Bachelorprojekt

Ikke-krydsende fraktil-regression for vindkraftdata

Minh Haw Truong, s090088

Sofie Pødenphant Jensen, s093096

Kgs. Lyngby 18. juni 2012

Danmarks Tekniske Universitet

Abstract

This work outlines the basic concepts of linear quantile regression which is a statistical tool used to estimate the quantile curves of a conditional distribution. Each quantile curve is presented as a linear model using a set of spline basis functions in order to model the non-linear dependency between the respons variable and the explanatory variable treated here. An estimate of the unknown parameters of each model is uniquely determined by solving a linear optimization problem according to an asymmetric loss function using the Simplex-method. Computing the parameters of each model independently, however, may lead to unwanted crossings among the corresponding quantile-curves and non-crossing constraints are thus imposed to avoid such crossings.

In addition, different approaches have been developed and used to determine the linear models and the results of these are compared and analysed in terms of performance and reliability.

Resumé

I rapporten gives en redegørelse for lineær fraktil-regression, og hvordan dette redskab kan benyttes til at estimere fraktil-kurver for en betinget fordeling.
Hver fraktil-kurve angives ved en lineær model, som opstilles ved brug af spline-basisfunktioner, og bestemmes ved at minimere en asymmetrisk tabsfunktion. Til dette formål opstilles et lineært optimeringsproblem, som løses vha. Simplex-metoden, hvor der under indførelse af non-crossing-betingelser, også estimeres ikke-krydsende fraktil-modeller.

På baggrund af Simplex-metoden bestemmes det samlede sæt af fraktil-modeller ved forskellige metoder, hvor resultaterne fra disse metoder sammenlignes og analyseres. Metoderne implementeres i MATLAB hvor der i denne forbindelse også gives en redegørelse for implementeringen af disse.

Notation

Generel notation:

$\mathbb{R}$	:	Mængden af reelle tal.
$\mathbb{R}_0$	:	Mængden af ikke-negative reelle tal.
$\mathbb{N}$	:	Mængden af naturlige tal, {1,2,3,...}.
$\mathbb{R}^n$	:	Det $n-$dimensionelle reelle vektorrum.
$\mathbb{R}^{m\times n}$	:	Mængden af reelle matricer med dimensionen $m \times n$.

Vektorer og matricer:

$\mathbf{I}$	:	Identitetsmatricen. Notationen, $\mathbf{I}_k$, betyder at matricen har dimensionen $k \times k$.
$\mathbf{0}$	:	Matrix eller vektor bestående af nuller. Notationen, $\mathbf{0}_k$, betyder at dimensionen er k.
$\mathbf{e}$	:	Vektor bestående af 1-taller. Notationen, $\mathbf{e}_k$, betyder at dimensionen er k.
$\mathbf{x} = [x_i]$	:	Små bogstaver med fed skrift benyttes til at angive søjle vektorer.
$\mathbf{x}(h)$	:	Det h'te element i vektoren $\mathbf{x}$. h kan også være en mængde af indekser.
$\mathbf{A}$	:	Store bogstaver med fed skrift benyttes til at angive matricer.
$\mathbf{A}_{:,i}$	:	Den i'te søjle i matricen $\mathbf{A}$.
$\mathbf{A}(h)$	:	Den h'te række i matricen $\mathbf{A}$. h kan også være en mængde af indekser.
$\mathbf{x} \odot \mathbf{y}$	:	Elementvis multiplikation.

Funktioner og operatorer:

$\mathbf{x} \leq \mathbf{y}$	:	$x_i \leq y_i \ \ \forall i$.
$I(udtryk)$	:	Sandhedsoperator. Dvs. lig 1 hvis udtrykket er sandt og 0 ellers.
$\text{sign}(x)$	:	$\text{sign}(x) = I(x > 0) - I(x < 0)$. Dvs. $\text{sign}(0) = 0$.
$\overline{h}$	:	Komplementær-mængden til mængden h.

Forord

Dette bachelorprojekt (15 ECTS) er udarbejdet af Minh Haw Truong og Sofie Pødenphant Jensen, som afslutning på bachelor-uddannelsen *Matematik og Teknologi* på Danmarks Tekniske Universitet.

Vi vil gerne takke vores vejleder Jan Kloppenborg Møller[1] for hans hjælp og inspiration igennem hele projektforløbet og for altid at være behjælpelig, når tvivl opstod.

Ansvarsfordeling

Rapporten er udarbejdet i fællesskab, dog har følgende haft hovedansvaret for:

- Afsnit 2: *Fraktil-regression* - Sofie P. Jensen
- Afsnit 3: *Splines* - Minh H. Truong
- Afsnit 4: *Lineær optimering og Simplex-metoden* - Sofie P. Jensen
- Afsnit 5: *Generel opsætning* - Minh H. Truong
- Afsnit 6: *Modeller for ikke-krydsende fraktiler* - Sofie P. Jensen
- Afsnit 7: *Beregning af alle fraktiler på én gang* - Minh H. Truong

De afsnit, som ikke er benævnt ovenfor, samt de skrevne MATLAB-koder er udarbejdet i fællesskab.

Kgs. Lyngby, 18. juni 2012

Minh Haw Truong	Sofie Pødenphant Jensen
s090088	s093096

[1] `http://www.imm.dtu.dk/English/Research/Mathematical_Statistics/People.aspx?lg=showcommon&type=person&id=20293`

Indhold

1 Indledning

I takt med de stadigt voksende globale miljøproblemer og klimaforandringer, som en mulig konsekvens af menneskets udledning af CO_2 og andre former for forurening, øges interessen for alternative energikilder. Disse alternativer skal fungere som et supplement til, og i sidste ende en erstatning for, anvendelsen af fossile brændstoffer. Af mulige vedvarende energikilder kan nævnes solenergi, vandkraft, biobrændsel og ikke mindst vindkraft. Fælles for disse energikilder er, at energiudbyttet afhænger af vejrforhold, som mennesket ikke umiddelbart har indflydelse på. I dette projekt beskæftiger vi os med vindkraft, og idet man ikke er herre over det daglige udbytte af vindkraft, kan denne i stedet forsøges forudsagt.

For en energileverandør er der en klar økonomisk gevinst ved præcise estimater af produktionen. Et estimat giver leverandøren et fingerpeg om, hvor meget vind der kan forventes at blive afsat, og udløser ligeledes en forventning hos køberen. Den producerede vindkraft sælges på markedet, *NordPool*, hvor der tildeles økonomiske straffe, hvis ikke disse forventninger indfries. Straffen er ikke symmetrisk[2], dvs. at straffen for at give for høje og for lave estimater ikke er identiske. En god forudsigelse kræver derfor ikke alene et kendskab til middelværdien af vindkraftsproduktionen, men også i høj grad af et kendskab til selve fordelingen af denne. En sådan fordeling afhænger af en meteorologisk forudsigelse; altså ønskes et kendskab til den betingede fordeling af vindkraftsproduktionen givet en meteorologisk forudsigelse.

Det primære mål i projektet er at bestemme den betingede fordeling af den faktiske vindkraftsproduktion givet et estimat af denne. Til formålet benyttes fraktil-regression, hvor en model for de betingede fraktil-kurver bestemmes ud fra et givent datasæt bestående af estimerede og målte vindkraftsproduktioner.

Bestemmes fraktil-modellerne uafhængigt af hinanden kan der opstå et uønsket fænomen kaldet fraktil-krydsning. Fænomenet strider imod det fundamentale princip ved fraktiler og kan undgås ved at beregne alle de ønskede fraktil-kurver under den betingelse, at disse ikke må krydse hinanden. De betingede fraktiler bestemmes her ved brug af linear programmering, mere eksakt ved brug af Simplex-algoritmen. Derfor gives en redegørelse for sammenhængen mellem fraktil-regression og lineær programmering, der muliggør implementeringen af algoritmer til beregning af de krydsende og ikke-krydsende fraktil-modeller.

Algoritmerne implementeres i MATLAB, hvor vi indledningsvist beregner fraktilerne uden at tage højde for krydsninger for herefter at udvide algoritmen således, at fraktil-krydsning undgås. Endvidere sammenlignes de forskellige fraktil-modellers performance.

De indledende afsnit i rapporten klarlægger en række centrale begreber, som *lineær fraktil-regression*, *splines* samt *linear programmering*, herunder *Simplex-metoden*, der sammen muliggør implementeringen af de ovennævnte algoritmer.
Dernæst opstilles de forskellige Simplex-modeller til beregning af fraktil-kurverne, og un-

[2] http://www.nordpoolspot.com/

dervejs gives en redegørelse for implementeringen af de tilhørende algoritmer, hvor der inddrages dele af den skrevne kode.
Afslutningsvist anvendes algoritmerne på et givent datasæt med fokus på performance, herunder kvalitet og køretid.

Del I
Teori: Fraktil-regression, splines og lineær programmering

2 Fraktil-regression

I projektet anvendes et udbredt statistisk værktøj kaldet lineær regression, som bygger på den antagelse, at sammenhængen mellem en responsvariabel og én (eller flere) forklarende variable kan beskrives ved en lineær model. Ved lineær regression forstås normalt *mindste-kvadraters-fittet*, hvor en lineær regressionsmodel bestemmes ved at minimere en kvadratisk tabsfunktion, eller mere specifikt, ved at minimere summen af de kvadrerede residualer. Herved fås regressionskurven for den betingede middelværdi, som dog sjældent giver tilstrækkelig information om fordelingen af Y, hvis ikke Y er normalfordelt.

Benyttes i stedet lineær *fraktil*-regression, bestemmes en lineær model ved at inddrage en asymmetrisk tabsfunktion, som introduceres senere i afsnittet. Ved anvendelse af lineær fraktil-regression kan hele den betingede fordeling af Y bestemmes.

I projektet repræsenteres den observerede vindkraftsproduktion ved responsvariablen, Y, og den forklarende variable, X_{pred}, repræsenterer de prædikterede vindkraftsproduktioner. Da Y i dette tilfælde ikke kan antages at være normalfordelt, vil det altså ikke være tilstrækkeligt at benytte almindelig linear regression, hvis man gerne vil gå i detaljer med den betingede fordeling af Y. Derfor bruges i stedet lineær fraktil regression, som i det følgende forklares nærmere.

Når man benytter lineær regression antages det som sagt, at sammenhængen mellem en responsvariabel og de forklarende variable kan beskrives ved en lineær model. Den lineære model ses af det følgende:

$$y = f(\mathbf{x}) + r, \quad y, r \in \mathbb{R}, \tag{2.1}$$

hvor

$$f(\mathbf{x}) = \hat{\beta}_1 x_1 + \ldots + \hat{\beta}_K x_K. \tag{2.2}$$

Modellen består af en linearkombination af K basisfunktioner , $x_1, \ldots x_K$, med tilhørende reelle koefficienter, $\hat{\beta}_1, \ldots, \hat{\beta}_K$, samt et fejlled angivet ved r. Basisfunktionerne er funktioner af den forklarende variabel, og x_1 sættes ofte lig 1, således at $\hat{\beta}_1$ angiver skæringen med anden aksen [6].
Hvad der her forstås ved basisfunktioner kan bedst illustreres med et eksempel:

Eksempel 2.1 - Polynomiel basis

En samling af polynomier af grad 2, med reelle koefficienter, har en basis $\{1, x, x^2\}$. Ethvert reelt 2. grads polynomium kan således skrives som en linearkombination af disse basisfunktioner, dvs:

$$f(x) = \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3, \tag{2.3}$$

hvor $x_1 = 1$, $x_2 = x$, og $x_3 = x^2$.

