returnere en totalmatrix med Bsplines for den pågældende x-værdi i hver søjler samt vektor y på konstantsøjle.
Input:	const Vektor& t: Vektor t, der indeholder knudepunkter int k: orden k const Vektor& x: Vektor x, der indeholder x-punkterne. const Vektor& y: Vektor y, der indeholder støjfyldte y-punkter
Output:	En totalmatrix $[B_n(x_m) \quad y_m]$. Det vil sige at den returnerer en totalmatrix som indeholder alle de evaluerede Bsplines samt konstantsøjlen som stammer fra y-vektor.
Virkemåde:	Først initialiseres størrelsen 'm' og 'n'. m er størrelsen af vektor x ($x.size()$), mens n er er størrelsen af vektor **t** – k. Matrix 'TM' initialiseres med størrelsen $(m, n+1)$ vha. funktionen dan_matrix, da vi også skal have konstantsøjlen med, derfor +1 på antallet af søjler (n). En **i** for-løkker køres fra 0 til m, og køres gennem hver rækker i matricen 'TM'. Her beregnes v-værdi vha. vknude, der bruges til at finde det indeks, hvor x[i] skal placeres i vektor **t**, for at holde på ordenen k. Dernæst tjekkes der om v er større eller lig med $n-1$. Hvis det er sandt, opdateres $v\ til\ n-1$. Dette sikrer at eval_spline ikke læses ud over enden af vektor **t**. Dermed skabes 1-tal på den sidste række og søjlen i koefficientmatricen, og dermed vil totalmatricen være konsistens. Vektor **b** initialiseres, der indeholder værdierne af B-Splines for intervallet bestemt af **v,** vha. funktionen $'eval\_spline(t,k,v,x[i])'$. Dernæst køres en indre **j** for-løkke, der itererer gennem hver søjler i den aktuelle række (**i**) i totalmatricen 'TM'. Her tilføjes de beregnede værdier af basisfunktioner indtil kontantsøjlen. $\mathrm{TM[i][j+v-(k-1)]} = \mathrm{b[j]};$ Bemærk at vi skriver $[j+v-(k-1)]$ skyldes at vi ønsker at 'søjleforskyde' Bsplines efter hvert nyt knudepunkter. Til sidst tilsættes værdien i konstantsøjlen $(TM[i][n] = y[i])$ med værdien fra y-vektor. Når matricen er blevet udfyldt, returneres den færdige matrix 'TM'.

3.2.6 Funktion house

`Vektor house(const Matrix& T, int j)`	
Formål:	Formålet er at danne en vektor v for en Housholder-transformation, der kan skabe nuller i søjle j af totalmatricen T.
Input:	const Matrix& T: Totalmatrix ved $T \in R^{m \times n+1}$ int j: Parameteren j, der angiver den valgte j-søjle.
Output:	Returnerer en vektor **v**, der repræsenterer en Householder-tranformation.
Virkemåde:	Variabel 'Sigma' initialiseres til 0: $\sigma = 0$

	En **i** for-løkke køres fra 1 til $m - j$. Denne løkke opdaterer sigma ved at summere $T[i + j][j]$, hvor den er blevet kvadreret. Samtidig gemmes elementet i vektoren **v**: $v[i] = T[j + i][j]$ Efter for-løkken initialiseres en ny variabel 'my' ved at tage kvadratoden af kvadratet af $T[j][j]$ og Sigma. my = sqrt(pow(T[j][j],2) + sigma) En **if**-betingelse startes, der tjekker om værdien i $T[j][j]$ er mindre eller lig med 0. Hvis det er sandt, opdateres den første element i vektor **v**, med $T[j][j] - my$ Hvis ikke, opdateres den første element i vektor **v**, med $-sigma \ / \ T[j][j] + my$ Til sidst returneres vektoren **v** som resultat.

3.2.7 Funktion anvend_house

`void anvend_house(Matrix& T, const Vektor& v, int j)`	
Formål:	Formålet er at anvende house-transformation på T's undermatrix.
Input:	Matrix& T: Totalmatrix ved $T \in R^{m \times n+1}$ const Vektor& v: Vektor **v**, der er en Householder-transformation repræsenteret int j: Offset j, der angiver den del af matricen T, der skal transformeres.
Output:	Funktionen returnerer ikke noget, men der anvendes en House-transformation på T's undermatrix.
Virkemåde:	En **k** for-løkke køres fra 0 til og med n-j. Pi og gamma initialiseres til 0. En ny **i** indre for-løkke startes og køres fra 0 til$m - j$. Pi opdateres ved at summere produktet af $v[i] * T[j + i][j + k]$ Gamma opdateres ved at summere kvadraterne af $v[i]$ **i** løkken afsluttes. En ny variabel 'beta' initialiseres med brøkformatet $2.0/gamma$. En ny indre **i** for-løkke startes fra 0 til $m - j$ Her trækkes $beta * pi * v[i]$ fra $T[j + i][j + k]$ for hver iteration. Begge for-løkker afsluttes.

3.2.8 Funktion HouseQR

`void houseQR(Matrix& T)`	
Formål:	Formålet er at køre en HouseHolder QR-transformation for totalmatrix T, så T bliver reducerert til en ny matrix:

	$\begin{bmatrix} R & Q^T y \\ 0 & * \end{bmatrix}$
Input:	Matrix& T: Totalmatrix ved $T \in R^{m \times n+1}$
Output:	Funktionen returnerer en ny matrix, hvor R er en øvre trekant af T i koefficientmatrix og $Q^T y$ på konstantsøjlen
Virkemåde:	Størrelsen af matricen T gemmes i variablerne m og n, og n reduceres med 1. En **j** for-løkke køres fra 0 til n. Vektor **v** dannes vha. "house" og bruges til en Householder-transformationen på matricen T vha. funktionen "anvend_house". Denne proces gentages for hver søjler i matricen T, hvilket gør at T bliver reduceres til $[R \quad Q^T y]$

3.2.9 Funktion bsub

Vektor **bsub**(**const** Matrix& T)	
Formål:	Formålet er at løse $R\boldsymbol{c} = \boldsymbol{v}$, hvor R er en øvre trekantsmatrix.
Input:	const Matrix& T: Totalmatrix ved $T \in R^{m \times n+1}$ Dog er T = $\begin{bmatrix} R & Q^T \boldsymbol{y} \\ 0 & * \end{bmatrix}$
Output:	Returnerer løsningsvektoren **c**.
Virkemåde:	Variablen 'n' initialiseres med antal søjler i den første række i matrix T. Variablen 'm' initialiseres til $n-1$, da vi kun ønsker at fokusere på $[R \quad Q^T \boldsymbol{y}]$ og ikke diverse nuller/vilkårlige tal under den øvre trekantmatrix R og $Q^T \boldsymbol{y}$. Løsningsvektor **c** initialiseres med størrelsen m. Den sidste indeks i vektor **c** udregnes først: $c[x-1] = \frac{T[m-1][m]}{T[m-1][m-1]}$ En **i** for-løkke køres fra $m-2$ til 0, hvor den 'tæller nedad' vha. $i--$. Variablen 'sum' initialiseres til 0 En indre **j** for-løkke køres fra $i+1\ til\ m$. I hver iteration summeres produktet af $T[i][j] * c[j]$ til variablen 'sum'. Den indre for-løkke afsluttes. De nye løsninger skrives ind i vektor **c.** $c[i] = T[i][m] - sum\ /\ T[i][i];$ Løsningsvektor **c** returneres.

4 TEST

Programmet testes internt for at sikre, at C++ koden samt diverse funktioner fungerer som den skal. Til at tjekke om disse funktioner fungerer korrekt, vil der bl.a. blive sammenlignet med værdierne fra det troværdige CAS-værktøj, Mathematica.

5 INTERN TEST

5.1 TEST AF MINMAX

Her testes funktionen '*minmax(const Vektor& x)'* , for at tjekke om funktionen kan returnere den mindste og største værdi fra parameteren x, der er en vektor.

Som det kan ses i Bilag 2, returnerer funktionen 'minmax' den mindste og største værdi fra vektor x. Dermed kan vi konkludere at funktionen fungerer korrekt.

5.2 TEST AF DAN_KNUDEPUNKTER

Her testes funktionen 'dan_knudepunkter' med henblik på om at der bliver dannet en vektor med de korrekte knudepunkter. Forskriften for knudepunkter kan ses i metodeafsnittet: Her testes der om de første k er lig med det mindste element i den indlæste **x-vektor**, og de k sidste skal være lig med det maksimale element i **vektor x**. De resterende skal være ækvidistant fordelt efter et brugervalgt antal underinterval l

Funktionen testes ved at tage mindste element i vektor **x**, som er 0, og det maksimale element som er 3. Og hvis brugeren fx har valgt $k = 4, l = 3$, bør knudepunkterne være $t = \{0, 0, 0, 0, 1, 2, 3, 3, 3\}$.

Som dette ses i Bilag 3, udregner funktionen korrekt, når man sammenligner med resultatet, der er opgivet i eksamenssættet.

5.3 TEST AF VKNUDE

Her testes funktionen 'vknude' ved at kontrollere om den returnerer det rigtige indeks v, nemlig når $t[v] \leq x < t[v+1]$.
Som det ses i Bilag 4, køres programmet med $k = 4, l = 3$, hvilket viser at funktionen returnerer det rigtige indeks baseret på den givne x-værdi. Som et eksempel kan vi tage $x = 1.2$. Her kan vi se, at den ligger mellem knudepunktet 1 og 2, og dermed skal funktionen returnere $t = 1$, dvs. $v = 4$, da $t[4] = 1$.
Dermed kan det konkluderes, at funktionen returnerer den korrekte indeks v.

5.4 TEST AF EVAL_SPLINE (B-SPLINES)

Her testes funktionen '*eval_spline'* ved at tjekke om funktionen returnerer alle B-Splines af orden op til og med k, der er **forskellige** fra nul, som følge af De Boors algoritme. Det vil sige, at hvis ordenen k er 4, så skal funktionen gerne kunne returnere en vektor **b** med 4 forskellige værdier. Værdierne testes ved at sammenligne med det troværdige CAS-værktøj, Mathematica.

Testning kan ses i Bilag 5, hvor programmet opnår de samme værdier som Mathematica, hvilket kan konkluderes at funktionen fungerer korrekt.

5.5 TEST AF SPLINE_TOTALMATRIX

Her testes funktionen '*spline_totalmatrix*' med det formål at kunne skabe en totalmatrix, hvor koefficientmatricen indeholder alle B-Splines. Hver række i koefficientmatricen udregnes ved hjælp af De Boors algoritme, der bestemmer funktionsværdierne fra vektor x. Derudover skal y-vektor indsættes på konstantsøjlen i totalmatricen.

Funktionen testes med at sammenligne koefficientmatricen udregnet i Mathematica med mit C++-program, for at være sikker på at funktionsværdiernw fra B-Splines er udregnet korrekt med De Boors algoritme.

Hermed kan der konkluderes fra Bilag 6, at funktionen '*spline_totalmatrix'* kan returnere den korrekte totalmatrix.

5.6 TEST AF HOUSE

Her testes funktionen '*house'* med henblik på at kunne danne vektor **v**, der repræsenterer en Householder-transformation. Denne transformation skal kunne skabe nuller i given søjle af matricen T. Men vi fokuserer her kun på at kontrollere, om funktionen returnerer den korrekte vektor **v**, når den sammenlignes med resultaterne udregnet i Mathematica.

Hos Mathematica testes vi med udtrykket: $\boldsymbol{v_n} = \boldsymbol{a_n} - ||\boldsymbol{a_n}|| * e_n$, hvor $\boldsymbol{a_n}$ angiver n søjle i matrix T, og $\boldsymbol{e_n}$ er en enhedsvektor. Hos Mathematica ganges hver vektor $\boldsymbol{a}_n$ med $\{1,1,1,1,1,1\}^T$, afhængighed af hvilken søjle i T vi er i gang med. Hvis vi for eksempel er i gang med den 3. søjle, altså $\boldsymbol{a}_3$, skal den ganges med $\{0,0,1,1,1,1\}^T$

Det fulde testresultaterne kan ses i Bilag 7. Som man kan se, startes den første søjle $j = 0$ hos C++, mens hos Mathematica er det $v1$. Dette skyldes at vektoren i C++ er nulindekseret. Dermed kan man se, at funktionen returnerer den samme vektor **v**, som Mathematica, og dermed kan det konkluderes, at funktionen house udregner korrekt.

5.7 TEST AF ANVEND_HOUSE

Funktionen '*anvend_house*' testes internt med algoritmen, der tager vektor **v**, der repræsenterer en Householder-transformation fra funktionen '*house'*, og anvender den på matricen T. Dermed skal man kunne se, at der dannes nuller i den givne søjle af matricen T. Resultaterne sammenlignes med Mathematica, hvor matricen H ganges med den oprindelige totalmatrix for at få den nye totalmatrix. Beregningen af Householder-matricen H er beskrevet i metodeafsnittet.