Givet observationerne $\mathbf{x}_{pred} = [(x_{pred})_i]$ og $\mathbf{y} = [y_i]$ med $i = 1,\ldots,N$, kan modellen (2.1) opstilles på matrix form ved brug af en såkaldt designmatrix, $\mathbf{X}$, bestående af de førnævnte basisfunktioner:

$$\mathbf{y} = \mathbf{X}\hat{\boldsymbol{\beta}} + \mathbf{r}, \tag{2.4}$$

hvor

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}, \ \mathbf{X} = \begin{bmatrix} x_{1,1} & \ldots & x_{1,K} \\ \vdots & & \vdots \\ x_{N,1} & \ldots & x_{N,K} \end{bmatrix}, \ \hat{\boldsymbol{\beta}} = \begin{bmatrix} \hat{\beta}_1 \\ \hat{\beta}_2 \\ \vdots \\ \hat{\beta}_K \end{bmatrix}, \ \mathbf{r} = \begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_N \end{bmatrix},$$

$$\mathbf{y}, \mathbf{r} \in \mathbb{R}^N, \quad \mathbf{X} \in \mathbb{R}^{N\times K}, \quad \hat{\boldsymbol{\beta}} \in \mathbb{R}^K.$$

I det mest simple tilfælde fås modellen (2.1) ved

$$y = \hat{\beta}_1 + \hat{\beta}_2 x_{pred} + r, \quad r, y, x, \hat{\beta}_1, \hat{\beta}_2 \in \mathbb{R}, \tag{2.5}$$

som beskriver sammenhængen mellem x_{pred} og y ved en ret linje bestemt af koefficienterne, $\hat{\beta}_1$ og $\hat{\beta}_2$. Anvendes designmatricen, kan modellen skrives på formen (2.4), hvor

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}, \ \mathbf{X} = \begin{bmatrix} 1 & (x_{pred})_1 \\ 1 & (x_{pred})_2 \\ \vdots & \vdots \\ 1 & (x_{pred})_N \end{bmatrix}, \ \hat{\boldsymbol{\beta}} = \begin{bmatrix} \hat{\beta}_1 \\ \hat{\beta}_2 \end{bmatrix}, \ \mathbf{r} = \begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_N \end{bmatrix}.$$

Målet er nu at bestemme koefficientvektoren, $\hat{\boldsymbol{\beta}}$, ved at minimere tabsfunktionen, ρ, som er en funktion af residualerne, r. Ved normal lineær regression benyttes som sagt en kvadratisk tabsfunktion, som giver regressionskurven for den betingede middelværdi, men ved lineær fraktil-regression benyttes i stedet en asymmetrisk tabsfunktion, som ses af (2.6). Denne tabsfunktion benyttes ved beregning af de betingede fraktilkurver, og for enhver af disse, som ønskes beregnet, bestemmes en model af formen (2.2) ved sættet af koefficienter, $\hat{\boldsymbol{\beta}}$. Hver model inddeler datapunkterne via fraktilkurverne, som i fællesskab karakteriserer fordelingen af Y - heraf navnet fraktil-regression. Med andre ord kan den betingede fordeling af en given responsvariabel bestemmes, hvis alle fraktilkurverne kan beregnes.

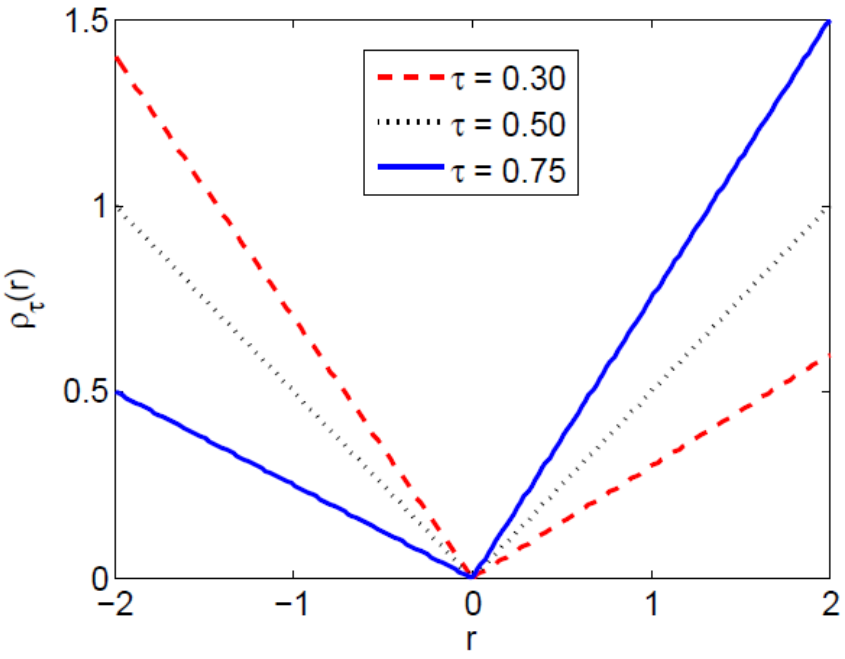
Her ses den førnævnte asymmetriske tabsfunktion, som blev introduceret af Koenker i 1978 [3]

$$\rho_\tau(r) = \begin{cases} \tau r, & \text{hvis } r \geq 0 \\ (\tau - 1)r, & \text{ellers} \end{cases}, \tag{2.6}$$

hvor $\tau \in [0,1]$ og $\rho_\tau(r_1) < \rho_\tau(r_2)$, hvis $|r_1| < |r_2|$, og $sign(r_1) = sign(r_2)$. τ angiver

her fraktilen som ønskes bestemt; sættes $\tau = 0.2$ bestemmes 20%-fraktilen.

På figur 2.1 ses tabsfunktionen for forskellige værdier af τ. Det ses, at tabsfunktionerne er asymmetriske, hvilket bevirker, at punkterne deles korrekt op. At de deles korrekt op, betyder her, at punkterne ikke blot deles korrekt op globalt, men at fraktil-kurverne også giver de betingede fraktiler. Dette bevises af Koenker i [3], og illustreres empirisk i afsnit 10.2.



Figur 2.1: Her ses tabsfunktionen, $\rho_\tau(r)$, for udvalgte værdier af τ. Det ses at $\rho_\tau(r)$ er en asymmetrisk funktion af r, som bevirker at $\mathbf{X}\hat{\boldsymbol{\beta}}$ inddeler datapunkterne korrekt.

Antag nu, at fraktilkurverne for et datasæt beregnes, hvor dataene er givet ved punkterne $((x_{pred})_1,y_1),\ldots,((x_{pred})_N,y_N)$, $(x_{pred})_i, y_i \in \mathbb{R}$. En fraktilkurve for den betingede fordeling af Y givet X_{pred} udtrykkes, som sagt, ved en model af K basisfunktioner:

$$\hat{Q}(\tau;\mathbf{x}) = \hat{\beta}_1(\tau)x_1 + \ldots + \hat{\beta}_K(\tau)x_K, \tag{2.7}$$

hvor der gælder, at $P(Y \leq Q(\tau;\mathbf{x})|X_{pred} = x_{pred}) = \tau$, og hvor τ angiver fraktilen, som ønskes bestemt. Residualerne, dvs. afvigelsen mellem observationerne, y_i, og fraktilkurven, $Q(\tau;\mathbf{x})$, bestemmes ved:

$$r_i = y_i - Q(\tau;\mathbf{x}). \tag{2.8}$$

Fraktilerne bestemmes ved at minimere summen af tabsfunktionerne, som afhænger af

residualerne. Dvs. et estimat af $\boldsymbol{\beta}$ findes ved at minimere tabet, $\sum_{i=1}^{N} \rho_\tau(r_i)$ m.h.t. $\boldsymbol{\beta}$ [3]:

$$\hat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}} \sum_{i=1}^{N} \rho_\tau(r_i) \tag{2.9}$$

Da hver af de førnævnte regressionsmodeller (2.7) bestemmes uafhængigt af hinanden, kan der opstå krydsninger i mellem de tilhørende fraktilkurver. Hvad der menes med krydsninger er, at der for to fraktiler givet ved (2.7),

$$\hat{Q}(\tau_1; \mathbf{x}) = \hat{\beta}_1(\tau_1)x_1 + \ldots + \hat{\beta}_K(\tau_1)x_K, \tag{2.10}$$

og

$$\hat{Q}(\tau_2; \mathbf{x}) = \hat{\beta}_1(\tau_2)x_1 + \ldots + \hat{\beta}_K(\tau_2)x_K, \tag{2.11}$$

givet $\tau_2 > \tau_1$, gælder at $\hat{Q}(\tau_1; \mathbf{x}) > \hat{Q}(\tau_2; \mathbf{x})$ i et vilkårligt punkt.

Fænomenet med krydsninger strider imod det fundamentale princip ved fraktiler, da det giver anledning til negative sandsynligheder, hvilket selvfølgelig ikke giver mening. Dette understreger problematikken omkring fraktil-krydsninger.

I afsnit 5 - 7 gives en detaljeret redegørelse for hvordan koefficienterne, $\hat{\boldsymbol{\beta}}$, i modellen, $\hat{Q}(\tau; \mathbf{x})$, bestemmes, og endvidere for hvordan sådanne krydsninger undgås.

3 Splines

Sammenhængen mellem de prædikterede og observerede vindkraftproduktioner, som behandles i projektet, er ikke-lineære, hvilket giver anledning til brugen af spline-basisfunktioner i den lineære regressionsmodel (2.2). Modellen kan skrives, som en linearkombination af K spline-basisfunktioner, som er funktioner af X_{pred}, således at den ikke-lineære sammenhæng beskrives ved den lineære model (2.4):

$$\mathbf{y} = \mathbf{X}\hat{\boldsymbol{\beta}} + \mathbf{r}, \tag{3.1}$$

hvor de K spline basisfunktioner repræsenteres ved søjlerne i designmatricen, $\mathbf{X}$.

Ved begrebet *splines* forstås en stykkevis regressionsfunktion, som har til formål at "fitte" et givent datasæt, $(x_1,y_1) \ldots (x_N,y_N)$, via separate kurvestykker. De enkelte kurvestykker fitter området mellem to *knudepunkter* og forbindes i selve knudepunkterne. I fællesskab danner de en samlet kurve, som beskriver sammenhængen mellem variablene x og y [2].

Eksempel 3.1

For at illustrere det grundlæggende princip ved spline-funktioner benyttes i det følgende eksempel, for enkelthedens skyld, en stykkevis lineær funktion, $f(x)$, til at fitte et givent datasæt (x,y), hvor

$$y = f(x) + r, \quad y, r \in \mathbb{R}. \tag{3.2}$$

I forlængelse heraf gives endvidere et eksempel på, hvordan ovenstående fit af y kan angives ved en lineær regressionsmodel på formen (3.1) bestående af basisfunktioner.

Til dette formål generes et datasæt, som ses af figur 3.1.