Her vises et uddrag af testningen, og resten af testresultaterne kan ses i Bilag 8. Som det kan ses, dannes der nuller i den givne søjle i matricen T ved hjælp af vektor **v.** Når dette sammenlignes med Mathematica fås de samme resultater, hvilket konkluderer at funktionen fungerer korrekt.

5.8 TEST AF HOUSEQR

Testning af funktionen *'HouseQR*' har to forskellige tests. De kan findes i Bilag 9. Den første test går ud på at sammenligne den øvre trekantsmatrix R fra C++ med Mathematica, hvor den øvre trekantsmatrix udregnes ved hjælp af '*QRDecomposition'*. Den anden test går ud på at teste algoritmen, der reducerer totalmatrix TM, som blev bestemt i Splines_totalmatrix, til

Denne gang vil vi teste programmet med 10 nye punkter med brugervalgt $k = 4, l = 3$. Som det ses, fås C++-program det samme resultat som Mathematica, og ikke mindst kan man se, at funktionen reducerer totalmatricen T til $\begin{bmatrix} R & Q^T y \\ 0 & * \end{bmatrix}$. Læg mærke til de blå strenger fra C++-program nedenunder. Hos Mathematica fås R med nogle negative tal. Men dette har ikke noget betydning, da en QR-faktorisering er kun entydig op til fortegn. Det vil sige, så længe $A = QR$, er fortegnet ligegyldigt.

Derfor kan der konkluderes, at funktionen HouseQR udregner korrekt.

5.9 TEST AF BSUB

Testning af funktion '*bsub'*, der løser $R\boldsymbol{c} = Q^T\boldsymbol{y}$ med baglæns substitution fungerer på den måde, at resultaterne fra C++ sammenlignes med Mathematica's udregning af løsningsvektor **c** ved hjælp af '*LeastSquares[A, y]'*, der løser mindste kvadrats metode problem for en matrix $A\boldsymbol{c} = \boldsymbol{y}$.

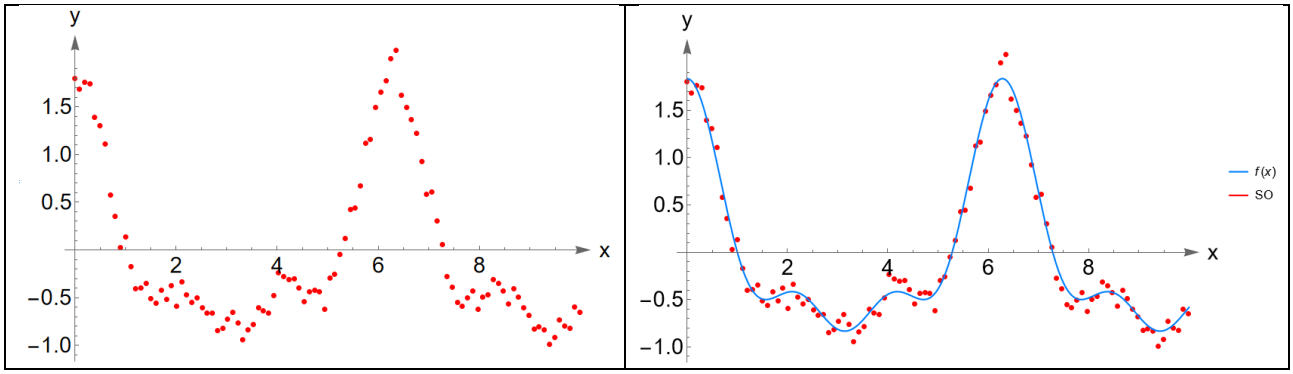
Testning af bsub kan findes i Bilag 10. Dermed kan der konkluderes at funktionen returnerer den korrekte værdi af løsningsvektor **c**, når der sammenlignes med Mathematica.

6 EKSTERN TEST

Her foretages programmet en ekstern test for at vurdere metodens egnethed. Til testningen bruges 100 punkter, der stammer fra støjfyldte observationer af funktionen $f(x)$.

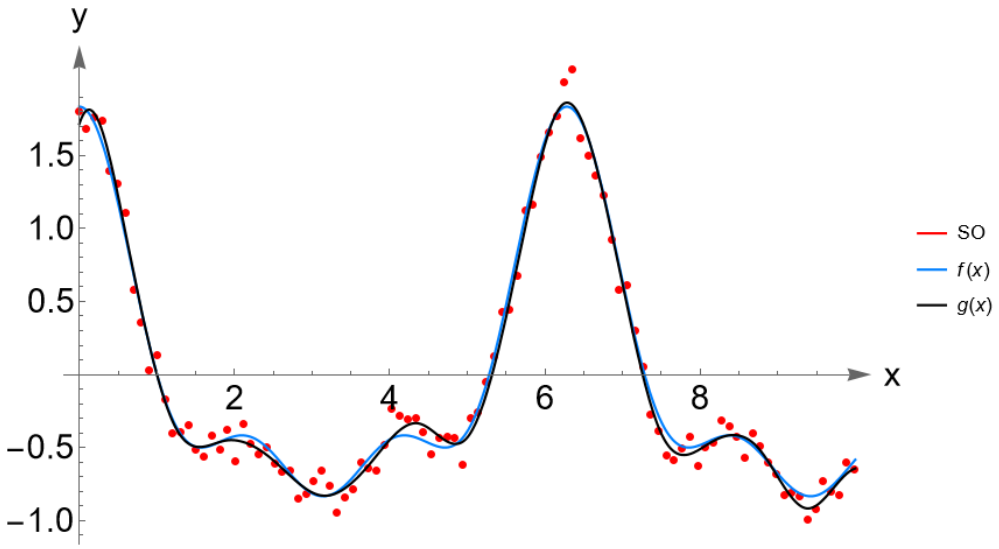
6.1 PLOT AF GRAFEN $g(x)$ med $k = 4, l = 16$

Først plottes grafen $g(x)$, der er blevet udregnet igennem vores program.



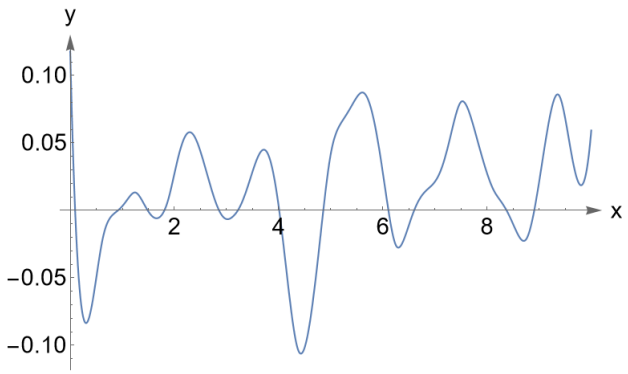
Disse punkter er blevet skabt ud fra funktionen $f(x)$ med støj. Hvis man plotter støjfyldte observationer, der forkortes til SO sammen med funktionen $f(x)$:

Når vi derimod prøver at danne B-Splines regression $g(x)$ ved $k = 4, l = 16$. Hvis man plotter punkterne og de to funktioner $f(x)$ & $g(x)$



Dermed kan man se at vi har forsøgt at efterligne $g(x)$ som funktionen $f(x)$

Hvis vi kigger på afvigelsen mellem $f(x)$ & $g(x)$, kan man se, at der ikke skabes en stor afvigelsen mellem de to funktioner.



6.2 En kort forklaring af udregningen af residualer

For at kigge på, hvordan resultaterne afhænger af k og l vil vi analysere residualerne på to forskellige måder.

Den første måde er at beregne residualer mellem funktionen $g(x)\ og\ f(x)$. Dette skyldes, at vi også ønsker en kurvetilpasning, hvor målet er at finde en funktion $g(x)$, der tilnærmelsesvist ligger tæt på funktionen $f(x)$, uden at den skal gå igennem alle punkterne. Dette kan gøres ved hjælp af:

$$g(x) - f(x) = \sqrt{\sum_{i=1}^{m} \big(g(x_i) - f(x_i)\big)^2}$$

Den anden måde er at beregne forskellen mellem de støjfyldte observerede punkter og selve funktionen $g(x)$. Selve residualerne beregnes som:

$$\big||r|\big| = \sqrt{\sum_{i=1}^{m} (g(x_i) - y_i)^2}$$

Hvor m angiver antal punkter, der er blevet observeret. I denne opgave sættes $m = 100$, da vi gerne vil arbejde med 100 punkter til ekstern test. Funktionen $g(x_i)$ består som sagt af Splines, mens y_i repræsenterer funktionsværdierne med støj.
Dette skyldes at vi ønsker at finde g, der minimerer summen af kvadraterne af forskellene mellem $g\ og\ y$, altså, $\sum_{i=1}^{m}(g(x_i) - y_i)^2$. Når vi undersøger residualer ved den måde, ser vi også på residualerne ved $\big||A\boldsymbol{c} - y|\big|$. Så jo mindre forskellen der er mellem $g(x)\ og\ y$, desto bedre er løsningsvektoren **c**.

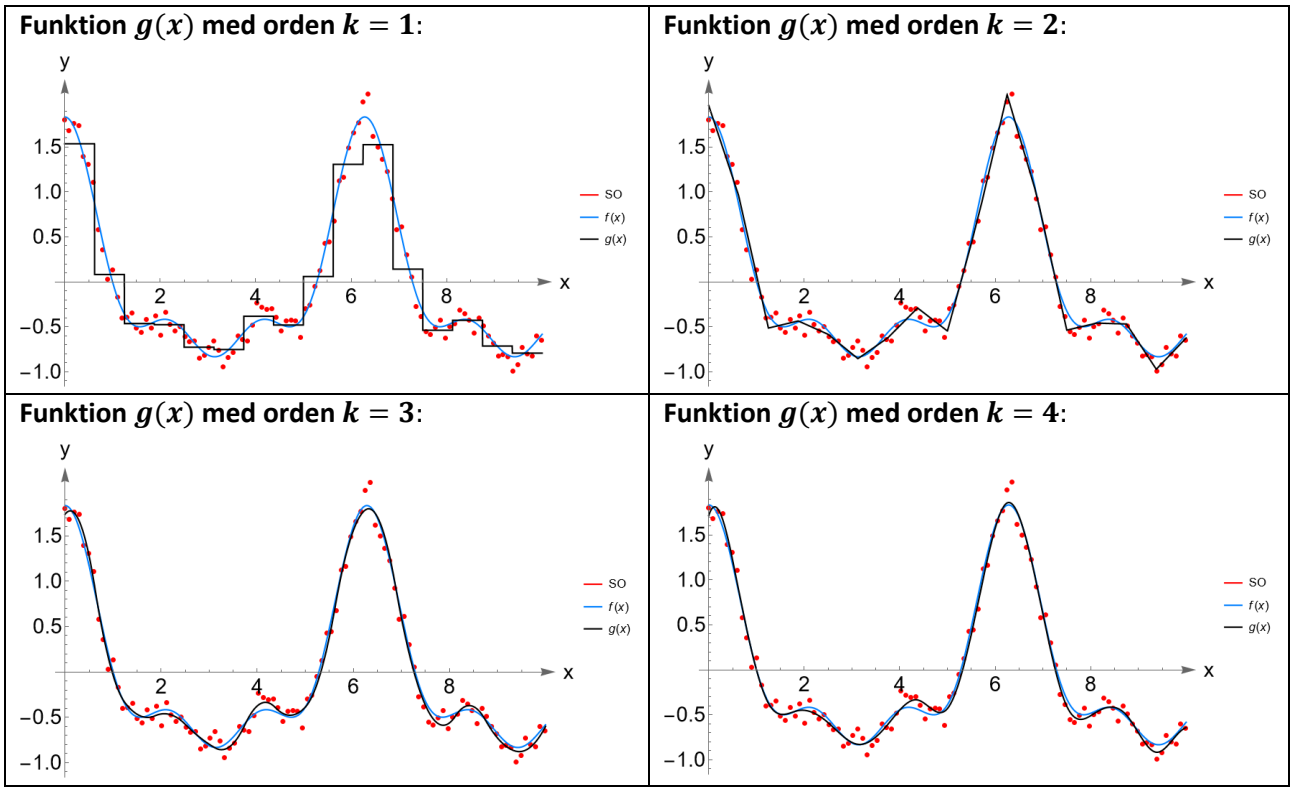
I denne opgave bliver alle beregninger på nær residualer udført i C++-program, og resultaterne indlæses derefter i Mathematica, hvor der foretages udregningen af residualerne.

6.3 Ordenen k's betydning:

Her foretages en ekstern test, hvor der kigges på hvilken betydning orden k har for funktionen $g(x)$.

6.3.1 Ordenen k's betydning grafisk set:

Først vises grafisk, hvordan funktionen $g(x)$ ser ud med orden $k = 1, k = 2, k = 3$ og $k = 4$ og antal interval l sættes lig med 16, $l = 16$.



Dermed kan man se, at grafen bliver lidt mere glattere, jo højere orden k, man bruger. Dettes skyldes at jo højere orden man bruger, desto højere grad polynomier Splines bruger.

6.3.2 Ordenen k's betydning i forhold til residualerne:

Forneden ses der en tabel over de forskelle orden k, hvor antal interval l holdes fast på $l = 16$. Tabellen med fulde decimaler kan findes i Bilag 11.

Ordenen k:	$g(x) - f(x)$	$\|\|r\|\| = g(x) - y$	$\|\|Ac - y\|\|$
1	2.122433	2.239305	2.239305
2	0.709207	0.991094	0.991094
3	0.488344	0.934547	0.934546
4	0.469738	0.891121	0.891121
5	0.465320	0.898598	0.898598
6	0.482184	0.885043	0.885044
8	0.484045	0.885116	0.885116
10	0.488350	0.882964	0.882963
15	0.571193	0.831830	0.831819
20	0.669169	0.755258	0.754996

40	0.795891	0.620290	0.620288

Hvis man kigger på den første regression, residualerne mellem $g(x)\ og\ f(x)$, kan man se det er ved orden $k = 5$, opnår man den bedste regression, da det er netop der, hvor forskellen mellem $g(x)\ og\ f(x)$ er mindst.

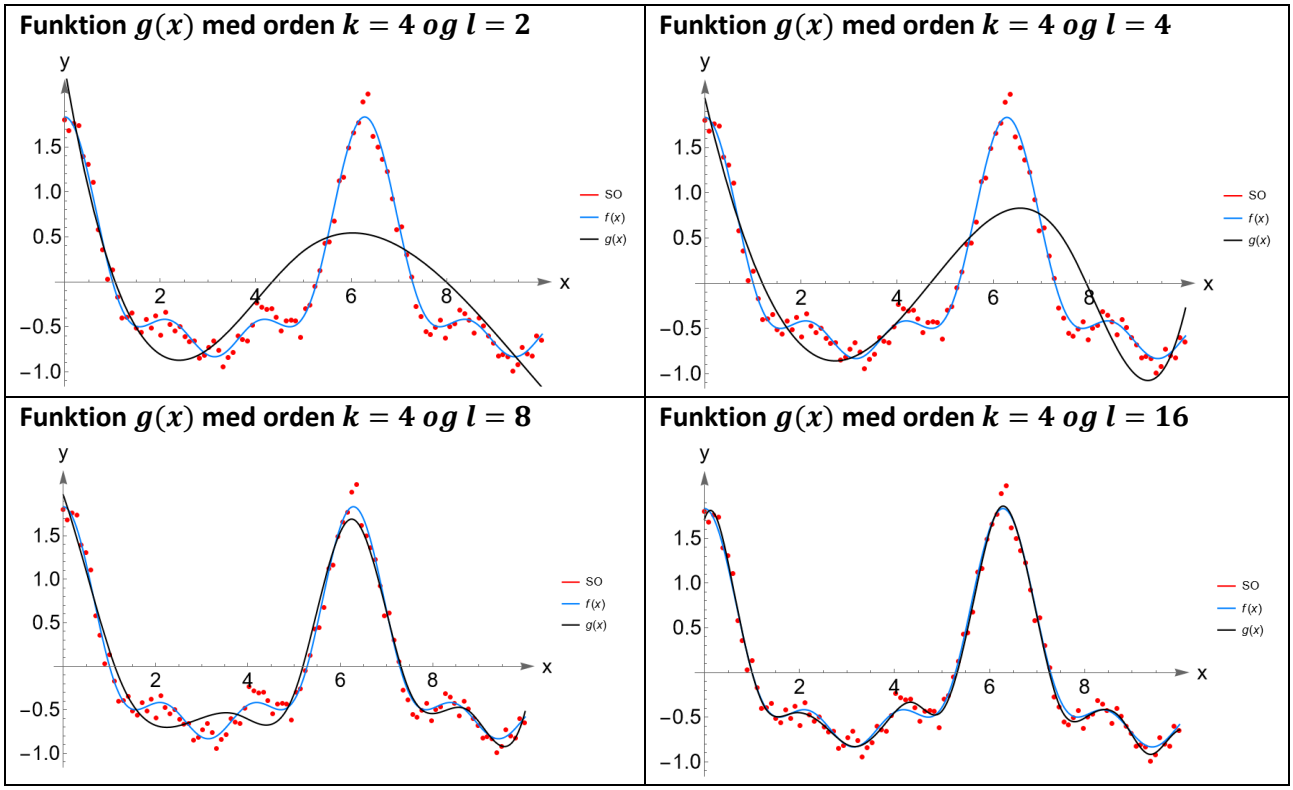
Hvis man kigger på den anden regressionudregning, ses der at man kan finde den bedste løsningsvektor **c**, når højere orden k er, da residualerne mellem $g(x)\ og\ y$ bliver mindre og mindre, jo højere orden k er.

6.4 ANTAL INTERVAL l'S BETYDNING:

Her foretages en test, der kigger på hvilken betydning antal intervaller l, har for funktionen $g(x)$.

6.4.1 Antal interval l's betydning grafisk set:

Først vises grafisk, hvordan funktionen $g(x)$ ser ud med intervaller på $l = 2, l = 4\ ,\ l = 8\ og\ l = 16$ og orden k sættes lig med 4, $k = 4$.



Som det kan ses, jo flere interval, desto flere knudepunkter og dermed tilnærmer $g(x)$ sig til den underliggende funktion $f(x)$.

6.4.2 Antal interval l's betydning i forhold til residualerne:

Forneden ses der en tabel over de forskelle antal af intervaller$l$, hvor orden k holdes fast på $k = 4$. Tabellen med fulde decimaler kan findes i Bilag 12.

Antal interval l:	$g(x) - f(x)$	$\|\|r\|\| = g(x) - y$	$\|\|Ac - y\|\|$
1	5.539802	5.647104	5.647103
2	4.774624	4.878203	4.878204
4	4.016675	4.186133	4.186132
6	2.667791	2.972651	2.972649
8	1.340285	1.777253	1.777253
10	1.035639	1.498423	1.498424
12	0.563531	0.974576	0.974576
13	0.501725	0.950786	0.950787
14	0.489258	0.906263	0.906262
15	0.440478	0.937160	0.937161
16	0.469737	0.891121	0.891121
17	0.480285	0.894205	0.480285
20	0.485059	0.886524	0.886524
22	0.569968	0.832535	0.832534
26	0.604076	0.809625	0.809625
32	0.651226	0.770571	0.770572
40	0.701534	0.725288	0.725288

Ifølge oversigten kan man se at til den første regression udregning, esidualerne mellem $g(x)\ og\ f(x)$, kan man se at man opnår den bedste regression på 0.440478 ved 15 antal intervaller l, når orden k er 4. Det vil sige at hvis man gerne vil lave en B-Splines af orden 4, skal man bruge 15 antal intervaller for at kunne tilnærme sig den bagvedliggende funktion, $f(x)$.

Hvis man kigger på den anden regression udregning, ses der at man kan finde den bedste løsningsvektor **c**, når der bruges flere intervaller l, da residualerne mellem $g(x)\ og\ y$ bliver mindre og mindre, jo flere intervaller der bruges.

6.5 B-SPLINESREGRESSION SAMMENLIGNES MED ANDRE TYPER REGRESSION

Her ønsker vi at sammenligne Bspline-regression med andre typer regression, for at undersøge hvilken regressionsmetode, der bedst tilpasser regressionssplines til de støjfylde observationer af funktionen $f(x)$.

6.5.1 Polynomiel regression

I denne test undersøges forskellen mellem Bspline-regression og polynomiel regression.

Der oprettes en oversigt over, hvor meget regression de forskellige grader skaber, når de hhv. sammenlignes med funktionen $f(x)$ og y-værdiern. Testning udføres i Mathematica, hvor x og y er punkterne, der opnås fra de støjfyldte observationer. Funktionen $ps(x)$, der er en funktion for en polynomiet, bestemmes i Mathematica via:

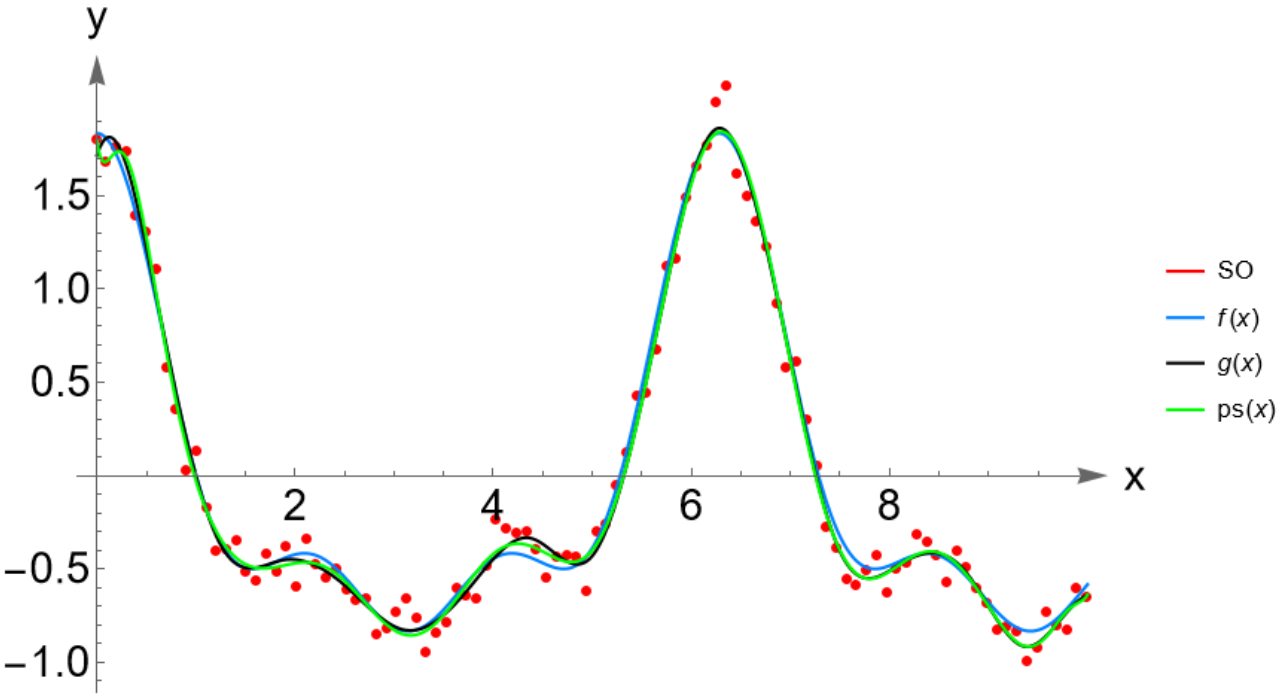
$$Fit[\text{Transpose}[\{\text{xv}, \text{yv}\}], \text{Table}[x^i, \{i, 0, \text{grad}\}], x]$$

Derefter beregnes residualerne for polynomielfunktionen i forhold funktionen $f(x)$ og y-værdierne.

Grad x^i	$ps(x) - f(x) = \sqrt{\sum_{i=1}^{100}(ps(x_i) - f(x_i))^2}$	$\|\|r\|\| = \sqrt{\sum_{i=1}^{100}(ps(x_i) - y_i)^2}$
4	4.67559	4.786243
8	3.099539	3.380963
12	1.607232	1.932783
16	0.490521	1.008516
18	0.477079	0.897144
19	0.480846	0.894276
20	0.475283	0.894185
21	0.475950	0.893546
22	0.477139	0.891908
24	0.480846	0.887526
30	0.484324	0.885211
⋮	⋮	⋮
100	0.545903	0.848512

Når der kigges på residualerne mellem $ps(x)\ og\ f(x)$ kan man se, at det er ved grad 20, hvor regression er mindst, nemlig ved 0.47583. Hvis man kigger på forholdet mellem $ps(x)$ og y-værdierne, kan man se at jo højere grad polynomierne er, desto bedre bliver regressionen.