Af figuren ses tydeligt, at sammenhængen mellem x og y er stykkevist lineær. Funktionen, $f(x)$, som vi her benytter, estimerer datapunkterne på begge sider af knudepunktet, $c_1 = 3$, som i dette tilfælde let kan aflæses af figuren. Med to regressionslinjer behøves kun ét knudepunkt, hvor flere knudepunkter er påkrævet ved et fit af flere linjer. Funktionen, $f(x)$, ses af det følgende:

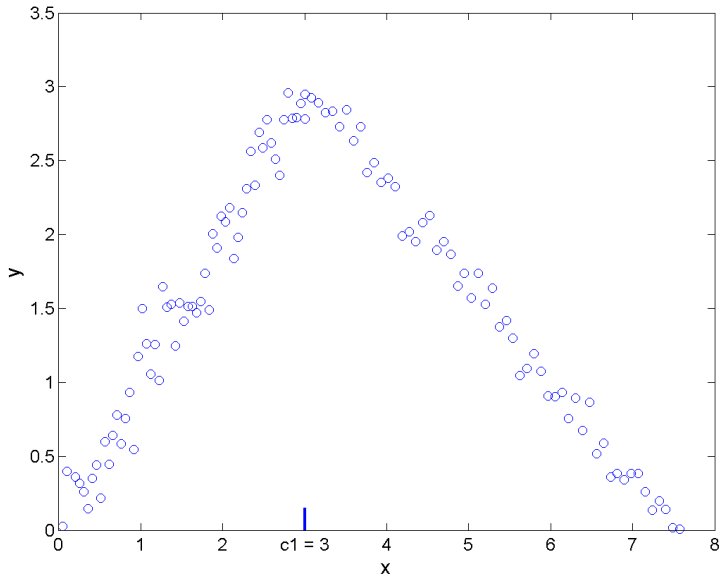
$$f(x) = \begin{cases} f_1(x) = a_1 + b_1 x, & \text{hvis } 0 \leq x \leq c_1 \\ f_2(x) = a_2 + b_2 x, & \text{hvis } c_1 \leq x. \end{cases} \tag{3.3}$$

Ved at bestemme de fire frie parametre for $f(x)$, dvs. koefficienterne, a_1, a_2, b_1 og b_2, fås herved et fit af y. På baggrund af figur 3.1 sættes $a_1 = 0$, og som tidligere nævnt er $f_1(x)$ og $f_2(x)$ forbundet i knudepunktet $c_1 = 3$. Herved fås altså betingelsen, $f_1(c_1) = f_2(c_1)$:

$$f_1(c_1) = f_2(c_1) \quad \Leftrightarrow \tag{3.4}$$

$$b_1 c_1 = a_2 + b_2 c_1 \quad \Leftrightarrow \tag{3.5}$$

$$a_2 = c_1(b_1 - b_2), \tag{3.6}$$



Figur 3.1: Her ses et sæt af simulerede datapunkter (x,y). Endvidere ses knudepunktet, c_1, i $x = 3$, hvor datapunkterne knækker.

hvorfor f kan skrives på følgende måde:

$$f(x) = \begin{cases} f_1(x) = b_1 x, & \text{hvis } c_0 \leq x \leq c_1 \\ f_2(x) = c_1 b_1 + b_2(x - c_1), & \text{hvis } c_1 \leq x. \end{cases} \tag{3.7}$$

Det ses, at $f(x)$, via omskrivningen, kun har to frie parametre, dvs. to frihedsgrader, og vælges $b_1 = 1$ og $b_2 = -\frac{2}{3}$, fås nu fittet, som ses af figur 3.2.

For at estimere y med én lineær model, frem for to adskilte lineære funktioner, inddrages et sæt af basisfunktioner; én basisfunktion for hver af de frie parametre:

$$B_1(x) \quad = \quad x \ , \qquad 0 < x, \tag{3.8}$$

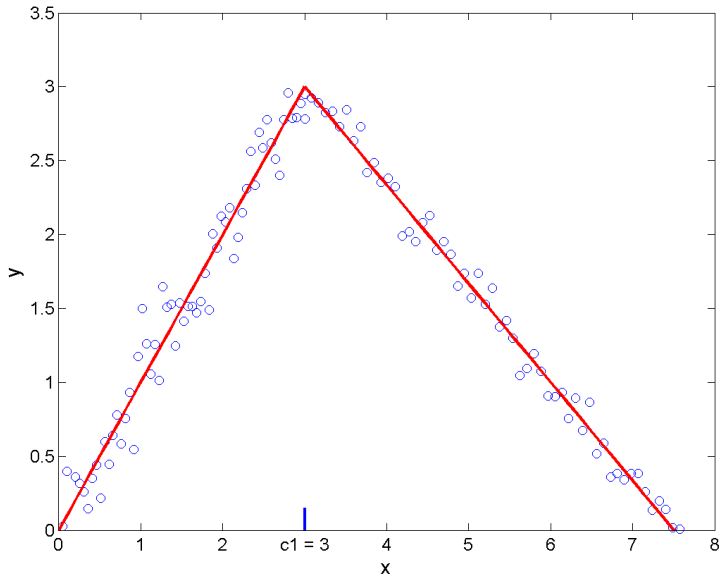
$$B_2(x) \quad = \quad \begin{cases} 0, & \text{hvis } 0 \leq x \leq c_1 \\ x - c_1, & \text{hvis } c_1 \leq x, \end{cases} \tag{3.9}$$

hvor $c_1 = 3$. Et plot af de valgte basisfunktioner ses af figur 3.3b. Herved opstilles, via basisfunktionerne, den lineære regressions model:

$$y = \beta_0 + \hat{\beta}_1 B_1(x) + \hat{\beta}_2 B_2(x) + r, \tag{3.10}$$

eller alternativt:

$$y = \mathbf{X}\hat{\boldsymbol{\beta}} + r, \tag{3.11}$$



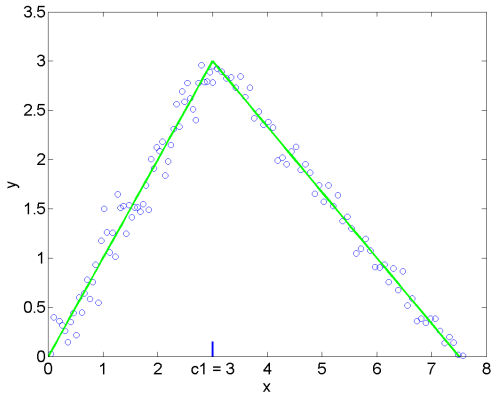
Figur 3.2: Her ses fittet af (x,y) via $f(x)$, hvor $b_1 = 1$ og $b_2 = -\frac{2}{3}$.

Designmatricen, $\mathbf{X}$, består herved af en konstant vektor af 1-taller samt to datavektorer bestående af basisfunktionerne. Den første datavektor antager værdien, x, hvorimod den anden datavektor antager værdien 0 indtil $x = c_1 = 3$, hvor den herefter antager værdien $x - c_1 = x - 3$. Herunder ses designmatricen:

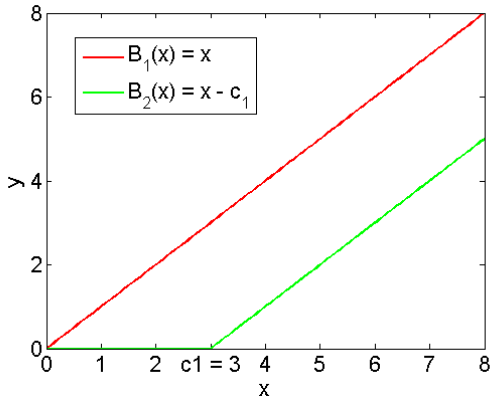
$$\mathbf{X} = \begin{bmatrix} 1 & x_1 & 0 \\ \vdots & \vdots & \vdots \\ 1 & c_1 & c_1 - c_1 \\ \vdots & \vdots & \vdots \\ 1 & x_N & x_N - c_1 \end{bmatrix}.$$

For koefficientvektoren, $\hat{\boldsymbol{\beta}} = \begin{bmatrix} 0 \\ 1 \\ -\frac{5}{3} \end{bmatrix}$, fås fittet i figur 3.3a, og bemærk, at fittet her, er identisk med fittet i figur 3.2.

I dette eksempel ses det, at et givent datasæt kan fittes ved en stykkevis lineær funktion i lighed med spline-funktioner, som vil blive beskrevet i det følgende. Endvidere ses det, at man, via basisfunktioner, kan omskrive den stykkevise lineære funktion, $f(x)$, således at den lineære regressionsmodel (3.11) opnås. Desuden blev det illustreret, hvordan antallet af frihedsgrader reduceres via indførelsen af betingelser. Da antallet af basisfunktioner, der skal til for at omskrive $f(x)$ til , er lig med antallet af frihedsgrader, medfører betingelserne desuden, at antallet af basisfunktioner bliver reduceret.



(a) Her ses fittet af y ved modellen (3.11) for $\hat{\beta} = (0, 1, -\frac{5}{3})$.



(b) Her ses de to valgte basisfunktioner, $B_1(x) = x$ og $B_2(x) = x - c_1$, hvor førstnævnte er en lineær funktion, og sidstnævnte er en stykkevis lineær funktion.

Figur 3.3: Her ses fittet af y, samt basisfunktionerne, $B_1(x)$ og $B_2(x)$.

3.1 Naturlig kubisk spline-basis

I projektet benyttes kubiske spline-funktioner til opbygningen af basis i modellen (3.1), dvs. at vi har en basis bestående af stykkevise 3. grads polynomier. Polynomier af denne orden er velegnede til at bestemme glatte funktioner, da de er kontinuert differentiable i knudepunkterne og derfor forhindrer skarpe kanter i fittet. Endvidere giver brugen 3. grads polynomier en tilpas grad af fleksibilitet i fittet af dataene [2].

Følgende definition på en kubisk spline er taget fra [1].

Definition: 3.1 Lad $a = c_0 < c_1 < \ldots < c_v = b$ være givne knudepunkter. En funktion, s, siges at være en kubisk spline i intervallet $[a, b]$, hvis

1. s, s' og s'' er kontinuerte i $[a, b]$.
2. s er et polynomium af grad $\leq (m = 3)$ i mellem hvert af knudepunkterne, $[c_{i-1}, c_i]$, hvor $i = 1, \ldots v$.

Man skal her notere sig, at $v + 1$ er antallet af knudepunkter, og $v - 1$, tilsvarende, er antallet af indre knudepunkter.
Den kubiske spline s er en stykkevis regressionsfunktion bestående af kubiske polynomier, s_i, hvor

$$s(x) = s_i(x), \quad c_{i-1} \leq x \leq c_i.$$

Enhver $s_i = a_i + b_i x + c_i x^2 + d_i x^3$ har fire koefficienter, hvilket giver $4v$ koefficienter, som skal bestemmes - herved haves $4v$ frihedsgrader. Da det ønskes, at spline-funktioner har visse egenskab, heriblandt kontinuitet (pkt. 1 i def. 3.1), indføres en række betingelser.

Kravet om kontinuitet giver kun $3(v-1)$ betingelser til bestemmelse af koefficienterne:

$$s_i(c_i) = s_{i+1}(c_i), \quad s_i'(c_i) = s_{i+1}'(c_i), \quad s_i''(c_i) = s_{i+1}''(c_i), \quad i = 1, \ \ldots \ v-1.$$

Vi efterlades altså med $4v - 3(v-1) = v + 3$ frihedsgrader. I projektet har vi valgt at benytte *naturlig splines* til at fitte dataene med, og disse splines fås ved tilføjelse af følgende to grænsebetingelser:

$$s_1''(c_0) = 0, \quad s_v''(c_v) = 0.$$

Sådanne linearitetskrav i endepunkter forhindrer pludselige udsving i splinekurven nær disse endepunkter [2].

Af årsager, som forklares senere i afsnittet, indføres endnu en grænsebetingelse:

$$s(c_0) = 0, \quad c_0 = 0.$$

Ved tilføjelse af disse tre grænsebetingelser, fås altså med v frihedsgrader. Dette medfører, at alle spline-funktioner, med ovenstående egenskaber, kan dannes vha. linearkombinationer af netop v lineært uafhængige spline-basisfunktioner. Enhver af disse spline-funktioner tilhører et funktionsrum af splines, $S_m(c_0, \ldots, c_v)$, ud fra hvilken vi vælger v lineært uafhængige basisfunktioner, $\{s\}_{i=2}^{v+1}$, som sammen danner en basis for $S_m(c_0, \ldots c_v)$. Hvordan en sådan serie af spline basisfunktioner bestemmes, ligger uden for projektets rammer og konstrueres via R[3].
Den sidste grænsebetingelse, $s(c_0) = 0$, bevirker, at vi selv kan tilføje skæringen med andenaksen ved at indføre basisfunktionen, x_1, som altså sættes til 1. Vi ender altså op med $v + 1$ basisfunktioner.