Forneden kan man se grafen for polynomiel Splines $(ps(x))$ sammen med B-Splines $(g(x), med\ k = 4, l = 16)$ og den underliggende funktion $(f(x))$ samt de støjfyldte observationer (SO):



6.5.2 Kubiske Splines

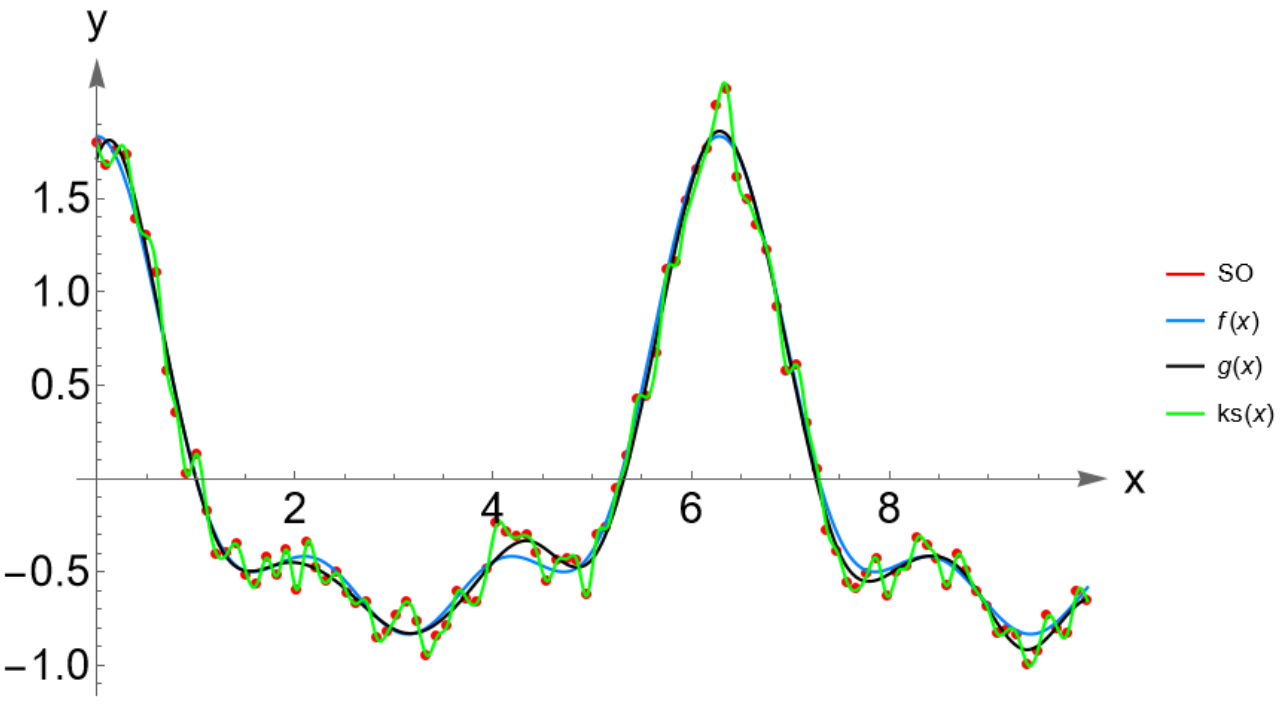
En anden form for regressionssplines kan være Kubiske splines, der er et interpolerende tredjegradspolynomium, hvor den kan tegne et stykvist defineret polynomium mellem hvert eneste punkt. Dermed kan man forvente at regressionen ved $ks(x) - f(x)$ er meget tæt på at være 0, hvor funktionen $ks(x)$ er en funktion for kubiske splines

Den kubiske parameterfremstilling $pi(x)$ bestemmes i mit C++-program, hvor residualerne udregnes i Mathematica. Når vi udregner normen af residualerne mellem $ks(x)\ og\ f(x)$ og $ks(x)\ og\ y$

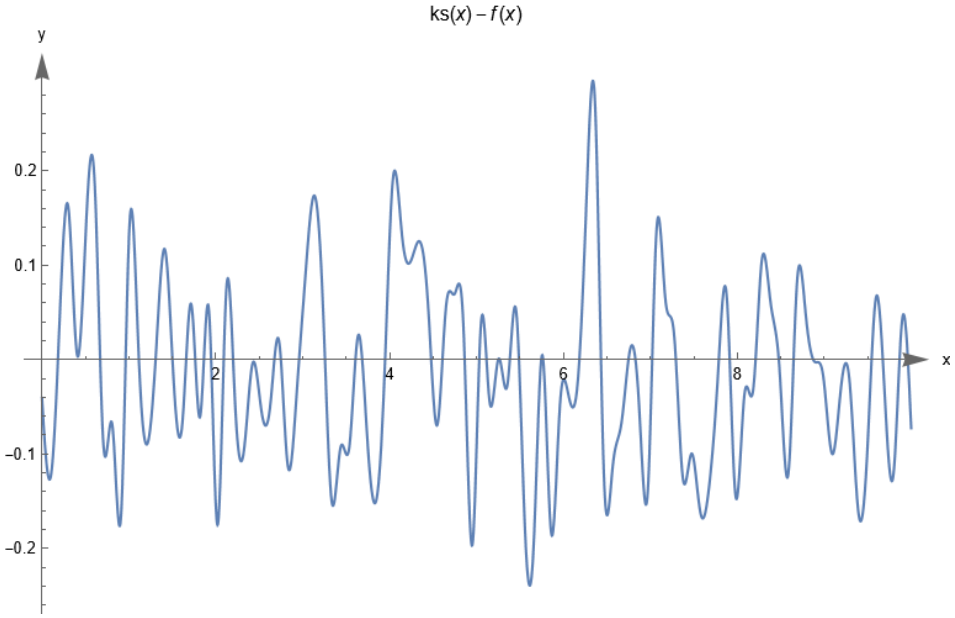
$ks(x) - f(x) = \sqrt{\sum_{i=1}^{100} \left(ks(x_i) - f(x_i)\right)^2}$	2.05969×10^{-10}
$\|\|r\|\| = \sqrt{\sum_{i=1}^{100} (ks(x_i) - y_i)^2}$	1.00906

Som forventeligt har kubiske interpolerende splines en meget lavere residualer end ved B-Splines, når der kigges på residualerne mellem $ks(x)\ og\ f(x)$. Dette skyldes, at de kubiske splines skal interpolere et hvert knudepunkt, og dermed vil residualerne være meget tæt på 0. Dermed kan man sige at de kubiske splines interpolerer ved approksimationen næsten nøjagtig alle knudepunkter.

Forneden kan man se grafen for kubiske splines $(ks(x))$ sammen med B-Splines $(g(x), med\ k = 4, l = 16)$ og den underliggende funktion $(f(x))$ samt de støjfyldte observationer (SO). Her kan man se, at kubiske splines kører igennem alle punkterne.



Når man kigger på afvigelsen mellem $ks(x)\ og\ f(x)$ kan se at den største afvigelse ligger ved $x = 6{,}28934$. Men når man kigger på grafen foroven, kan man se at der ligger et punkt længst væk fra funktionen $f(x)$ ved $x = 6.28$.



6.5.3 Konklusion med de forskellige regressionsplines:

I undersøgelsen af B-Splines regression kom man frem til at man kan opnå den bedste regression på afstanden mellem $g(x) - f(x)$ på 0.440478 med $k = 4, l = 15$.

Fra polynomiel regressionsundersøgelsen kom man frem til, at man kan opnå den bedste regression ved grad 20, hvor regressionen er på 0.475285. Hvor Hos kubiske splines havde man en regression på 1.00906, når der kigges på $ks(x) - f(x)$.

Dermed kan man konkludere at B-Splines er den bedste løsning, hvis man bedst mulig gerne vil kunne tilnærme sig den bagvedliggende funktion, $f(x)$, da B-Splines har mindst residualer i forhold til de to andre splines. Dette skyldes at B-Splines kan skabes en 'glat' splines, der ikke nødvendigvis skal gå igennem alle punkterne. Med andre ord interpolerer B-Splines ikke. Og B-Splines har også en 'lokal' kontrol, og dermed kan skabe en fin kurve uden at blive påvirket af andre knudepunkter.

Hvis man ønsker at skabe en kurve, der bedst tilpasser punkterne, er kubiske splines den oplagte valg, da den havde den mindste residualer i forhold til de andre. Kubiske splines har en regression på næsten 0, når der kigges på $ks(x) - y$, hvor derimod polynomielregression havde en på 0.848512 og B-Splines havde en regression på 0.620290 med $k = 40, l = 16$. Dette skyldes at kubiske splines bruges til at udføre en interpolering.

I Bilag 13 kan man se en graf med alle forskellige regressionsplines.

6.6 HOUSEHOLDERS ALGORITME SAMMENLIGNES MED ANDRE METODER.

Her sammenlignes Householders algoritme med andre algoritme til mindste kvadraters metode, for at undersøge om, hvordan Householders algoritme står i forhold til de andre algoritmer til mindste kvadraters metode.

Der findes forskellige typer algoritmer, der kan minimere summen af kvadraterne på residualerne. Der findes blandt andet normalligningerne, Gram-Schmidt processen og modificeret Gram-Schmidt.

Fra den obligatoriske opgave blev der regnet frem til at modificeret Gram-Schmidt er den mest pålidelige algoritme. Derfor vil vi teste algoritmen modificeret Gram-Schmidt med Householders algoritme, for at undersøge, hvem af dem der er den mest pålidelige algoritme til mindste kvadraters metode.

Måden at undersøge på, hvornår forskellen mellem de to algoritmer afviger, er at køre programmet med matricer, som er specielt vanskelige at arbejde med, nemlig Hilbertmatricer. Men da opgaven netop beskæftiger sig med ligningssystemer, hvor $m > n$, testes begge algoritmer med Hilbertmatricer, hvor $m > n$. Helt præcis vil der bruges en $20\ x\ 8$ Hilbertmatrix til testning.

Vi vil teste med forskellige typer konditionstal, for jo højere en konditionstal er, desto sværere er det at løse ligningen. For at fastholde matricen $20\ x\ 8$, ønsker vi at øge konditionstal ved at øge tallet i diagonalen i matricen. Diagonalen ganges med t, og dermed kan vi styre konditionstallet. På Mathematica skrives der:

$$H = \text{HilbertMatrix}[20][[\text{All},;;8]] + t * \text{SparseArray}[\text{IdentityMatrix}[20], \{20,8\}]$$

De to algoritme skal løse Hilbertmatrix H med en vektor **b**, der indeholder værdier fra søjle nr. 9 fra en $20x9$ Hilbertmatrix, og dermed vil det være en svær ligning at arbejde med.

Dermed vil totalmatricen være:

$$[H\ \boldsymbol{x} \quad \boldsymbol{b}]$$

For at skabe en overskuelige oversigt, beregnes der gennemsnit af hvor mange korrekte decimaler, der udregnes for hvert konditionstal.

Beregningen baseres på den rigtige løsning, der bliver udregnet i Mathematica ved hjælp af *'LeastSquares'.* Denne sammenlignes med løsningerne udregnet af Householders metode og Modificeret Gram-Schmidt-metode, i mit C++-program.

Det vil sige, at der tages den rigtige løsning fra Mathematica ind i C++-programmet, hvor hvert tal i løsningsvektor **x** tjekkes for, hvor mange decimaler de forskellige metoder har udregnet korrekt. Derefter tages der et gennemsnit af summen af de korrekte decimaler. Her findes der 8 tal i løsningsvektor **x**, da vi har 8 søjler i Hilbertmatricen og dermed kan gennemsnit bestemmes ved at dividere summen af alle korrekte tal med 8 inden for en Hilbertmatrix. Se Bilag 14 for at se beregningen.

t:	Konditionstal:	Mgs Korrekte decimaler i gennemsnit	Householder Korrekte decimaler i gennemsnit
$\mathbf{1}$	2,75174	15,125	15
$\mathbf{1 * 10^{-1}}$	18,79223	14,5	15
$\mathbf{1 * 10^{-2}}$	179.352	12,875	14,875
$\mathbf{1 * 10^{-3}}$	1784.98	11	14,125
$\mathbf{1 * 10^{-4}}$	17841.2	9.5	12,75
$\mathbf{1 * 10^{-5}}$	178401	7.125	12.25
$\mathbf{1 * 10^{-6}}$	$1.78369 * 10^6$	5.25	11.25
$\mathbf{1 * 10^{-7}}$	$1.77829 * 10^7$	4.125	10.125
$\mathbf{1 * 10^{-8}}$	$1.58908 * 10^8$	3,5	9
$\mathbf{1 * 10^{-9}}$	$3.71631 * 10^8$	0.875	9,875
$\mathbf{1 * 10^{-10}}$	$3.83382 * 10^8$	1.125	9

Det ses af oversigten at Modificeret Gram-Schmidt-metoden begynder at miste sin præcision, jo højere konditionstallet er. Men hos householders algoritme mister den sin præcision langsomt ved et højere konditionstal. Dermed kan der konkluderes at Householders algoritme er den mest pålidelige metode til at løse kvadraters metode.

7 KONKLUSION

Det kan konkludere, at når man bruger De Boors algoritme til at opstille en totalmatrix for B-Splines baseret på støjfyldt (x, y)-data og derefter anvender Householder's algoritme til mindste kvadraters metode, kan man opnå en fin kurve, der tilnærmer sig den bagvedliggende funktion, $f(x)$.

Programmet blev testet både internt og eksternt. Ud fra de interne tests kan det ses, at der ikke har været nogen fejl, da resultaterne ikke afviger meget fra CAS-værktøjet Mathematica. Derfor kan det konkluderes at koden i programmet er skrevet og fungerer korrekt.
I forhold den eksterne test, kan det konkluderes at øgningen af intervalstørrelsen l og orden k kun giver en bedre tilnærmelse op til at vist punkt, hvorefter præcisionen falder igen. Heraf kan det også konkluderes, at hvis man ønsker at tilpasse kurven til disse støjfyldte data, vil det kræve flere intervaller l og en højere k.

B-Splines blev også testes eksternt i forhold til Splinesregression med polynomiel-Splines og Kubiske Splines. Her kan man konkludere at B-Splines er den bedste løsning, hvis man bedst mulig gerne vil kunne tilnærme sig den bagvedliggende funktion, $f(x)$, da B-Splines har mindst residualer i forhold til de to andre splines.

Desuden bliver Householder-metoden også testet sammen med en anden metode, Modificeret Gram Schmidt, for at undersøge, hvilken af de to metoder, der klarer sig bedst i forhold til at løse mindste kvadraters metode. I denne sammenhæng blev de to testet med Hilbertmatricer, der har høje konditionstal. Her kan det konkluderes at Householders algoritme klarer sig bedre end Modificeret Gram Schmidt-metoden, når der kommer til beregning af matricer med meget høje konditionstal. Dermed kan der også konkluderes, som set i den obligatoriske opgave, at Householder er den mest pålidelige metode, herefter kommer hhv. Modificeret Gram Schmidt-metode, Gram-Schmidt metoden og normalligningerne som de mindst pålidelige metoder i den nævnte rækkefølge.

Til sidst kan det konkluderes, at der er udviklet et velfungerende C++-program, der kan tilpasse en regressionspline til de indlæste data ved hjælp af De Boors algoritme og Householder-transformation til at levere pålidelige resultater.

8 BILAG

8.1 KILDEKODE

```
#include <iostream> //cout + cin
#include <vector> //vector
#include <fstream> //fileoutput
#include <iomanip> //setw
#include <cmath> //pow

using namespace std;

using Vektor = vector<double>;
using Matrix = vector<Vektor>; // vector<vector<double>>

//Funktionerdeklarationer

//Erklærer de forskellige funktioner til hhv. vektorer og
matricer.
void velkomsttekst();
Vektor hent_vektor(string fn);
Matrix dan_matrix(size_t m, size_t n);
void udskriv_vektor(const Vektor& v);
void udskriv_matrix(const Matrix& A);
pair<size_t, size_t> size(const Matrix& A);
pair<double, double> minmax(const Vektor& x);

//Erklærer funktioner til De Boors algoritme
Vektor eval_spline(const Vektor& t, int k, int v, double x);
int vknude(const Vektor& t, double x);
Vektor dan_knudepunkter(double tmin, double tmax, int l, int k);
Matrix spline_totalmatrix(const Vektor& t, int k, const Vektor&
x, const Vektor& y);

//Erklærer funktioner til Householders algoritme
Vektor house(const Matrix& T, int j);
void anvend_house(Matrix& T, const Vektor& v, int j);
void houseQR(Matrix& T);
Vektor bsub(const Matrix& T);
```

```
int main() {
        velkomsttekst();
        Vektor xv = hent_vektor("x.txt");
        Vektor yv = hent_vektor("y.txt");

        auto[xmin,xmax] = minmax(xv);
        cout << "xmin: " << xmin << endl;
        cout << "xmax: " << xmax << endl;

        int k, l;
        cout << "Indtast k: ", cin >> k;
        cout << "Indtast l: ", cin >> l;
        Vektor t = dan_knudepunkter(xmin, xmax , l, k);
        cout << "Knudepunkter t: " << endl;
        udskriv_vektor(t);
        cout << endl;

        cout << "Koefficient Matrix: " << endl;
        Matrix TM = spline_totalmatrix(t, k, xv, yv);
        udskriv_matrix(TM);

        houseQR(TM);
        Vektor c = bsub(TM);
        cout << "House: " << endl;
        udskriv_matrix(TM);
        cout << "\n" << endl;

        cout << "Vektor c: " << endl;
        udskriv_vektor(c);

        return 0;
}

void velkomsttekst(){
        cout << "Velkommen til mit projekt\n" << endl;

        cout << "Formålet for programmet er at minimere normen
af vektoren Ac-y.\n"
                        "Et sådant minimeringsproblem kaldes
for mindste kvadraters metode" << endl;
```

```
        cout << "Opgaven beskæftiger sig med De Boors algoritme
til at danne en totalmatrix\n"
                        "og Householders algoritme til at løse
problemet.\n" << endl;
}

Vektor hent_vektor(string fn) {
        Vektor v;
        ifstream fil(fn);

        if (!fil.is_open()) {
                cout << "Kunne ikke åbne filen " << fn <<
".\n";
                return v;
        }

        for (double e; fil >> e;) {
                v.push_back(e);
        }

        return v;
}

Matrix dan_matrix(size_t m, size_t n) {
        return Matrix(m, Vektor(n));
}

void udskriv_vektor(const Vektor& v) {
        cout << "{";
        if (!v.empty()) {
                cout << v[0];
                for(size_t k = 1; k < v.size(); k++) {
                        cout << ", " << v[k];
                }
        }
        cout << "}\n";
}

void udskriv_matrix(const Matrix& A) {
        for (const auto& v: A) {
                for (const auto& e: v) {
                        cout << setw(15) << e;
```

```
                }
                cout << endl;
        }
}

//Returnerer Matrix A's størrelsen (mxn)
pair<size_t, size_t> size(const Matrix& A) {
        size_t n=0,m=A.size();
        if (m>0)
                n=A[0].size();
        return {m,n};
}

//Returnerer den mindste og største værdi fra vektor x.
pair<double, double> minmax(const Vektor& x){
        double xmin = x[0];
        double xmax = x[0];

        for (size_t i = 0; i < x.size(); i++){
                if (x[i] < xmin){
                        xmin = x[i];
                }
                else if (x[i] > xmax){
                        xmax = x[i];
                }
        }
        return {xmin,xmax};
}


//Evaluer alle ikke-nul B-splines af orden k (grad k-1) med
knudepunkter i t i x.
Vektor eval_spline(const Vektor& t, int k, int v, double x){
        Vektor b(k), dL(k-1), dR(k-1);
        double term, saved;

        for (int s = 0; s <= k-2; s++){
                dL[s] = x - t[v-s];
                dR[s] = t[v+1+s]-x;
        }
        b[0]=1;
```

```
        for (int j = 1; j <= k-1; j++){
                saved = 0;

                for (int r = 0; r <= j-1; r++){
                        term = b[r]/(dL[j-1-r]+dR[r]);
                        b[r] = saved + (dR[r]*term);
                        saved = dL[j-1-r]*term;
                }
                b[j]=saved;
        }
        return b;
}

//Danner en vektor med knudepunkter
Vektor dan_knudepunkter(double tmin, double tmax, int l, int k){
        //Definerer størrelsen til vektor t:
        int n = k*2+l-1;
        Vektor t(n);

        //De første k skal være lig med det mindste element i x
(tmin)
        //De sidste k skal være lig med det maksimale element i
x (tmax)
        for (int i = 0; i < k; i++){
                t[i] = tmin;
                t[n-i-1] = tmax;
        }

        //Danner ækviditant.
        double dt =(tmax - tmin)/l;
        double x = tmin;

        //Danner de midterste punkter, der er ækvidistant
fordelt:
        for (int j = k-1; j < n-k; j++){
                t[j] = x;
                x += dt;
        }

        //Vektor t returneres
        return t;
}
```

```
int vknude(const Vektor& t, double x){
        int n = t.size();

        //Sikre os, at x ikke er "udenfor" af de område, vi
ønsker at beregne.
        if (x < t[0]){
                return 0;
        }

        //Der findes den v, der er større end x
        //og dermed returnes v-1, altså række før.
        for (int v = 0; v < n; v++){
                if (t[v] > x){
                        return v-1;
                }
        }
        //Returner det sidste indeks i vektor t.
        return n-1;
}

//Koefficientmatricen dannes ved hjælp af De Boors algoritme
Matrix spline_totalmatrix(const Vektor& t, int k, const Vektor&
x, const Vektor& y){
        size_t m = x.size();
        int n = t.size()-k; //da vi ikke kender l (t.size() - k)
        Matrix TM = dan_matrix(m,n+1); //da vi også skal have
konstantsøjlen.

        for (size_t i = 0; i < m; i++){
                int v = vknude(t, x[i]);
                if (v >= n){
                        v = n-1;
                }
                Vektor b = eval_spline(t, k, v, x[i]);
                for (int j = 0; j < k; j++){ //Der tilføjes
data på hver søjler i i-række indtil konstantsøjlen.
                        TM[i][j+v-(k-1)] = b[j];
                }
                TM[i][n] = y[i]; //Danner konstantsøjlen
        }
```

```
        return TM;
}

//Dan v for en Householder-transformation der kan skabe nuller i
søjle j af T
Vektor house(const Matrix& T, int j){
        double m = T.size(); //m = antal rækker
        double sigma = 0;
        //Definerer vektor v med størrelsen (m-j)
        Vektor v(m-j);

        for (int i = 1; i < m-j; i++){
                sigma = sigma + pow(T[j+i][j],2);
                v[i] = T[j+i][j];
        }

        double my = sqrt(pow(T[j][j],2) + sigma);

        //v[] startes i 0 grundet nulindekseret.
        if(T[j][j] <= 0){
                v[0] = T[j][j]-my;
        }
        else{
                v[0] = (-sigma) / (T[j][j]+my);
        }

        return v;
}

// Housholder-transformation anvendes på Ts undermatrix startende
i række og søjle j.
void anvend_house(Matrix& T, const Vektor& v, int j){
        auto[m,n] = size(T);
        n--; //konstantsøjle tages ikke med:

        for (size_t k = 0; k <= n-j; k++){
                double pi = 0;
                double gamma = 0;

                for (size_t i = 0; i < m-j; i++){
                        pi = pi + (v[i] * T[j+i][j+k]);
                        gamma = gamma + (v[i]*v[i]);
```

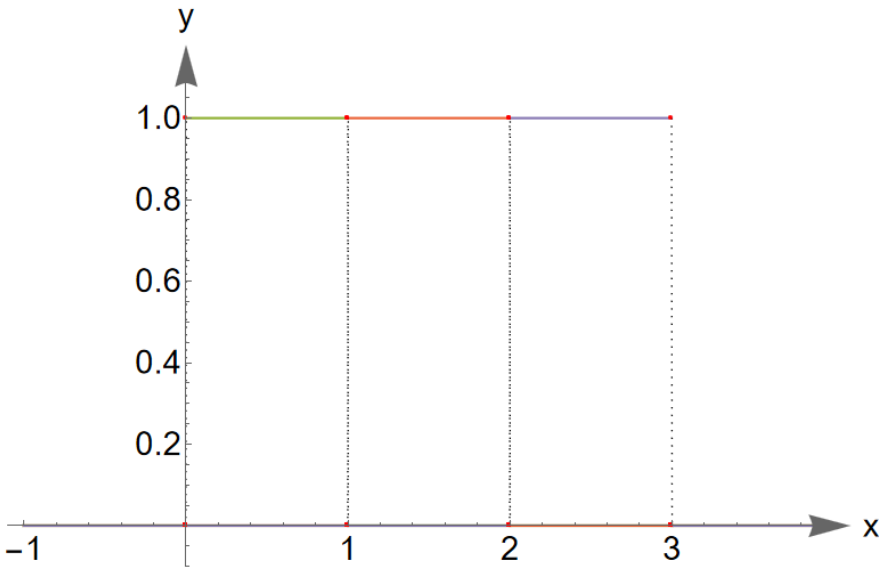
```
                    }

                    double beta = 2.0/gamma;

                    for (size_t i = 0; i < m-j; i++){
                            T[j+i][j+k] -= (beta*pi*v[i]);
                    }
            }
}

//Householder QR for totalmatrix. R bliver lagt i øvre trekant af
T.
void houseQR(Matrix& T){
            auto[m,n] = size(T);
            n--;

            for (size_t j = 0; j < n; j++){
                    Vektor v = house(T,j);
                    anvend_house(T,v,j);
            }
}

 //Løser Rc = v med baglæns substitution under antagelse af at R
er en øvre trekantsmatrix.
Vektor bsub(const Matrix& T){
            int n = T[0].size();
            int m = n-1;

            Vektor c(m);

            c[m-1] = T[m-1][m]/T[m-1][m-1];
            for (int i = m-2; i >= 0; i--){
                    double sum = 0;
                    for (int j = i+1; j < m; j++){
                            sum += T[i][j]*c[j];
                    }
                    c[i] = (T[i][m]-sum)/T[i][i];
            }
            return c;
}
```