Basisfunktionerne anvendes i en regressionsmodel, hvis koefficienter således kan bestemmes entydigt. Bestemmelsen af fraktilkurverne, som beskrevet i forrige afsnit, tager sit udgangspunkt i modellen

$$\hat{Q}(\tau, \mathbf{x}) = \sum_{i=1}^{v+1} \hat{\beta}_i(\tau) x_i(x_{pred})$$

Basisfunktionerne angives ved designmatricen, $\mathbf{X}$, og herved fås den lineære regressionsmodel i (2.4):

$$y = \hat{Q}(\tau, \mathbf{x}) + r = \mathbf{X}\hat{\boldsymbol{\beta}}(\tau) + r,$$

$$\hat{\boldsymbol{\beta}}(\tau) \in \mathbb{R}^K, \quad \mathbf{X} \in \mathbb{R}^{N \times K}, \quad r \in \mathbb{R}^N.$$

[3] R er et software for statistisk databehandling

4 Lineær optimering og Simplex-metoden

Løsningen, $\hat{\boldsymbol{\beta}}$, til problemet, (2.9), kan bestemmes ved at løse et lineært optimeringsproblem via Simplex-metoden. I de følgende to afsnit gives en redegørelse for hvad et lineært optimeringsproblem indebærer, og hvordan et sådan problem kan løses vha. Simplex-metoden.

4.1 Lineær optimering

Afsnittet er baseret på [7].

I et lineært optimeringsproblem bestemmes en vektor, $\mathbf{x}$, som optimerer (minimerer eller maksimerer) objektfunktion hørende til problemet, givet et sæt af lineære begrænsninger. Det lineære optimeringsproblem er således formuleret, som enten et minimerings- eller et maksimeringsproblem [7]

$$\min\{\mathbf{c}^T\mathbf{x} : \text{sæt af lineære begrænsninger}\}, \tag{4.1}$$

$$\text{maks}\{\mathbf{c}^T\mathbf{x} : \text{sæt af lineære begrænsninger}\}, \tag{4.2}$$

hvor vektorerne $\mathbf{c}, \mathbf{x} \in \mathbb{R}^n$. Objektfunktionen er $\mathbf{c}^T\mathbf{x}$, hvor $\mathbf{c}$ er givet, og et ethvert punkt, $\mathbf{x}$, som opfylder de lineære begrænsninger, kaldes en mulig løsning.

Hvis man har et problem på formen (4.2), hvor objektfunktionen, $\mathbf{c}^T\mathbf{x}$, ønskes maksimeret, kan problemet formuleres som (4.1) under omskrivning af objektfunktionen til $-\mathbf{c}^T\mathbf{x}$, således at $-\mathbf{c}^T\mathbf{x}$ nu minimeres, og $\mathbf{c}^T\mathbf{x}$ derved maksimeres. Af denne grund behandler vi udelukkende problemer af formen (4.1).

De lineære begrænsninger kan opstilles på følgende tre måder:

- Ligheds-begrænsninger: $\mathbf{a}_i^T\mathbf{x} = b_i$
- Uligheds-begrænsninger: $\mathbf{a}_i^T\mathbf{x} \geq b_i$ eller $\mathbf{a}_i^T\mathbf{x} \leq b_i$
- Simple begrænsninger: $x_j \geq l_j$ eller $x_j \leq u_j$,

hvor $\mathbf{a}_i \in \mathbb{R}^n$, og b_i,l_j,$u_j \in \mathbb{R}$.

I resten af afsnittet antages det, for enkelthedens skyld, at sættet af simple begrænsninger er $\mathbf{x} \geq \mathbf{0}$.
Det siges, at et lineært optimeringsproblem antager sin standardform, hvis de lineære begrænsninger er angivet ved ligheds-begrænsninger:

$$\min\{\mathbf{c}^T\mathbf{x} : \mathbf{A}\mathbf{x} = \mathbf{b}, \mathbf{x} \geq \mathbf{0}\}, \quad \mathbf{A} \in \mathbb{R}^{m\times n}, \mathbf{b} \in \mathbb{R}^m. \tag{4.3}$$

Hvis sættet af lineære begrænsninger i stedet er givet ved uligheder, kan disse omskrives til ligheds-begrænsninger, så problemet igen antager sin standardform; til dette formål anvendes de såkaldte slack-variable, $s_i \geq 0$.

Antag nu, at de lineære begrænsninger er givet ved ulighederne $\mathbf{a}_i^T\mathbf{x} \geq b_i$. Slack-variablen indføres på følgende måde:

$$\mathbf{a}_i^T\mathbf{x} - s_i = b_i, \quad s_i \geq 0, \tag{4.4}$$

og har man begrænsningen $\mathbf{a}_i^T\mathbf{x} \leq b_i$ indføres de således:

$$\mathbf{a}_i^T\mathbf{x} + s_i = b_i, \quad s_i \geq 0. \tag{4.5}$$

På denne måde ændres ulighederne til ligheder, og problemet antager standardformen.

Et problems område af mulige løsninger, P, er mængden af punkter i $\mathbb{R}^n$, der opfylder *alle* begrænsningerne. Hvis denne mængde ikke er tom, kaldes problemet for løseligt; ellers kaldes det uløseligt. Problemet kaldes desuden for begrænset, hvis objektfunktionen er begrænset i området P; ellers kaldes det ubegrænset.

Vi betragter nu et løseligt problem, hvor $\mathbf{x}$ og $\mathbf{y}$ er to vilkårlige punkter i P. Der må gælde at:

$$\mathbf{c}^T\mathbf{y} = \mathbf{c}^T\mathbf{x} - \mathbf{c}^T(\mathbf{x} - \mathbf{y}). \tag{4.6}$$

Hvis $\mathbf{c}^T(\mathbf{x} - \mathbf{y}) > 0$, medfører det, at $\mathbf{c}^T\mathbf{x} > \mathbf{c}^T\mathbf{y}$, og $\mathbf{x}$ kan dermed ikke være en optimal løsning.
Hvis vi nu antager, at vi befinder os i punktet $\mathbf{x}$, og derefter bevæger os i retningen, $\mathbf{h}$, med den skridtlængde, $\alpha > 0$, som fører os til $\mathbf{y}$, må der altså gælde, at $\mathbf{y}$ kan skrives som $\mathbf{x} + \alpha\mathbf{h}$. Derfor kan (4.6) omskrives til:

$$\mathbf{c}^T\mathbf{y} = \mathbf{c}^T\mathbf{x} - \mathbf{c}^T(\mathbf{x} - (\mathbf{x} + \alpha\mathbf{h})) \Leftrightarrow \tag{4.7}$$

$$\mathbf{c}^T\mathbf{y} = \mathbf{c}^T\mathbf{x} + \mathbf{c}^T\alpha\mathbf{h}. \tag{4.8}$$

Det betyder, at $\mathbf{y}$ er en bedre løsning end $\mathbf{x}$, hvis $\mathbf{c}^T\alpha\mathbf{h} < 0 \Rightarrow \mathbf{c}^T\mathbf{h} < 0$. Er dette opfyldt, aftager objektfunktionen i retningen $\mathbf{h}$ for alle $\alpha > 0$, og $\mathbf{h}$ er derfor en aftagende retning. Det relative fald i objektfunktionen, dvs. hvor meget funktionen aftager pr. længdeenhed, er størst, hvis $\mathbf{h}$ vælges til at være den stejlest-aftagende retning (steepest descent), som må være:

$$\mathbf{h}_{sd} = -\mathbf{c}. \tag{4.9}$$

Bevæger man sig i en aftagende retning, bliver objektfunktionen altså ved med at aftage, så længe man følger denne retning. Som en umiddelbar konsekvens heraf, må den optimale løsning ligge på randen af P i et hjørnepunkt.
I det følgende eksempel anvendes en grafisk løsningsmetode for at illustrere ideen bag et lineært optimeringsproblem.

Eksempel 3.1

Betragt følgende lineære problem, hvor de lineære begrænsninger er angivet på kanonisk form:

$$\min\{\mathbf{c}^T\mathbf{x} : \mathbf{A}\mathbf{x} \geq \mathbf{b}, \mathbf{x} \geq \mathbf{0}\}, \tag{4.10}$$

$$\min\ \{5x_1 + 4x_2 : \begin{bmatrix} x_1 + x_2 \\ -\frac{1}{3}x_1 + x_2 \\ -\frac{1}{9}x_1 - x_2 \end{bmatrix} \geq \begin{bmatrix} 5 \\ 1 \\ -5 \end{bmatrix},\ x_1, x_2 \geq 0\}. \tag{4.11}$$

Omskrives problemet til standard formen ved indførelse af slackvariablene, s_1, s_2 og s_3, fås herved følgende problem:

$$\min\ \{5x_1 + 4x_2 : \begin{bmatrix} x_1 + x_2 - s_1 \\ -\frac{1}{3}x_1 + x_2 - s_2 \\ \frac{1}{9}x_1 + x_2 + s_3 \end{bmatrix} = \begin{bmatrix} 5 \\ 1 \\ 5 \end{bmatrix},\ x_1, x_2, s_1, s_2, s_3 \geq 0\}, \tag{4.12}$$

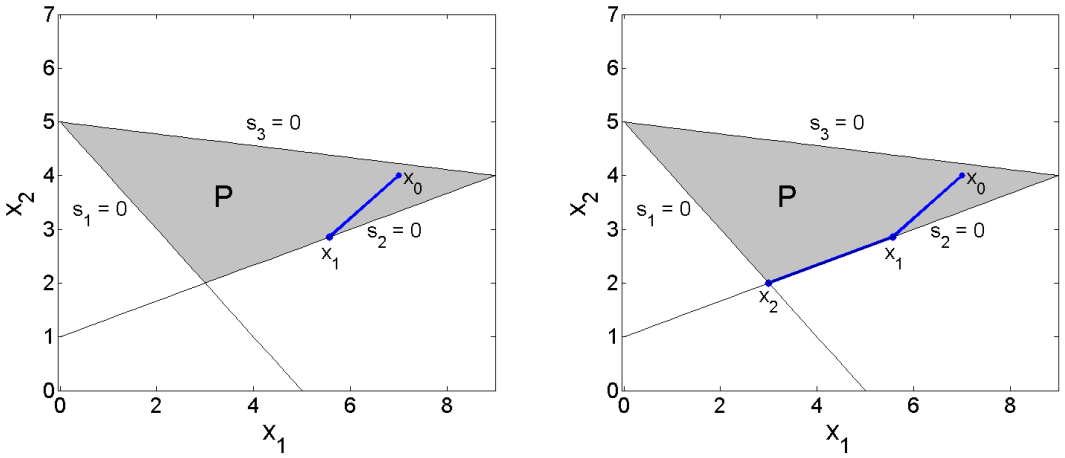
hvor

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ s_1 \\ s_2 \\ s_3 \end{bmatrix},\ \mathbf{c} = \begin{bmatrix} 5 \\ 4 \\ 0 \\ 0 \\ 0 \end{bmatrix},\ \mathbf{A} = \begin{bmatrix} 1 & 1 & -1 & 0 & 0 \\ -\frac{1}{3} & 1 & 0 & -1 & 0 \\ \frac{1}{9} & 1 & 0 & 0 & 1 \end{bmatrix},\ \mathbf{b} = \begin{bmatrix} 5 \\ 1 \\ 5 \end{bmatrix}.$$

I figur 4.1 nedenfor ses de tre lineære begrænsninger, som sammen indrammer et område, P. Der gælder for ethvert $\mathbf{x} \in P$, at alle begrænsningerne er opfyldte, og P er derfor mængden af mulige løsninger til det lineære problem.