8.2 BILAG NR:

8.2.1 Bilag 1

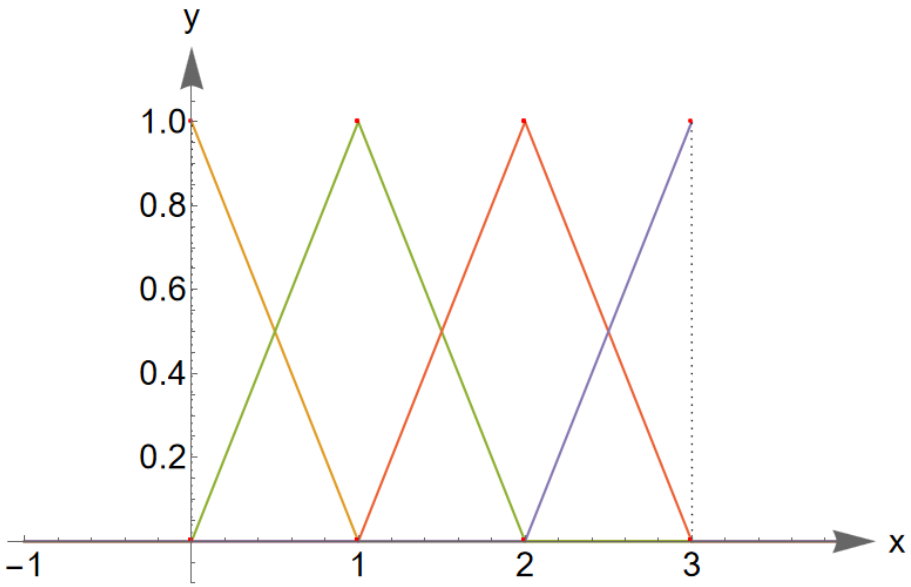
Splines af orden 1 med t knudepunkter

```
Plot[Evaluate@Table[BSplineBasis[{0, t}, k, x], {k, 1, 5}], {x, -1, 4}, Evaluate@st]
```



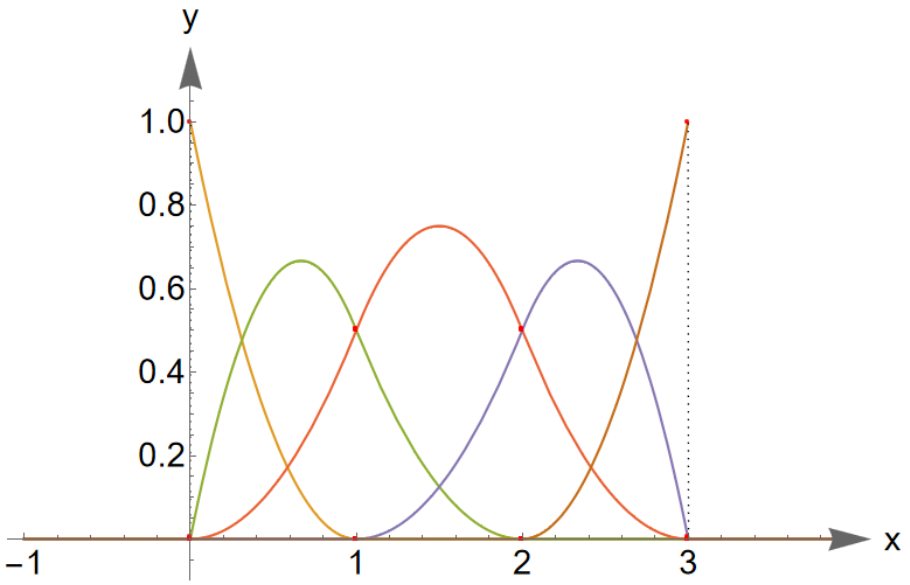
Splines af orden 2 med t knudepunkter

```
Plot[Evaluate@Table[BSplineBasis[{1, t}, k, x], {k, 1, 5}], {x, -1, 4}, Evaluate@st]
```



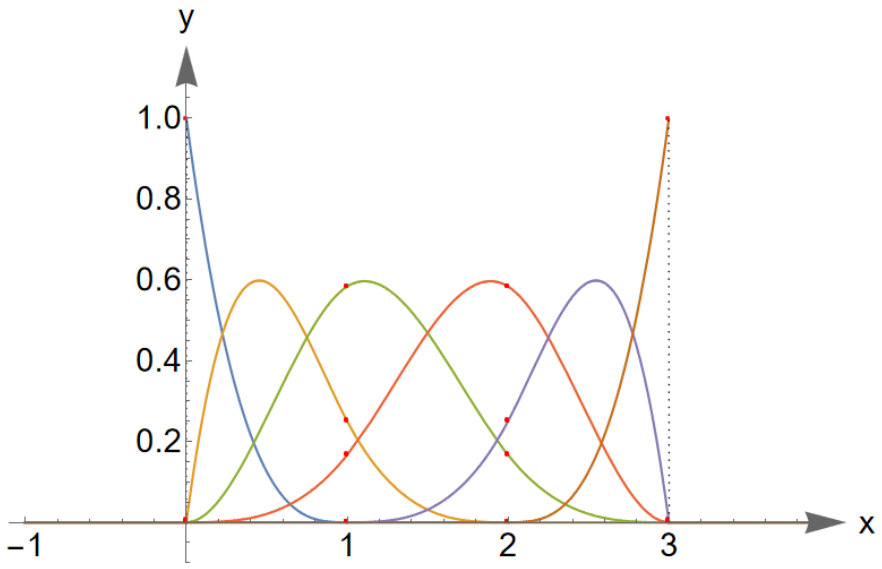
Splines af orden 3 med t knudepunkter

```
Plot[Evaluate@Table[BSplineBasis[{2, t}, k, x], {k, 0, 5}], {x, -1, 4}, Evaluate@st]
```



B-Splines af orden 4 med t knudepunkter

```
Plot[Evaluate@Table[BSplineBasis[{3, t}, k, x], {k, 0, 5}], {x, -1, 4}, Evaluate@st]
```



8.2.2 Bilag 2

Testning af *'minmax()'*

```
Testning af minmax:
Vektor x:
{-1, 3, 1, 50, -4, 3, 46}
xmin: -4 , xmax: 50
```

8.2.3 Bilag 3

```
Testning af dan_knudepunkter

Vektor x:
{1.2, 0, 1.2, 2.1, 3, 2.9}
xmin: 0 , xmax: 3

Indtast k: 4
Indtast l: 3
Knudepunkter t:
{0, 0, 0, 0, 1, 2, 3, 3, 3, 3}
```

```
cout << "Testning af dan_knudepunkter" << endl;
Vektor x = {1.2, 0, 1.2, 2.1, 3, 2.9};
cout << "\nVektor x: " << endl;
udskriv_vektor(x);
auto[xmin, xmax] = minmax(x);
cout << "xmin: " << xmin << " , xmax: " << xmax << "\n" << endl;

int k, l;
cout << "Indtast k: ", cin >> k;
cout << "Indtast l: ", cin >> l;

Vektor t = dan_knudepunkter(xmin, xmax, l, k);
cout << "Knudepunkter t: " << endl;
udskriv_vektor(t);
```

8.2.4 Bilag 4

Testning af 'vknude()'

```
Testning af vknude:
Vektor x:
{0, 0.5, 1.2, 2.1, 2.6, 3}
xmin: 0 , xmax: 3

Indtast k: 4
Indtast l: 3
Knudepunkter t:
{0, 0, 0, 0, 1, 2, 3, 3, 3, 3}

x: 0
v: 3 , t[v] --> t[3]: 0

x: 0.5
v: 3 , t[v] --> t[3]: 0

x: 1.2
v: 4 , t[v] --> t[4]: 1

x: 2.1
v: 5 , t[v] --> t[5]: 2

x: 2.6
v: 5 , t[v] --> t[5]: 2

x: 3
v: 9 , t[v] --> t[9]: 3
```

8.2.5 Bilag 5

Testning af *'eval_spline()'*. Udklip fra mit C++-program

```
Testning af eval_spline
Vektor x:
{0, 0.5, 1.2, 2.1, 2.6, 3}
xmin: 0 , xmax: 3

Indtast k: 4
Indtast l: 3
Knudepunkter t:
{0, 0, 0, 0, 1, 2, 3, 3, 3, 3}

x: 0   b: {1, 0, 0, 0}

x: 0.5   b: {0.125, 0.59375, 0.260417, 0.0208333}

x: 1.2   b: {0.128, 0.588, 0.282, 0.002}

x: 2.1   b: {0.1215, 0.54675, 0.33075, 0.001}

x: 2.6   b: {0.0106667, 0.181333, 0.592, 0.216}

x: 3   b: {0, 0, 0, 1}
```

Et udklip fra Mathematica.

```
k = 4; l = 3;

n = l + k - 1;

Table[BSplineBasis[{k - 1, {0, 0, 0, 0, 1, 2, 3, 3, 3, 3}}, j, 0], {j, 0, n - 1}]
```

{1, 0, 0, 0, 0, 0}

```
Table[BSplineBasis[{k - 1, {0, 0, 0, 0, 1, 2, 3, 3, 3, 3}}, j, 0.5], {j, 0, n - 1}]
```

{0.125, 0.59375, 0.260417, 0.0208333, 0, 0}

```
Table[BSplineBasis[{k - 1, {0, 0, 0, 0, 1, 2, 3, 3, 3, 3}}, j, 1.2], {j, 0, n - 1}]
```

{0, 0.128, 0.588, 0.282, 0.002, 0}

```
Table[BSplineBasis[{k - 1, {0, 0, 0, 0, 1, 2, 3, 3, 3, 3}}, j, 2.1], {j, 0, n - 1}]
```

{0, 0, 0.1215, 0.54675, 0.33075, 0.001}

```
Table[BSplineBasis[{k - 1, {0, 0, 0, 0, 1, 2, 3, 3, 3, 3}}, j, 2.6], {j, 0, n - 1}]
```

{0, 0, 0.0106667, 0.181333, 0.592, 0.216}

```
Table[BSplineBasis[{k - 1, {0, 0, 0, 0, 1, 2, 3, 3, 3, 3}}, j, 3], {j, 0, n - 1}]
```

{0, 0, 0, 0, 0, 1}

8.2.6 Bilag 6

Testning af *'spline_totalmatrix'*

```
Testning af spline_totalmatrix
Vektor x:
{0, 0.5, 1.2, 2.1, 2.6, 3}
Vektor y:
{1, 1, 1, 1, 1, 1}
xmin: 0 , xmax: 3

Indtast k: 4
Indtast l: 3
Knudepunkter t:
{0, 0, 0, 0, 1, 2, 3, 3, 3, 3}

Totalmatrix for Splines
         1          0          0          0          0          0          1
     0.125    0.59375   0.260417  0.0208333          0          0          1
         0      0.128      0.588      0.282      0.002          0          1
         0          0     0.1215    0.54675    0.33075      0.001          1
         0          0  0.0106667   0.181333      0.592      0.216          1
         0          0          0          0          0          1          1
```

Og sammenlignes med Mathematica

```
x = {0, 0.5, 1.2, 2.1, 2.6, 3};

Table[BSplineBasis[{k - 1, {0, 0, 0, 0, 1, 2, 3, 3, 3, 3}}, j, x[[i]]], {i, 1, 6},
 {j, 0, n - 1}]
```

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0.125 & 0.59375 & 0.260417 & 0.0208333 & 0 & 0 \\ 0 & 0.128 & 0.588 & 0.282 & 0.002 & 0 \\ 0 & 0 & 0.1215 & 0.54675 & 0.33075 & 0.001 \\ 0 & 0 & 0.0106667 & 0.181333 & 0.592 & 0.216 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

8.2.7 Bilag 7

Testning af '*house*'

```
Vektor x:
{0, 0.5, 1.2, 2.1, 2.6, 3}
Vektor y:
{1, 2, 3, 4, 5, 6}
xmin: 0 , xmax: 3

Indtast k: 3
Indtast l: 3
Knudepunkter t:
{0, 0, 0, 1, 2, 3, 3, 3}

Totalmatrix for Splines
         1          0          0          0          0          1
      0.25      0.625      0.125          0          0          2
         0       0.32       0.66       0.02          0          3
         0          0      0.405      0.585       0.01          4
         0          0       0.08       0.56       0.36          5
         0          0          0          0          1          6
```

```
1. Transformation:
j; 0
og v efter house(T,0):
{-0.0307764, 0.25, 0, 0, 0, 0}

2. Transformation:
j; 1
og v efter house(T,1):
{-0.0771574, 0.32, 0, 0, 0}

3. Transformation:
j; 2
og v efter house(T,2):
{-0.118476, 0.405, 0.08, 0}

4. Transformation:
j; 3
og v efter house(T,3):
{-0.22483, 0.56, 0}

5. Transformation:
j; 4
og v efter house(T,4):
{-0.702826, 1}
```

Mathematicas udregning:

```
dataA = ReadList[filnavnA, {Number, Number, Number, Number, Number, Number}]
```

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 \\ 0.25 & 0.625 & 0.125 & 0 & 0 & 2 \\ 0 & 0.32 & 0.66 & 0.02 & 0 & 3 \\ 0 & 0 & 0.405 & 0.585 & 0.01 & 4 \\ 0 & 0 & 0.08 & 0.56 & 0.36 & 5 \\ 0 & 0 & 0 & 0 & 1 & 6 \end{pmatrix}$$

```
T = dataA[[All]]
```

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 \\ 0.25 & 0.625 & 0.125 & 0 & 0 & 2 \\ 0 & 0.32 & 0.66 & 0.02 & 0 & 3 \\ 0 & 0 & 0.405 & 0.585 & 0.01 & 4 \\ 0 & 0 & 0.08 & 0.56 & 0.36 & 5 \\ 0 & 0 & 0 & 0 & 1 & 6 \end{pmatrix}$$

$$a1 = T[[All, 1]] * \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}; \quad a2 = T[[All, 2]] * \begin{pmatrix} 0 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}; \quad a3 = T[[All, 3]] * \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix};$$

$$a4 = T[[All, 4]] * \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \end{pmatrix}; \quad a5 = T[[All, 5]] * \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix}; \quad a6 = T[[All, 6]] * \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix};$$

```
v1 = a1 - Norm[a1] * e1
```

$$\begin{pmatrix} -0.0307764 \\ 0.25 \\ 0. \\ 0. \\ 0. \\ 0. \end{pmatrix}$$

```
v2 = a2 - Norm[a2] * e2
```

$$\begin{pmatrix} 0. \\ -0.0771574 \\ 0.32 \\ 0. \\ 0. \\ 0. \end{pmatrix}$$

```
v3 = a3 - Norm[a3] * e3
```

$$\begin{pmatrix} 0. \\ 0. \\ -0.118476 \\ 0.405 \\ 0.08 \\ 0. \end{pmatrix}$$

```
v4 = a4 - Norm[a4] * e4
```

$$\begin{pmatrix} 0. \\ 0. \\ 0. \\ -0.22483 \\ 0.56 \\ 0. \end{pmatrix}$$

```
v5 = a5 - Norm[a5] * e5
```

$$\begin{pmatrix} 0. \\ 0. \\ 0. \\ 0. \\ -0.702826 \\ 1. \end{pmatrix}$$

```
v6 = a6 - Norm[a6] * e6
```

$$\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

8.2.8 Bilag 8

C++'s udregning:

```
Totalmatrix for Splines
         1          0          0          0          0          1
      0.25      0.625      0.125          0          0          2
         0       0.32       0.66       0.02          0          3
         0          0      0.405      0.585       0.01          4
         0          0       0.08       0.56       0.36          5
         0          0          0          0          1          6

1. Transformation:
j; 0
og v efter house(T,0):
{-0.0307764, 0.25, 0, 0, 0, 0}
Efter anvend_house:
   1.03078  0.151585  0.030317          0          0   1.45521
         0 -0.606339 -0.121268          0          0  -1.69775
         0       0.32       0.66       0.02          0          3
         0          0      0.405      0.585       0.01          4
         0          0       0.08       0.56       0.36          5
         0          0          0          0          1          6
```

```
2. Transformation:
j; 1
og v efter house(T,1):
{-1.29194, 0.32, 0, 0, 0}
Efter anvend_house:
   1.03078  0.151585  0.030317         0         0   1.45521
         0    0.6856    0.41530.00933489         0   2.90171
         0         0  0.527098 0.0176878         0   1.86076
         0         0     0.405     0.585      0.01         4
         0         0      0.08      0.56      0.36         5
         0         0         0         0         1         6
3. Transformation:
j; 2
og v efter house(T,2):
{-0.142422, 0.405, 0.08, 0}
Efter anvend_house:
   1.03078  0.151585  0.030317         0         0   1.45521
         0    0.6856    0.41530.00933489         0   2.90171
         0         0   0.66952  0.434712  0.049065   4.48202
         0         0         0 -0.600873 -0.129524  -3.45396
         0         0         0  0.325754   0.33244   3.52761
         0         0         0         0         1         6
```

```
4. Transformation:
j; 3
og v efter house(T,3):
{-1.28437, 0.325754, 0}
Efter anvend_house:
   1.03078  0.151585  0.030317         0         0   1.45521
         0    0.6856    0.41530.00933489         0   2.90171
         0         0   0.66952  0.434712  0.049065   4.48202
         0         0         0  0.683494  0.272308   4.71771
         0         0         0         0  0.230523   1.45503
         0         0         0         0         1         6
5. Transformation:
j; 4
og v efter house(T,4):
{-0.795703, 1}
Efter anvend_house:
   1.03078  0.151585  0.030317         0         0   1.45521
         0    0.6856    0.41530.00933489         0   2.90171
         0         0   0.66952  0.434712  0.049065   4.48202
         0         0         0  0.683494  0.272308   4.71771
         0         0         0         0   1.02623   6.17351
         0         0         0         0         0 0.0700562
```

Mathematica's udregning:

```
In[6]:= T = dataA[[All]]

Out[6]= {{1, 0, 0, 0, 0, 1},
         {0.25, 0.625, 0.125, 0, 0, 2},
         {0, 0.32, 0.66, 0.02, 0, 3},
         {0, 0, 0.405, 0.585, 0.01, 4},
         {0, 0, 0.08, 0.56, 0.36, 5},
         {0, 0, 0, 0, 1, 6}}
```

1. Transformation

In[7]:=
```
a1 = T[[All, 1]] * {1, 1, 1, 1, 1, 1};
```

In[8]:=
```
v1 = a1 - Norm[a1] * e1
```

Out[8]=
$$\begin{pmatrix} -0.0307764 \\ 0.25 \\ 0. \\ 0. \\ 0. \\ 0. \end{pmatrix}$$

In[9]:=
```
H1 = IdentityMatrix[6] - (2 / Norm[v1]^2) * v1.Transpose[v1]
```

Out[9]=
$$\begin{pmatrix} 0.970143 & 0.242536 & 0. & 0. & 0. & 0. \\ 0.242536 & -0.970143 & 0. & 0. & 0. & 0. \\ 0. & 0. & 1. & 0. & 0. & 0. \\ 0. & 0. & 0. & 1. & 0. & 0. \\ 0. & 0. & 0. & 0. & 1. & 0. \\ 0. & 0. & 0. & 0. & 0. & 1. \end{pmatrix}$$

In[10]:=
```
T1 = H1.T
```

Out[10]=
$$\begin{pmatrix} 1.03078 & 0.151585 & 0.030317 & 0. & 0. & 1.45521 \\ 0. & -0.606339 & -0.121268 & 0. & 0. & -1.69775 \\ 0. & 0.32 & 0.66 & 0.02 & 0. & 3. \\ 0. & 0. & 0.405 & 0.585 & 0.01 & 4. \\ 0. & 0. & 0.08 & 0.56 & 0.36 & 5. \\ 0. & 0. & 0. & 0. & 1. & 6. \end{pmatrix}$$

2. Transformation

In[11]:=
```
a2 = T1[[A11, 2]] * {0, 1, 1, 1, 1, 1};
```

In[12]:=
```
v2 = a2 - Norm[a2] * e2
```

Out[12]=
$$\begin{pmatrix} 0. \\ -1.29194 \\ 0.32 \\ 0. \\ 0. \\ 0. \end{pmatrix}$$

In[13]:=
```
H2 = IdentityMatrix[6] - (2 / Norm[v2]^2) * v2.Transpose[v2]
```

Out[13]=
$$\begin{pmatrix} 1. & 0. & 0. & 0. & 0. & 0. \\ 0. & -0.884392 & 0.466745 & 0. & 0. & 0. \\ 0. & 0.466745 & 0.884392 & 0. & 0. & 0. \\ 0. & 0. & 0. & 1. & 0. & 0. \\ 0. & 0. & 0. & 0. & 1. & 0. \\ 0. & 0. & 0. & 0. & 0. & 1. \end{pmatrix}$$

In[14]:=
```
T2 = H2.T1
```

Out[14]=
$$\begin{pmatrix} 1.03078 & 0.151585 & 0.030317 & 0. & 0. & 1.45521 \\ 0. & 0.6856 & 0.4153 & 0.00933489 & 0. & 2.90171 \\ 0. & 5.55112 \times 10^{-17} & 0.527098 & 0.0176878 & 0. & 1.86076 \\ 0. & 0. & 0.405 & 0.585 & 0.01 & 4. \\ 0. & 0. & 0.08 & 0.56 & 0.36 & 5. \\ 0. & 0. & 0. & 0. & 1. & 6. \end{pmatrix}$$

3. Transformation

In[15]:=
```
a3 = T2[[All, 3]] * {0, 0, 1, 1, 1, 1};
```

In[16]:=
```
v3 = a3 - Norm[a3] * e3
```

Out[16]=
$$\begin{pmatrix} 0. \\ 0. \\ -0.142422 \\ 0.405 \\ 0.08 \\ 0. \end{pmatrix}$$

In[17]:=
```
H3 = IdentityMatrix[6] - (2 / Norm[v3]^2) * v3.Transpose[v3]
```

Out[17]=
$$\begin{pmatrix} 1. & 0. & 0. & 0. & 0. & 0. \\ 0. & 1. & 0. & 0. & 0. & 0. \\ 0. & 0. & 0.787277 & 0.604911 & 0.119489 & 0. \\ 0. & 0. & 0.604911 & -0.720159 & -0.339785 & 0. \\ 0. & 0. & 0.119489 & -0.339785 & 0.932882 & 0. \\ 0. & 0. & 0. & 0. & 0. & 1. \end{pmatrix}$$

In[18]:=
```
T3 = H3.T2
```

Out[18]=
$$\begin{pmatrix} 1.03078 & 0.151585 & 0.030317 & 0. & 0. & 1.45521 \\ 0. & 0.6856 & 0.4153 & 0.00933489 & 0. & 2.90171 \\ 0. & 4.37027 \times 10^{-17} & 0.66952 & 0.434712 & 0.049065 & 4.48202 \\ 0. & 3.35793 \times 10^{-17} & 6.245 \times 10^{-17} & -0.600873 & -0.129524 & -3.45396 \\ 0. & 6.63295 \times 10^{-18} & 2.77556 \times 10^{-17} & 0.325754 & 0.33244 & 3.52761 \\ 0. & 0. & 0. & 0. & 1. & 6. \end{pmatrix}$$

4. Transformation

In[19]:=
```
a4 = T3[[All, 4]] * {0, 0, 0, 1, 1, 1};
```

In[20]:=
```
v4 = a4 - Norm[a4] * e4
```

Out[20]=
$$\begin{pmatrix} 0. \\ 0. \\ 0. \\ -1.28437 \\ 0.325754 \\ 0. \end{pmatrix}$$

In[21]:=
```
H4 = IdentityMatrix[6] - (2 / Norm[v4]^2) * v4.Transpose[v4]
```

Out[21]=
$$\begin{pmatrix} 1. & 0. & 0. & 0. & 0. & 0. \\ 0. & 1. & 0. & 0. & 0. & 0. \\ 0. & 0. & 1. & 0. & 0. & 0. \\ 0. & 0. & 0. & -0.87912 & 0.476601 & 0. \\ 0. & 0. & 0. & 0.476601 & 0.87912 & 0. \\ 0. & 0. & 0. & 0. & 0. & 1. \end{pmatrix}$$

In[22]:=
```
T4 = H4.T3
```

Out[22]=
$$\begin{pmatrix} 1.03078 & 0.151585 & 0.030317 & 0. & 0. & 1.45521 \\ 0. & 0.6856 & 0.4153 & 0.00933489 & 0. & 2.90171 \\ 0. & 4.37027 \times 10^{-17} & 0.66952 & 0.434712 & 0.049065 & 4.48202 \\ 0. & -2.6359 \times 10^{-17} & -4.16728 \times 10^{-17} & 0.683494 & 0.272308 & 4.71771 \\ 0. & 2.18351 \times 10^{-17} & 5.41642 \times 10^{-17} & 5.55112 \times 10^{-17} & 0.230523 & 1.45503 \\ 0. & 0. & 0. & 0. & 1. & 6. \end{pmatrix}$$

5. Transformation

In[23]:=
$$a5 = T4[\![All, 5]\!] * \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix};$$

In[24]:=
$$v5 = a5 - Norm[a5] * e5$$

Out[24]=
$$\begin{pmatrix} 0. \\ 0. \\ 0. \\ 0. \\ -0.795703 \\ 1. \end{pmatrix}$$

In[25]:=
$$H5 = IdentityMatrix[6] - \left(2 / Norm[v5]^2\right) * v5.Transpose[v5]$$

Out[25]=
$$\begin{pmatrix} 1. & 0. & 0. & 0. & 0. & 0. \\ 0. & 1. & 0. & 0. & 0. & 0. \\ 0. & 0. & 1. & 0. & 0. & 0. \\ 0. & 0. & 0. & 1. & 0. & 0. \\ 0. & 0. & 0. & 0. & 0.224632 & 0.974444 \\ 0. & 0. & 0. & 0. & 0.974444 & -0.224632 \end{pmatrix}$$

In[26]:=
$$T5 = H5.T4$$

Out[26]=
$$\begin{pmatrix} 1.03078 & 0.151585 & 0.030317 & 0. & 0. & 1.45521 \\ 0. & 0.6856 & 0.4153 & 0.00933489 & 0. & 2.90171 \\ 0. & 4.37027 \times 10^{-17} & 0.66952 & 0.434712 & 0.049065 & 4.48202 \\ 0. & -2.6359 \times 10^{-17} & -4.16728 \times 10^{-17} & 0.683494 & 0.272308 & 4.71771 \\ 0. & 4.90485 \times 10^{-18} & 1.2167 \times 10^{-17} & 1.24696 \times 10^{-17} & 1.02623 & 6.17351 \\ 0. & 2.12771 \times 10^{-17} & 5.278 \times 10^{-17} & 5.40925 \times 10^{-17} & -2.498 \times 10^{-16} & 0.0700562 \end{pmatrix}$$