Antag nu at begyndelsespunktet er $\mathbf{x}_0 = (7,4)$, som befinder sig i det indre af P. I dette punkt er værdien af objektfunktion lig med $\mathbf{c}^T\mathbf{x} = 5x_1 + 4x_2 = 5 \cdot 7 + 4 \cdot 4 = 51$. Da punktet ikke befinder sig på randen af P, eller med andre ord, ikke befinder sig på randen af én af de tre begrænsninger, gælder der, at de tilhørende slackvariable s_1, s_2 og s_3 er strengt større end nul. Fra $\mathbf{x}_0$ følges nu den stejlest aftagende retning, $\mathbf{h}_{sd} = -c = (-5, -4)$, indtil vi befinder os på randen af P i punktet $\mathbf{x}_1 = (\frac{39}{7}, \frac{20}{7})$, hvor $s_2 = 0$. Bevæger vi os herfra i retningen $\mathbf{h} = (-3, -1)$ og forbliver på randen, mindskes objektfunktionen yderligere, idet $\mathbf{c}^T\mathbf{h} = -19 < 0$. Vi havner herved i et hjørnepunkt $\mathbf{x}_2 = (3,2)$, hvor både $s_2 = 0$ og $s_1 = 0$.

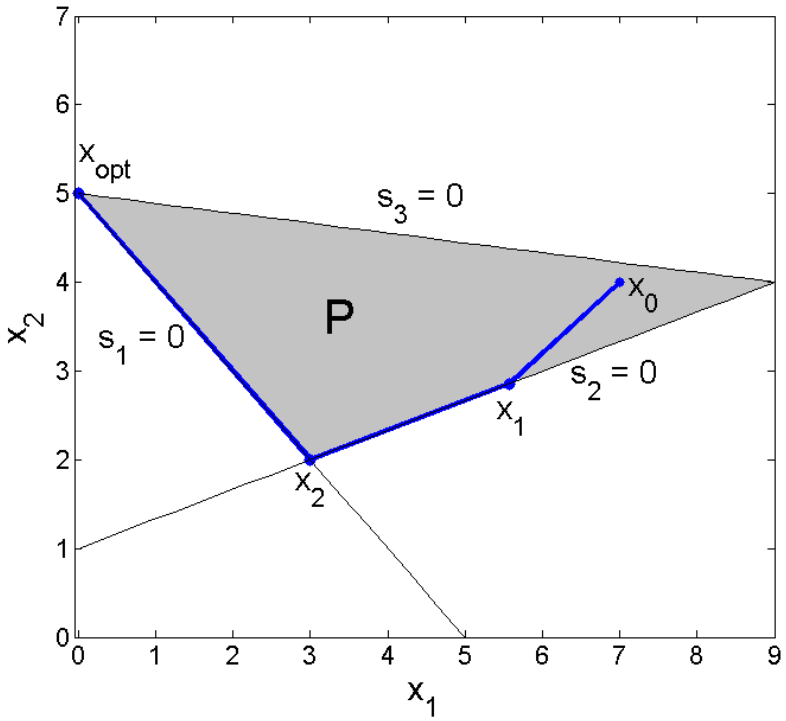
Det næste skridt leder os retningen $\mathbf{h} = (-1, 1)$, hvor objektfunktionen forbedres yderligere, da der også her gælder, at $\mathbf{c}^T\mathbf{h} = -1 < 0$. Dette leder os frem til hjørnepunktet $\mathbf{x} = (0,5)$, og slackvariablen s_2 ændrer sig fra $s_2 = 0$ til $s_2 > 0$ og $s_3 = 0$. Den eneste retning vi nu kan bevæge os i er $\mathbf{h} = (9, -1)$, men da $\mathbf{c}^T\mathbf{h} = 41 > 0$, forværres objektfunktionen. Vi befinder os, med andre ord, i det optimale punkt, $\mathbf{x}_{opt} = (0,5)$, og



(a) Fra $\mathbf{x}_0 = (7{,}5)$ til $\mathbf{x}_1 = (\frac{39}{7}, \frac{20}{7})$.

(b) Fra $\mathbf{x}_1 = (\frac{39}{7}, \frac{20}{7})$ til $\mathbf{x}_2 = (3{,}2)$.

Figur 4.1: Ovenfor ses den foretaget rute mod den optimale løsning.



Figur 4.2: Fra $\mathbf{x}_2 = (3{,}2)$ til den optimale løsning $\mathbf{x}_{opt} = (0{,}5)$.

objektfunktionen har værdien $\mathbf{c}^T\mathbf{x} = 20$, se figur 4.2.

Bevægelsen fra punktet $\mathbf{x}_1 = (\frac{39}{7}, \frac{20}{7})$ frem mod $\mathbf{x}_{opt}$ illustrerer det grundlæggende princip i Simplex-algoritmen, hvor hver iteration leder til et nyt hjørnepunkt med en lavere (eller en konstant) objektfunktion. Denne iterative fremgangsmåde fortsættes, indtil det optimale hjørne er fundet.

4.2 Simplex-metoden

Afsnittet er baseret på [6] og [7].

Grundprincippet i Simplex-algoritmen er at starte i et hjørnepunkt, $\mathbf{x}^{(k)}$, på randen af P, og herfra bevæge sig i en aftagende retning (langs med randen) til det næste naboliggende hjørnepunkt. Med en aftagende retning menes her, at objektfunktionen fra $\mathbf{x}^k$ til $\mathbf{x}^{k+1}$ aftager eller holdes konstant. Springene fra hjørnepunkt til hjørnepunkt giver et fald objektfunktionen og leder i sidste ende til det optimale hjørnepunkt. Der gælder for dette hjørnepunkt, at enhver retning væk fra punktet er stigende frem for aftagende, og ethvert naboliggende hjørne forøger objektfunktionen. Et sådan hjørnepunkt kaldes den optimale løsning.

Denne fremgangsmåde er mulig, idet objektfunktionen er lineær, og derfor enten stiger eller aftager (eller er konstant) i en given retning uafhængigt af skridtlængden. Der gælder desuden, at mængden, P, grundet de lineære begrænsninger, bliver konveks. Under disse nævnte omstændigheder bliver et lokalt minimum også et globalt minimum, som herved kan bestemmes ved at følge randen af P i en aftagende retning, som nævnt ovenfor.

Simplex-algoritmen benyttes nu til at løse et lineært optimeringsproblem på standardformen:

$$\min\{\mathbf{c}^T\mathbf{x} : \mathbf{A}\mathbf{x} \geq \mathbf{b}, \mathbf{x} \geq \mathbf{0}\},$$

$$\mathbf{A} \in \mathbb{R}^{m\times n}, \quad \mathbf{c}, \mathbf{x} \in \mathbb{R}^n, \quad \mathbf{b} \in \mathbb{R}^m.$$

De lineære begrænsninger repræsenteres ved et system af m lineære ligninger og n variable, hvor $m < n$.

Antag nu, at vi befinder os i hjørnepunktet $\mathbf{x}^{(k)}$. Indeks-mængden tilhørende elementerne i $\mathbf{x}^{(k)}$, $\{1...n\}$, deles i to mængder $\mathcal{B}$ og $\mathcal{C}$, som indeholder hhv. m og $n-m$ elementer. Dette gøres ved først at udvælge $n-m$ nul-elementer fra $\mathbf{x}^{(k)}$, hvis indeks-værdier udgør $\mathcal{C}$. De resterende indeks-elementer fra $\{1...n\}$ udgør $\mathcal{B}$. Dette betyder altså at:

$$\mathbf{x}_{\mathcal{B}}^{(k)} \geq \mathbf{0} \tag{4.13}$$

$$\mathbf{x}_{\mathcal{C}}^{(k)} = \mathbf{0} \tag{4.14}$$

Idet $\mathbf{x}^{(k)}$ er et hjørnepunkt, og derfor ligger på randen af mindst $n-m$ begrænsninger, eksisterer der mindst $n-m$ nul-elementer i $\mathbf{x}^{(k)}$, hvad enten disse er "originale" elementer eller de tilføjede slackvariable.
Elementerne i $\mathbf{x}_\mathcal{B}$ kaldes for basis-variable, mens elementerne i $\mathbf{x}_\mathcal{C}$ kaldes for ikke-basis-variable.

Når man bevæger sig fra et hjørnepunkt til et andet hjørnepunkt, er der et element $(x_\mathcal{C})_s \in \mathbf{x}_\mathcal{C}$, der nu bliver større end 0. Der er her tale om elementet der hører til den kant man bevæger sig væk fra. Tilsvarende er der et element i $(x_\mathcal{B})_q \in \mathbf{x}_\mathcal{B}$, der går mod 0, og som bliver 0 ved ankomsten til den næste kant, dvs. det næste hjørnepunkt. Ved ankomsten til det nye hjørnepunkt, bytter element q i $\mathcal{B}$, plads med element s i $\mathcal{C}$.

Nu defineres $\mathbf{B} = \mathbf{A}_{:,\mathcal{B}}$ og $\mathbf{C} = \mathbf{A}_{:,\mathcal{C}}$, og dermed må der gælde at:

$$\mathbf{A}\mathbf{x} = \mathbf{B}\mathbf{x}_\mathcal{B} + \mathbf{C}\mathbf{x}_\mathcal{C} = \mathbf{b}. \tag{4.15}$$

Da $\mathbf{x}_\mathcal{C}^{(k)} = \mathbf{0}$, må der i hjørnepunkterne gælde at:

$$\mathbf{A}\mathbf{x}^{(k)} = \mathbf{B}\mathbf{x}_\mathcal{B}^{(k)} = \mathbf{b} \Leftrightarrow \tag{4.16}$$

$$\mathbf{x}_\mathcal{B}^{(k)} = \mathbf{B}^{-1}\mathbf{b}, \tag{4.17}$$

forudsat at $\mathbf{B}$ er regulær.

4.2.1 Simplex-algoritmen

Antag nu, at vi befinder os i hjørnepunktet, $\mathbf{x}^{(k)}$, således at

$$\text{rank}(\mathbf{B}) = m, \tag{4.18}$$

$$\mathbf{x}_\mathcal{C}^{(k)} = \mathbf{0}, \tag{4.19}$$

$$\mathbf{x}_\mathcal{B}^{(k)} = \mathbf{B}^{-1}\mathbf{b} \geq \mathbf{0}. \tag{4.20}$$

Der undersøges om $\mathbf{x}^{(k)}$ er den optimale løsning ved at betragte objektfunktionen, $\mathbf{c}^T\mathbf{x}$.

Først omskrives (4.15):

$$\mathbf{x}_\mathcal{B} = \mathbf{B}^{-1}(\mathbf{b} - \mathbf{C}\mathbf{x}_\mathcal{C}) = \mathbf{B}^{-1}\mathbf{b} - \mathbf{B}^{-1}\mathbf{C}\mathbf{x}_\mathcal{C} = \mathbf{x}_\mathcal{B}^{(k)} - \mathbf{B}^{-1}\mathbf{C}\mathbf{x}_\mathcal{C}. \tag{4.21}$$

Ved at inddrage (4.21) kan objektfunktionen skrives som:

$$\begin{aligned}
\mathbf{c}^T\mathbf{x} &= \mathbf{c}_\mathcal{B}^T\mathbf{x}_\mathcal{B} + \mathbf{c}_\mathcal{C}^T\mathbf{x}_\mathcal{C} \\
&= \mathbf{c}_\mathcal{B}^T(\mathbf{x}_\mathcal{B}^{(k)} - \mathbf{B}^{-1}\mathbf{C}\mathbf{x}_\mathcal{C}) + \mathbf{c}_\mathcal{C}^T\mathbf{x}_\mathcal{C} \\
&= \mathbf{c}_\mathcal{B}^T\mathbf{x}_\mathcal{B}^{(k)} - \mathbf{c}_\mathcal{B}^T\mathbf{B}^{-1}\mathbf{C}\mathbf{x}_\mathcal{C} + \mathbf{c}_\mathcal{C}^T\mathbf{x}_\mathcal{C} \\
&= \mathbf{c}^T\mathbf{x}^{(k)} + (\mathbf{c}_\mathcal{C}^T - \mathbf{c}_\mathcal{B}^T\mathbf{B}^{-1}\mathbf{C})\mathbf{x}_\mathcal{C} \\
&= \mathbf{c}^T\mathbf{x}^{(k)} + (\mathbf{c}_\mathcal{C} - \mathbf{C}^T\mathbf{B}^{-T}\mathbf{c}_\mathcal{B})^T\mathbf{x}_\mathcal{C} \\
&= \mathbf{c}^T\mathbf{x}^{(k)} + \mathbf{d}^T\mathbf{x}_\mathcal{C},
\end{aligned} \tag{4.22}$$

hvor

$$\mathbf{d} = \mathbf{c}_{\mathcal{C}} - \mathbf{C}^T\mathbf{B}^{-T}\mathbf{c}_{\mathcal{B}}. \tag{4.23}$$

Elementerne i vektoren, $\mathbf{d}$, repræsenterer de relative ændringer i objektfunktionen fra hjørnepunktet, $\mathbf{x}^{(k)}$, til hvert af de naboliggende hjørnepunkter. Hvis $\mathbf{d} \geq 0$ vil ethvert spring til ét af de naboliggende hjørnepunkter øge objektfunktionen, idet $\mathbf{x}_{\mathcal{C}} \geq 0$. I dette tilfælde kan det bedst betale sig, at lade $\mathbf{x}_{\mathcal{C}}$ forblive 0, med andre blive i det nuværende hjørnepunkt, $\mathbf{x}^{(k)}$, hvorfor dette må være den optimale løsning.

Hvis $\mathbf{d} < 0$ betaler det sig at springe til et nyt hjørnepunkt, hvorfor der må gælde, at $\mathbf{x}^{(k)}$ ikke er den optimale løsning. I dette tilfælde, vælges et negativt element, $d_s \in \mathbf{d}$, og det tilhørende element $(x_{\mathcal{C}})_s \in \mathbf{x}_{\mathcal{C}}$ ændres. Man bevæger sig altså i retningen h, jvf. (4.21).

$$\mathbf{h} = \mathbf{B}^{-1}\mathbf{C}_{:,s}. \tag{4.24}$$

Den største værdi elementet $(x_{\mathcal{C}})_s$ kan påtage sig er lig den skridtlængde, α, man kan gå inden det næste hjørne rammes, dvs. den skridtlængde der skal til, før et element i $\mathbf{x}_{\mathcal{B}}$ bliver 0. Derfor beregnes for ethvert element i $\mathbf{x}_{\mathcal{B}}$, hvor langt man skal bevæge sig i den givne retning, før det pågældende element bliver 0. Disse skridtlængder beregnes således:

$$\sigma_j = \begin{cases} (x_{\mathcal{B}}^{(k)})_j/h_j & \text{hvis} \quad h_j > 0 \\ \infty & \text{hvis} \quad h_j \leq 0 \end{cases},$$

hvor den endelige skridtlængde er

$$\alpha = \min\{\sigma_1,...,\sigma_m\}. \tag{4.25}$$

Hvis $\alpha = \infty$, kan man bevæge sig uendelig langt i en aftagende retning. Objektfunktionen bliver herved lig $-\infty$, og problemet er altså ubegrænset. Hvis dette ikke er tilfældet, beregnes $\mathbf{x}_{\mathcal{B}}$ i det nye hjørne:

$$\mathbf{x}_{\mathcal{B}}^{(k+1)} = \mathbf{x}_{\mathcal{B}}^{(k)} - \alpha\mathbf{h}, \tag{4.26}$$

og elementerne $(\mathbf{x}_{\mathcal{B}})_q$ og $(\mathbf{x}_{\mathcal{C}})_s$ bytter plads, hvor

$$q = \arg\min_j\{\sigma_j\} \quad \Rightarrow \quad (\mathbf{x}_{\mathcal{B}}^{(k+1)})_q = \alpha. \tag{4.27}$$

Ydermere fås $\mathcal{B}^{k+1}$ og $\mathcal{C}^{k+1}$ ved at bytte om på element nr. q i $\mathcal{B}$ og element nr. s i $\mathcal{C}$.

Herefter beregnes $\mathbf{d}$ på ny, og hvis $\mathbf{d} \leq \mathbf{0}$, er løsningen stadig ikke optimal, og der foretages endnu en iteration. Denne procedure gentages indtil $\mathbf{d} > \mathbf{0}$, således at løsningen er optimal.

Del II
Fraktil-regression og linear programmering

5 Generel opsætning

Afsnittet er skrevet på baggrund af [6].

I følgende afsnit beskrives hvordan koefficienterne i en fraktil-regressionsmodel, (2.7), kan bestemmes ved løsning af et lineært optimeringsproblem vha. Simplex-metoden. Et lineært optimeringsproblem på standardformen ses i det følgende, jvf. afsnit 4.1:

$$\min\{\mathbf{c}^T\mathbf{x} : \mathbf{A}\mathbf{x} = \mathbf{b}, \mathbf{x} \geq \mathbf{0}\}. \tag{5.1}$$

Optimeringsproblemet (2.9) kan ved omformulering angives ved et lineært minimeringsproblem på ovenstående form. Omformuleringen indledes ved at omskrive residualerne, $\mathbf{r}$:

$$\mathbf{r} = \mathbf{r}^+ - \mathbf{r}^-, \tag{5.2}$$

hvor

$$r_i^+ = I(r_i \geq 0)r_i \quad \text{og} \quad r_i^- = -I(r_i \leq 0)r_i. \tag{5.3}$$

Der gælder således for r_i^+ og r_i^- at

$$r_i^+ > 0 \;\Rightarrow\; r_i^- = 0 \quad \text{og} \quad r_i^- > 0 \;\Rightarrow\; r_i^+ = 0. \tag{5.4}$$

Via denne omskrivning antager (2.9) den ønskede form:

$$\min\{\tau\mathbf{e}^T\mathbf{r}^+ + (1-\tau)\mathbf{e}^T\mathbf{r}^- : \mathbf{X}\boldsymbol{\beta} + \mathbf{r}^+ - \mathbf{r}^- = \mathbf{y}, \;\; \mathbf{r}^+, \mathbf{r}^- \in \mathbb{R}_0^N, \;\; \boldsymbol{\beta} \in \mathbb{R}^K\}, \tag{5.5}$$

som forkortes til:

$$\min\{\mathbf{c}^T\mathbf{x} : \mathbf{A}\mathbf{x} = \mathbf{b}, \;\; \mathbf{r}^+, \mathbf{r}^- \geq \mathbf{0}, \;\; \boldsymbol{\beta} \in \mathbb{R}^K\}, \tag{5.6}$$

hvor

$$\mathbf{c} = \begin{bmatrix} \mathbf{0}_K \\ \tau\mathbf{e} \\ (1-\tau)\mathbf{e} \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} \boldsymbol{\beta} \\ \mathbf{r}^+ \\ \mathbf{r}^- \end{bmatrix}, \quad \mathbf{A} = [\mathbf{X}, \mathbf{I}, -\mathbf{I}], \quad \mathbf{b} = \mathbf{y}.$$

Vektoren $\mathbf{x}$ indeholder koefficienterne, $\boldsymbol{\beta}$, hørende til regressionsmodellen samt slackvariable repræsenteret ved $\mathbf{r}^+$ og $\mathbf{r}^-$. Bemærk at $\boldsymbol{\beta}$ her både kan antage positive og negative værdier til forskel fra $\mathbf{x}$ i (5.1).

Løsningen til problemet, $\mathbf{x}$, ligger i et hjørnepunkt, $\mathbf{x}^{(k)}$, med K aktive begrænsninger, jvf. afsnit 4.1. At den i'te begrænsning er aktiv, betyder her, at man befinder sig på grænsen mellem, at r_i^+ er større end 0 og, at r_i^- er større end 0, dvs. i et punkt, hvor de begge er 0. Da der som sagt skal være mindst K aktive begrænsninger, må det dermed betyde, at

mindst K elementer i $\mathbf{r}$ er lig med 0.

Nu betegner $h \in \mathbb{N}^K$ en indeks-mængde, der bevirker, at $\mathbf{r}(h) = \mathbf{0}$, og der gælder dermed at:

$$\begin{aligned} \mathbf{X}(h)\boldsymbol{\beta} + \mathbf{r}(h) &= \mathbf{y}(h) \quad \Leftrightarrow \\ \mathbf{X}(h)\boldsymbol{\beta} &= \mathbf{y}(h) \end{aligned} \tag{5.7}$$

Der bør gælde at $\text{rank}(\mathbf{X}) = K$, da basisfunktionerne skal være lineært uafhængige, idet man ellers ville kunne skrive én eller flere af funktionerne, som en lineær kombination af de andre, hvilket ville betyde, at problemet blev singulært. Hvis $\text{rank}(\mathbf{X}) = K$, eksisterer der en indeks-mængde, h, med ovennævnte egenskab (5.7), som opfylder at $\text{rank}(\mathbf{X}(h)) = K$. Ved brugen af h kan $\boldsymbol{\beta}$ bestemmes, ved at løse det regulære problem.

Grundprincippet bag modellen er nu klarlagt, og derfor kan problemet så småt løses vha. Simplex-metoden, jvf. afsnit 4.2.

Det skal dog bemærkes, at $\mathbf{B}^{-1}$, grundet problemets størrelse, er kostbar at beregne[4], men det er muligt at strukturere $\mathbf{B}$ på en sådan måde, at $\mathbf{B}^{-1}$ kan bestemmes vha. allerede beregnede matricer og $\mathbf{X}(h)^{-1}$.

I det følgende afsnit forklares det, hvordan man beregner $\mathbf{B}^{-1}$ på denne måde.

5.1 Beregning af $\mathbf{B}^{-1}$

Når vi befinder os i et hjørnepunkt, gælder der, at

$$\mathbf{A}\mathbf{x}^{(k)} = \mathbf{B}\mathbf{x}_{\mathcal{B}}^{(k)} = \mathbf{b}. \tag{5.8}$$

Samtidig ved vi, at $\mathbf{A}\mathbf{x}^{(k)}$ kan skrives som

$$\mathbf{A}\mathbf{x}^{(k)} = \mathbf{X}\boldsymbol{\beta} + \mathbf{r} = \mathbf{b}. \tag{5.9}$$

Derfor må derfor gælde, at

$$\mathbf{B}\mathbf{x}_{\mathcal{B}}^{(k)} = \mathbf{X}\boldsymbol{\beta} + \mathbf{r} = \mathbf{b}. \tag{5.10}$$

Dette er et ligningssystem med N ligninger. De K ligninger, der svarer til indeks-værdierne i h, kan skrives som:

$$\mathbf{B}(h)\mathbf{x}_{\mathcal{B}}^{(k)} = \mathbf{X}(h)\boldsymbol{\beta} + \mathbf{r}(h) = \mathbf{X}(h)\boldsymbol{\beta} = \mathbf{y}(h), \tag{5.11}$$

Nu betegner $\overline{h}$ komplementær-mængden til h, hvilket betyder at $\overline{h}$ indeholder index-værdierne $\{1, \ldots, N\} \setminus \{h\}$. De ligninger, der svarer til indeks-værdierne i $\overline{h}$, kan skrives

[4]Der er i det mest simple tilfælde lige så mange begrænsninger som der er datapunkter.

som:

$$\mathbf{B}(\overline{h})\mathbf{x}_{\mathcal{B}}^{(k)} = \mathbf{X}(\overline{h})\boldsymbol{\beta} + \mathbf{r}(\overline{h}) = \mathbf{X}(\overline{h})\boldsymbol{\beta} + \text{sign}(\mathbf{r}(\overline{h})) \odot |\mathbf{r}(\overline{h})| = \mathbf{y}(\overline{h}), \tag{5.12}$$