8.2.9 Bilag 9

Testning af HouseQR

C++'s udregning:

```
Totalmatrix for Splines
          1           0           0           0           0           0     1.79259
   0.296296    0.564815    0.132716  0.00617284           0           0    0.915559
   0.037037    0.518519    0.395062   0.0493827           0           0   -0.146742
          0        0.25    0.583333    0.166667           0           0   -0.330806
          0   0.0740741    0.549383    0.367284  0.00925926           0   -0.424341
          0  0.00925926    0.367284    0.549383   0.0740741           0   -0.576624
          0           0    0.166667    0.583333        0.25           0   -0.610615
          0           0   0.0493827    0.395062    0.518519    0.037037   -0.578598
          0           0  0.00617284    0.132716    0.564815    0.296296   -0.523458
          0           0           0           0           0           1   -0.555146
HouseQR:
    1.04363    0.178758   0.0516995  0.00350505           0           0     1.97237
          0    0.789934    0.582953    0.129663  0.00173653           0  -0.0392658
          0           0    0.799019    0.704478    0.127706  0.00457809   -0.984954
          0           0           0    0.683977    0.554908   0.0741692   -0.718678
          0           0           0           0    0.575947    0.251438   -0.469337
          0           0           0           0           0     1.01016   -0.550262
          0           0           0           0           0           0   -0.180832
          0           0           0           0           0           0   -0.119714
          0           0           0           0           0           0  -0.0649156
          0           0           0           0           0           0   -0.179495
```

Mathematica's udregning:

```
dataA = ReadList[filnavnA, {Number, Number, Number, Number, Number, Number, Number}];

T = dataA[[All]]
```

$$\begin{pmatrix} 1.04363 & 0.178758 & 0.0516995 & 0.00350505 & 0 & 0 & 1.97237 \\ 5.55112\times10^{-17} & 0.789934 & 0.582953 & 0.129663 & 0.00173653 & 0 & -0.0392658 \\ 6.93889\times10^{-18} & 0 & 0.799019 & 0.704478 & 0.127706 & 0.00457809 & -0.984954 \\ 0 & 0 & 0 & 0.683977 & 0.554908 & 0.0741692 & -0.718678 \\ 0 & 0 & 0 & -2.77556\times10^{-17} & 0.575947 & 0.251438 & -0.469337 \\ 0 & 0 & 0 & 2.77556\times10^{-17} & -1.38778\times10^{-17} & 1.01016 & -0.550262 \\ 0 & 0 & 0 & 5.55112\times10^{-17} & -6.93889\times10^{-18} & -1.73472\times10^{-18} & -0.180832 \\ 0 & 0 & 0 & 5.55112\times10^{-17} & 5.55112\times10^{-17} & -1.38778\times10^{-17} & -0.119714 \\ 0 & 0 & 0 & 2.77556\times10^{-17} & 5.55112\times10^{-17} & 1.38778\times10^{-17} & -0.0649156 \\ 0 & 0 & 0 & 0 & 0 & 1.11022\times10^{-16} & -0.179495 \end{pmatrix}$$

```
{Q, R} = QRDecomposition[T];

R // MatrixForm
```

atrixForm=

$$\begin{pmatrix} -1.04363 & -0.178758 & -0.0516995 & -0.00350505 & -9.41459\times10^{-19} & -3.04388\times10^{-20} & -1.97237 \\ 0. & -0.789934 & -0.582953 & -0.129663 & -0.00173653 & 6.88815\times10^{-21} & 0.0392658 \\ 0. & 0. & 0.799019 & 0.704478 & 0.127706 & 0.00457809 & -0.984954 \\ 0. & 0. & 0. & -0.683977 & -0.554908 & -0.0741692 & 0.718678 \\ 0. & 0. & 0. & 0. & -0.575947 & -0.251438 & 0.469337 \\ 0. & 0. & 0. & 0. & 0. & -1.01016 & 0.550262 \\ 0. & 0. & 0. & 0. & 0. & 0. & 0.288902 \end{pmatrix}$$

8.2.10 Bilag 10

Testning af bsub:

C++'s udregning:

```
Indtast k: 4
Indtast l: 3
xmin: 0 & xmax: 5
Knudepunkter t bestemmes:
Totalmatrix for Splines udregnes:
HouseQR udregnes:

c-værdi udregnes Rc=y vha. baglæns substitution:
{1.83331, 0.536219, -0.675809, -0.523477, -0.577086, -0.544728}
```

Mathematica's udregning:

```
R = dataT[[All, ;; 6]]
```

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0.296296 & 0.564815 & 0.132716 & 0.00617284 & 0 & 0 \\ 0.037037 & 0.518519 & 0.395062 & 0.0493827 & 0 & 0 \\ 0 & 0.25 & 0.583333 & 0.166667 & 1.59617\times10^{-47} & 0 \\ 0 & 0.0740741 & 0.549383 & 0.367284 & 0.00925926 & 0 \\ 0 & 0.00925926 & 0.367284 & 0.549383 & 0.0740741 & 0 \\ 0 & 0 & 0.166667 & 0.583333 & 0.25 & 1.89175\times10^{-47} \\ 0 & 0 & 0.0493827 & 0.395062 & 0.518519 & 0.037037 \\ 0 & 0 & 0.00617284 & 0.132716 & 0.564815 & 0.296296 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

```
y = {1.79259, 0.915559, -0.146742, -0.330806, -0.424341, -0.576624, -0.610615,
   -0.578598, -0.523458, -0.555146};

LeastSquares[R, y]
```

{1.83331, 0.536217, −0.675809, −0.523476, −0.577086, −0.544729}

8.2.11 Bilag 11

Forneden ses der en tabel over de forskelle ordenen k, hvor antal interval l holdes fast på $l = 16$.

Ordenen k:	$\|\|r\|\|$	$g(x) - f(x)$	$\|\|Ac - y\|\|$
1	2.239305229693787	2.122433332562517	2.239305229693787
2	0.9910938238092992	0.7092078431758415	0.9910940748911858
3	0.9345475877688721	0.48834413585554054	0.9345469784936544
4	0.8911214646937683	0.4697378550527895	0.8911217361840222
5	0.8985981023903244	0.4653209288570661	0.8985986220331906

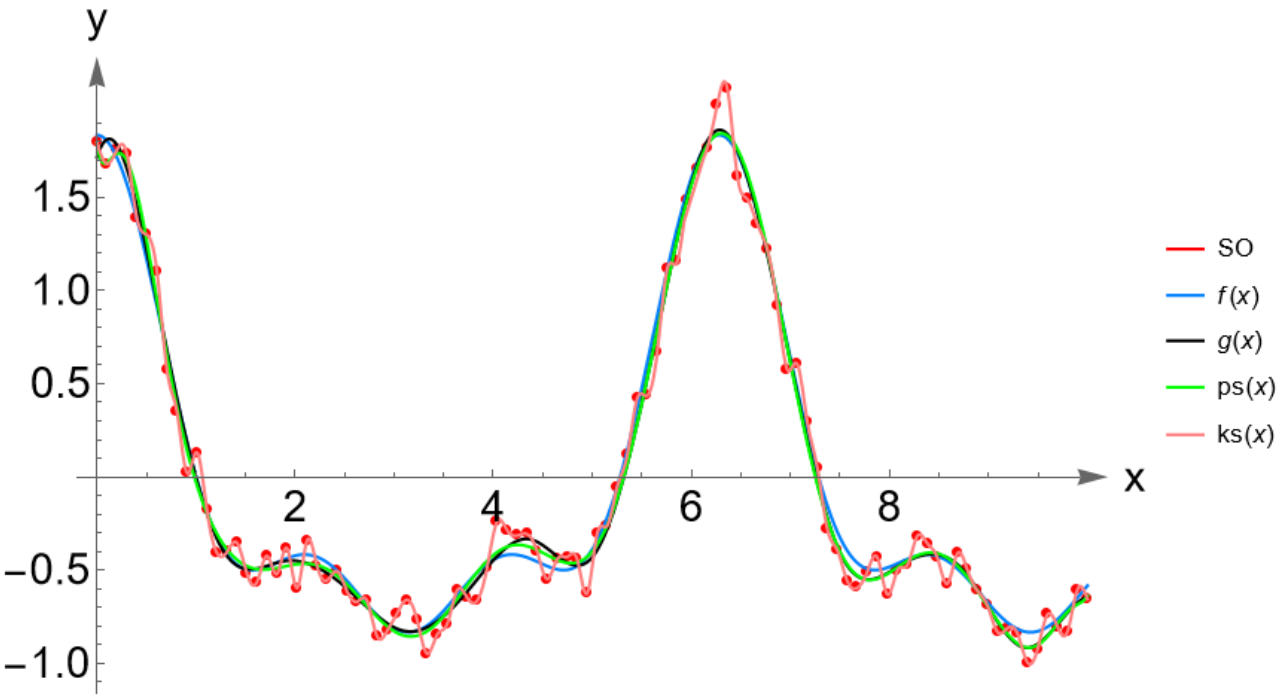
6	0.8850438572012977	0.4821843124333814	0.8850441686793147
8	0.8851163825626249	0.4840450165258046	0.8851161661328024
10	0.8829649892227343	0.488350229085208	0.8829630689268111
15	0.8318307229205302	0.5711936518324491	0.8318190943944067
20	0.7552582617027388	0.6691697051669838	0.7549966192813105
40	0.6202905452388667	0.7958912400322191	0.620288794982238

8.2.12 Bilag 12

Forneden ses der en tabel over de forskelle antal intervaller l, hvor orden k holdes fast på 4, $k = 4$

Antal interval l:	$\|\|r\|\|$	$g(x) - f(x)$	$\|\|Ac - y\|\|$
1	5.6471041183754345	5.539802663766798	5.647103521690508
2	4.878203197438076	4.774624799229285	4.878204577555058
4	4.18613270946861	4.01667523630174	4.186132264590851
6	2.9726512894912434	2.667791749486555	2.9726499563432647
8	1.7772536210900487	1.3402857876678176	1.7772539080533942
10	1.4984237697857148	1.035639095805178	1.4984243457926596
12	0.9745765674049691	0.5635315783689938	0.9745769703060462
13	0.950786949239963	0.501725013813651	0.9507870031008118
14	0.9062630282725864	0.4892582315305268	0.9062626079268808
15	0.9371606096817431	0.44047836262809553	0.937161471297872
16	0.8911214646937683	0.4697378550527895	0.8911217361840222
17	0.894205242156302	0.4802859901206795	0.4802859901206796
20	0.8865240408259898	0.485059620057093	0.8865240285463833
22	0.8325352085668744	0.5699687877175672	0.8325347766748266
26	0.8096250236804416	0.6040760025543295	0.8096248347972164
32	0.7705718224869975	0.6512269757787779	0.770572298689248
40	0.7252885516335497	0.7015345858518288	0.7252887176132407

8.2.13 Bilag 13



8.2.14 Bilag 14

Et skærmbillede fra C++-program hvor Householders metode sammenlignes med modificeret Gram Schmidt-metode.

```
Hilbertmatrix (20x8)
t: 1*pow(10,-3)

Løsningsvektor x udregnet af Householder:
{-0.014466, 0.131044, -0.119029, -0.221449, -0.113602, 0.127762, 0.433034, 0.757353}

Løsningsvektor x udregnet af Modificeret Gram Schmidt:
{-0.014466, 0.131044, -0.119029, -0.221449, -0.113602, 0.127762, 0.433034, 0.757353}

Den rigtige løsningsvektor x fra Mathematica udskrives
{-0.014466, 0.131044, -0.119029, -0.221449, -0.113602, 0.127762, 0.433034, 0.757353}

Sammenligning med de korrekte decimaler:
Antal korrekte decimaler:
{14, 14, 13, 14, 15, 14, 15, 14}
Gennemsnit for Householder: 14.125

{12, 11, 10, 10, 10, 11, 13, 11}
Gennemsnit for mgs: 11
```

Den korrekte løsningsvektor **x** udregnet i Mathematica:

```
t = 1 * 10^-3
```

$$\frac{1}{1000}$$

```
H = HilbertMatrix[20][[All, ;; 8]] + t * SparseArray[IdentityMatrix[20], {20, 8}] // N;

b = HilbertMatrix[20][[All, 9]];

x = LeastSquares[H, b] // N
```

$$\{-0.014466, 0.131044, -0.119029, -0.221449, -0.113602, 0.127762, 0.433034, 0.757353\}$$

```
NumberForm[x, 16] // N
```

umberForm=

$$\{-0.01446601454000257, 0.1310438622691721, -0.1190287052648864, -0.2214486450107251,$$
$$-0.1136020904773009, 0.1277622315069655, 0.4330337826652549, 0.7573527311066185\}$$