Derved kan man, ved ombytning af rækkerne, angive $\mathbf{B}$ ved:

$$\mathbf{B} = \begin{bmatrix} \mathbf{X}(h) & \mathbf{0} \\ \mathbf{X}(\overline{h}) & \mathbf{P} \end{bmatrix}, \tag{5.13}$$

således at

$$\mathbf{B}\mathbf{x}_{\mathcal{B}} = \begin{bmatrix} \mathbf{X}(h) & \mathbf{0} \\ \mathbf{X}(\overline{h}) & \mathbf{P} \end{bmatrix} \mathbf{x}_{\mathcal{B}} = \begin{bmatrix} \mathbf{y}(h) \\ \mathbf{y}(\overline{h}) \end{bmatrix}, \tag{5.14}$$

hvor $\mathbf{P}$ er en diagonal-matrix med $\text{sign}(\mathbf{r}(\overline{h}))$ i diagonalen. Omstruktureringen af $\mathbf{B}$ leder til en tilsvarende omstrukturering af $\mathbf{x}_{\mathcal{B}}$, som nu antager formen:

$$\mathbf{x}_{\mathcal{B}} = \begin{bmatrix} \boldsymbol{\beta} \\ |\mathbf{r}(\overline{h})| \end{bmatrix}. \tag{5.15}$$

Nu skrives $\mathbf{B}^{-1}$ som:

$$\mathbf{B}^{-1} = \begin{bmatrix} \mathbf{B}_{11}^{-1} & \mathbf{B}_{12}^{-1} \\ \mathbf{B}_{21}^{-1} & \mathbf{B}_{22}^{-1} \end{bmatrix}, \tag{5.16}$$

hvor $\mathbf{B}_{11}^{-1} \in \mathbb{R}^{K\times K}$, $\mathbf{B}_{12}^{-1} \in \mathbb{R}^{K\times(N-K)}$, $\mathbf{B}_{21}^{-1} \in \mathbb{R}^{(N-K)\times K}$ og $\mathbf{B}_{22}^{-1} \in \mathbb{R}^{(N-K)\times(N-K)}$.

Per definition gælder der, at $\mathbf{B}\cdot\mathbf{B}^{-1} = \mathbf{I}$, og dette medfører, at

$$\begin{aligned}
\mathbf{X}(h)\mathbf{B}_{11}^{-1} + \mathbf{0}\mathbf{B}_{21}^{-1} &= \mathbf{I}, && (5.17)\\
\mathbf{X}(h)\mathbf{B}_{12}^{-1} + \mathbf{0}\mathbf{B}_{22}^{-1} &= \mathbf{0}, && (5.18)\\
\mathbf{X}(\overline{h})\mathbf{B}_{11}^{-1} + \mathbf{P}\mathbf{B}_{21}^{-1} &= \mathbf{0}, && (5.19)\\
\mathbf{X}(\overline{h})\mathbf{B}_{12}^{-1} + \mathbf{P}\mathbf{B}_{22}^{-1} &= \mathbf{I}, && (5.20)
\end{aligned}$$

hvilket må betyde, at

$$\begin{aligned}
\mathbf{B}_{11}^{-1} &= \mathbf{X}(h)^{-1}, && (5.21)\\
\mathbf{B}_{12}^{-1} &= \mathbf{0}, && (5.22)\\
\mathbf{B}_{21}^{-1} &= -\mathbf{P}\mathbf{X}(\overline{h})\mathbf{X}(h)^{-1}, && (5.23)\\
\mathbf{B}_{22}^{-1} &= \mathbf{P}, && (5.24)\\
& && (5.25)
\end{aligned}$$

hvor $\mathbf{P} = \mathbf{P}^{-1}$, fordi det er en diagonal-matrix med værdierne 1 og -1.

På denne måde kan $\mathbf{B}^{-1}$ dannes ud fra et produkt af kendte matricer og $\mathbf{X}(h)^{-1}$. Man

undgår herved at invertere $\mathbf{B}$, hvilket er fordelagtigt ud fra et programmeringsmæssigt synspunkt, idet det er kostbart at invertere store matricer.
Da strukturen af $\mathbf{B}$ er ændret, skal $\mathbf{C}$ struktureres på en tilsvarende måde. Vektoren, $\mathbf{x}_{\mathcal{C}}$, bestående af $\mathbf{r}(h)$, som inddeles i $\mathbf{r}(h)^+$ og $\mathbf{r}(h)^-$, ser ud som følger:

$$\mathbf{x}_{\mathcal{C}} = \begin{bmatrix} \mathbf{r}(h)^+ \\ \mathbf{r}(h)^- \end{bmatrix}, \tag{5.26}$$

hvorfor matricen, $\mathbf{C}$, får følgende struktur:

$$\mathbf{C} = \begin{bmatrix} \mathbf{I}_K & -\mathbf{I}_K \\ 0 & 0 \end{bmatrix}, \quad \mathbf{C} \in \mathbb{R}^{N \times 2K}, \tag{5.27}$$

og angiver de $2K$ mulige retninger[5] fra hjørnepunktet, $\mathbf{x}^{(K)}$. Der gælder, med andre ord, at man for enhver slackvariabel uden for basis, $r(h_i) \in \mathbf{r}(h)$, i hjørnepunktet, $\mathbf{x}^{(K)}$, kan bevæge sig i to retninger svarende til at gøre $r(h_i)$ positiv eller negativ.

Det er nu blevet klarlagt, hvordan koefficienterne i en linear fraktil-regressionsmodel kan bestemmes ved at løse et lineært optimeringsproblem. Ovenstående fremgangsmåde kan dog lede til fraktil-krydsninger, idet fraktil-modellerne beregnes uafhængigt af hinanden. Derfor gives, i de senere afsnit, en redegørelse for, hvordan ovenstående optimeringsproblem kan udvides, så krydsninger undgås.

Hvis man vil beregne fraktilerne enkeltvist og uafhængigt af hinanden, benyttes altså modellen, (5.6), hvor

$$\mathbf{c} = \begin{bmatrix} \mathbf{0}_K \\ \tau\mathbf{e} \\ (1-\tau)\mathbf{e} \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} \boldsymbol{\beta} \\ \mathbf{r}^+ \\ \mathbf{r}^- \end{bmatrix}, \quad \mathbf{A} = [\mathbf{X}, \mathbf{I}, -\mathbf{I}].$$

Her er $\mathbf{X}$ og $\boldsymbol{\beta}$ hhv. designmatricen og koefficienterne tilhørende en enkelt fraktil-model. Denne metode til at beregne fraktil-kurver kaldes fremover for *Cross*, og implementeres i funktionen `CrossQuant`. Inden der gives en redegørelse for implementeringen af `CrossQuant`, opsummeres fremgangsmåden i Simplex-algoritmen i det følgende afsnit.

[5] Der eksisterer $2K$ mulige retninger, hvis der, som i tilfældet her, er at tale om fraktil regression, hvor der *ikke* tages højde for krydsninger.

5.2 Implementering af Simplex-metoden

Givet et datasæt, (x_i, y_i), hvor $i = 1, \ldots, N$, beregnes fraktil-modellerne ved at løse et minimeringsproblem på formen (5.6).

Nøgleværdierne i Simplex-algoritmen, (4.23), (4.24) og (4.25), er:

$$\mathbf{d} = \mathbf{c}_{\mathcal{C}} - \mathbf{C}^T\mathbf{B}^{-T}\mathbf{c}_{\mathcal{B}} \ , \qquad \mathbf{h} = \mathbf{B}^{-1}\mathbf{C}_{:,s} \ , \qquad \alpha = min\{\sigma_1, \ldots, \sigma_m\} \tag{5.28}$$

Elementerne i vektoren, $\mathbf{d}$, repræsenterer de relative ændringer i objektfunktionen fra hjørnepunktet, $\mathbf{x}^{(k)}$, til hvert af de naboliggende hjørnepunkter. Vektoren, $\mathbf{h}$, angiver retningen hvormed $\mathbf{x}_{\mathcal{B}}$ ændres i, og α angiver skridtlængden.

Simplex-metoden

Bemærk at Simplex-algoritmen *altid* indledes i et hjørnepunkt. Antag nu, at vi befinder os i et sådan punkt.

1. Beregn residualerne, $\mathbf{r}$, indeks-mængderne, h og $\overline{h}$ og konstruer herefter $\mathbf{x}_{\mathcal{B}}$
2. Beregn matricerne $\mathbf{B}^{-1}$ og $\mathbf{C}$ og herefter vektoren $\mathbf{d}$.
   - Hvis $\mathbf{d} \geq \mathbf{0} \Rightarrow$ *Stop*, idet $\mathbf{x}$ er den optimale løsning.
   - Ellers vælges en indeksværdi, s, således at $d_s < 0$.
3. Beregn retningen, $\mathbf{h} = \mathbf{B}^{-1}\mathbf{C}_{:,s}$.
   - Hvis $\mathbf{h} \leq \mathbf{0} \Rightarrow$ *Stop*, idet problemet er ubegrænset.
4. Beregn, via $\mathbf{h}$, skridtlængderne, σ, således at $\alpha = \sigma_q$.
5. Ombyt element $\overline{h}_q$ med h_s.
6. Opdater $\mathbf{x}_{\mathcal{B}}$ ved $\mathbf{x}_{\mathcal{B}} - \alpha\mathbf{h}$ og sæt $(x_{\mathcal{B}})_q = \alpha$.

   Gentag skridtene 2-6 indtil $\mathbf{d} \geq \mathbf{0}$.

5.3 Algoritme 1 - CrossQuant

Implementeringen af funktionen `CrossQuant` er foretaget med udgangspunkt i dele af implementering i [5].

I dette afsnit gives, på baggrund af de enkelte Simplex-skridt beskrevet i foregående afsnit, en redegørelse for opbygningen af `CrossQuant`, som er implementeringen af metoden *Cross*. Under redegørelsen inddrages dele af den skrevne `Matlab`-kode.

`CrossQuant` har følgende funktionskald:

```
[Beta, T] = CrossQuant(X, beta0, y, tau)
```

Her ses en kort præsentation af input samt output-argumenterne for funktionen:

Input argumenter:

`X` : Designmatrix for datapunkter.

`beta0` : En vektor med startgættet.

`y` : En datavektor for responsvariablen.

`tau` : En vektor bestående af værdierne $\tau_i \in [0,1]$.

Output argumenter:

`Beta` : En matrix, hvis i'te søjle angiver $\hat{\boldsymbol{\beta}}(\tau_i)$.

`T` : En vektor, hvis i'te element indeholder tabet for den τ_i'te fraktil.

Funktionen, `CrossQuant`, bestemmer sættet af koefficienter tilhørende fraktil-modellerne enkeltvist. Givet startgættet $\hat{\boldsymbol{\beta}}_0$, bestemmes koefficienterne $\hat{\boldsymbol{\beta}}(\tau_i)$ for alle τ_i i inputvektoren, τ, hvor τ angiver fraktilerne, for hvilke koefficienterne ønskes beregnet. Herefter returneres en matrix, `Beta`, hvis i'te søjle angiver koefficientvektoren $\hat{\boldsymbol{\beta}}(\tau_i)$, samt en vektor, `T`, indeholdende tabet for de enkelte fraktiler.

I modsætning til funktionerne `NonCrossSingle` samt `NonCrossHuge`, som beskrives i afsnit 6.3 og 7.2, tager `CrossQuant` ikke højde for eventuelle krydsninger af fraktilerne under beregningen af $\hat{\boldsymbol{\beta}}(\tau)$. Sidstnævnte funktion har dannet grobund for udviklingen af `NonCrossSingle` og `NonCrossHuge`. Den fulde kode ses i afsnit A.3.

Implementering af CrossQuant

Den beskrevne kode nedenfor indgår i en `for`-løkke, som gentages fra `j = 1` til antallet af fraktiler (`length(tau)`), som ønskes bestemt.

Skridt 1

Givet de observerede data, `y`, designmatricen, `X` samt et startgæt, `beta0`, beregnes residualerne, `r`, samt indeksmængderne `h` og `non_h` (h og $\overline{h}$), som anvist i Box 1.

Box 1

```
15    % Beregner residualerne.
16    r = y - X*beta0;
17
18    % Indeksmængden, 'h', bestemmes via funktionen 'find_h'.
19    [h, non_h, P] = find_h(X, r, beta0, N);
20
21    % Samler xB, som indeholder beta og slackvariablene.
22    xB = [beta0; abs(r(non_h))];
```

Funktionen `find_h` (afsnit A.6) beregner, som navnet antyder det, indeks-mængden `h` ved først at bestemme de indeksværdier, `I`, for hvilke der gælder at `abs(r(I))< tol`. Tolerancen, `tol = 1e-06`, er fundet ved "*trial and error*" og sættes til en værdi, som aftvinger mindst `K` elementer i `I`. Herefter udvælges rækkerne `X(I,:)`, og af disse findes `K` lineært uafhængige rækker. Indeks-værdierne for disse rækker udgør `h`, og via `h` bestemmes `non_h`.

Endvidere samles vektoren, `xB` ($\mathbf{x}_\mathcal{B}$), bestående af stargættet, `beta0`, samt de absolutte residualer, `abs(r(non_h))`, som her repræsenterer slackvariablene, der er i basis.

Skridt 2

Via inputargumentet, `tau` (τ), bestemmes vektoren, `rho` (ρ), som repræsenterer den asymmetriske tabsfunktions - Box 2.

Herefter konstrueres matricerne `B_inv` ($\mathbf{B}^{-1}$) og `C` ($\mathbf{C}$), hvor man under beregningen af førstnævnte matrix, udnytter den særlige struktur i $\mathbf{B}$, som ses i afsnit 5.1.

Box 2

```
24    % Beregner koefficienterne i objektfunktionen.
25    rho = P;
26    rho(rho==-1) = 1-tau(j);
27    rho(rho==1) = tau(j);
28
```

```
29    % Beregner B_inv
30    P = diag(P);
31    B11 = eye(size(X(h,:)))/X(h,:);
32    B12 = zeros(K,mX-K);
33    B21 = -P*X(non_h,:)/X(h,:);
34    B22 = P;
35    Binv = [B11 B12; B21 B22];
36    P = diag(P);
37
38    % Beregner 'C'.
39    C = [eye(K) -eye(K); zeros(mX-K,2*K)];
```

Herefter beregnes vektoren, `d`, hvis elementerne repræsenterer de relative ændringer i objektfunktionen fra $\mathbf{x}^{(K)}$, og der undersøges herefter, via en `while`-løkke, om `d` $\geq$ `0` - se Box 3. Hvis dette er tilfældet afbrydes funktionen, hvis ikke, udvælges et negativt element, `d(s)`, i `d` samt dets tilhørende indeks, `s`. Én måde at vælge `s` og `d(s)` er, at udvælge det mest negative element i `d`, som det er valgt at gøre her. Selvom `d(s)`, i dette tilfælde, angiver det største relative fald i objektfunktionen, afhænger det *egentlige* fald i objektfunktionen af både `d(s)` *og* skridtlængden, `alpha` (α).

Box 3

```
41    % Beregner 'd', som angiver det relative fald for alle retninger.
42    cC = [tau(j)*ones(K,1); (1-tau(j))*ones(K,1)];
43    cB = [zeros(K,1); rho];
44    d = cC - C'*Binv'*cB;
|
54        % Finder 's', dvs. indeksnummeret for det mest negative element i
55        % 'd', som angiver det relative fald i objektfunktionen i den på-
56        % gældende retning.
57        s = index1(d == min(d));
58        s = s(1);
```

I funktionerne `NonCrossSingle` samt `NonCrossHuge` benyttes en anden måde til at udvælge, `s`, som forklares i de pågældende afsnit.

Skridt 3

Via indekset, `s`, bestemmes retningen, `hdir`, og der undersøges om `hdir` $\leq$ `0`. Hvis dette er tilfældet afbrydes funktionen, idet problemet er ubegrænset.

Box 4

```
60        % Beregner den aftagende retning for den valgte 's'.
61        hdir = Binv*C(:,s);
62        if (hdir <= 0)
63            break;
64        end
```

Skridt 4

Når retningen er bestemt, vælges en skridtlængde, `alpha`, således at ingen elementer i `xB`, udover `beta`, kan blive negative i retningen, `hdir`. Som tidligere nævnt består `xB` af `beta` og de absolutte residualer, hvor residualernes egentlige fortegn angives ved matricen, `P`. Hvis `alpha` vælges for stort, vil en eller flere slackvariable blive negative, hvilket strider imod de *simple betingelser* i Simplex-modellen (5.6). Med andre ord skal ethvert residual, som skifter fortegn først ud af basis og dernæst ind i basis igen, hvorved fortegnet ændres i `P`, således at slackvariablen i `xB` fortsat er positiv.

For at bestemme `alpha` beregnes først `sigma(i)` (σ_i) for alle `hdir(i)` - se Box 5. Skridtlængden vælges til `alpha = min(sigma)`, og det tilsvarende indeks, `q`, gemmes.

Box 5

```
66        % Beregner de mulige skridtlængder i retningen 'hdir'.
67        sigma = zeros(length(hdir)-K,1);
68        for i = K+1:length(hdir)
68            if (hdir(i) > 10^(-6))
70                sigma(i-K) = xB(i)/hdir(i);
71            else
72                sigma(i-K) = inf;
73            end
74        end
75
76        % Skridtlængden, 'alpha', bestemmes.
77        alpha = min(sigma);
78
79        % 'xB(q)' er elementet i 'xB', som bliver nul ved skridtlængden
80        % 'alpha' i retningen 'hdir'.
81        q = index2(sigma==min(sigma));
82        q = q(1);
```

Skridt 5

Elementet `non_h(q)` byttes om med `h(s)`, som markerer ankomsten til et nyt hjørnepunkt.

Fortegnet, for den nytilkomne slackvariabel i `xB`, tilpasses i `P` via indekset, `q` - se Box 6.

Box 6

```
84        % Skifter en slackvariabel ind og ud af basis.
85        non_h_ud = non_h(q);
86        if s>K
87            h_ud = h(s-K);
88            non_h(q) = h_ud;
89            h(s-K) = non_h_ud;
90        else
```

```
 91              h_ud = h(s);
 92              non_h(q) = h_ud;
 93              h(s) = non_h_ud;
 94          end
 95
 96          % Ændring af residualets fortegn.
 97          if s>K
 98              P(q) = -1;
 99          else
100              P(q) = 1;
101          end
```

Skridt 6

Nu ændres `xB` i retningen, `hdir`, med skridtlængden, `alpha`.

Box 7

```
103          % Opdaterer 'xB'.
104          xB = xB - alpha*hdir; xB(q+K) = alpha;
```

Simplex-skridtene 2−6 repeteres indtil `d` $\geq$ `0`. I dette tilfælde er $\mathbf{x}^{(K)}$ den optimale løsning.

Som nævnt ovenfor, kan der ved anvendelse af den beskrevne funktion, `CrossQuant`, opstå krydsninger i mellem fraktilerne. I det følgende afsnit beskrives en udvidet model, som tager højde for fraktil-krydsninger ved indførelse af såkaldte non-crossing-betingelser i modellen.

6 Modeller for ikke-krydsende fraktiler

Det følgende afsnit er skrevet med inspiration fra [6].

I dette afsnit illustreres det, hvordan metoden, *Cross*, kan udvides, for herved at undgå de førnævnte fraktil-krydsninger.

Udfaldsrummet for de forklarende variable angives nu ved $\mathcal{P}$. Hvis $\mathcal{P}$ er ubegrænset, kan fraktil-krydsninger i udfaldsrummet kun undgås, hvis fraktilerne er parallelle. Med de vindkraftdata, som behandles i projektet, er det dog ikke rimeligt at antage, at fraktilerne er parallelle. Heldigvis *er* $\mathcal{P}$ begrænset[6], hvorfor det er tilstrækkeligt at kræve, at fraktilerne-krydsninger ikke må fremkomme i selve udfaldsrummet [6].
Idet udfaldsrummet er kontinuert, er det ikke muligt, at tjekke efter fraktil-krydsninger i ethvert punkt tilhørende $\mathcal{P}$. I stedet udvælges nogle punkter fra $\mathcal{P}$, hvor der undersøges, om fraktil-krydsninger finder sted. Vælges disse punkter til at ligge relativt tæt, mindskes sandsynligheden for krydsninger væsentligt [6]. De udvalgte punkter repræsenteres nu i matricen $\mathbf{X}_{nc}$, hvor rækkerne i $\mathbf{X}_{nc}$ er værdierne af basisfunktionerne i de pågældende punkter.

Der findes forskellige metoder til at undgå fraktil-krydsninger, og de benævnes alle som *non-crossing*-metoder. I den mest simple metode beregnes fraktilerne enkeltvist med det krav, at ingen af disse fraktiler krydser hinanden. Metoden beskrives i det følgende afsnit.

6.1 En simpel non-crossing model

I denne metode beregnes først en fraktil uden at angive nogle non-crossing-betingelser. Fraktilen kan beregnes via R, og det vil typisk være 50%-fraktilen, man vælger at begynde med, hvilket begrundes senere. Ud fra denne fraktil beregnes en nabofraktil (enten fraktilen lige over eller lige under den første fraktil) med den betingelse, at denne fraktil ikke krydser den forrige. Nabo-fraktilen, der ønskes beregnet, betegnes nu som τ_2, mens fraktilen, som allerede er beregnet, betegnes som τ_1. Når den ønskede nabofraktil således er bestemt angives *denne* ved τ_1, og bliver nu udgangspunktet i beregningen af den næste fraktil, som derved kaldes τ_2. Denne procedure fortsættes, indtil den øverste eller nederste fraktil bestemt; afhængigt af om man startede med at bevæge sig opad eller nedad. Herefter tages der igen udgangspunkt i begyndelses-fraktilen, men denne gang bestemmes fraktilerne i den modsatte vej end før, således at alle de ønskede fraktiler i den sidste ende bestemmes. Denne metode kaldes fremover for NC_{single}.

Der ønskes med denne metode at kunne beregne $\boldsymbol{\beta}(\tau_2)$, således at $Q(\mathbf{x};\tau_2)$ ikke krydser $Q(\mathbf{x};\tau_1)$. Antages det nu, at $\tau_2 > \tau_1$, fås begrænsningerne:

$$\begin{aligned} \mathbf{X}_{nc}\boldsymbol{\beta}(\tau_2) &\geq \mathbf{X}_{nc}\boldsymbol{\beta}(\tau_1) \Leftrightarrow & (6.1)\\ \mathbf{X}_{nc}\boldsymbol{\beta}(\tau_2) - \mathbf{X}_{nc}(\boldsymbol{\beta}(\tau_2) - \boldsymbol{\beta}(\tau_1)) &= \mathbf{X}_{nc}\boldsymbol{\beta}(\tau_1) \Leftrightarrow & (6.2)\\ \mathbf{X}_{nc}\boldsymbol{\beta}(\tau_2) + \mathbf{X}_{nc}(\boldsymbol{\beta}(\tau_1) - \boldsymbol{\beta}(\tau_2)) &= \mathbf{X}_{nc}\boldsymbol{\beta}(\tau_1). & (6.3) \end{aligned}$$

[6] Udfaldsrummet for X_{pred} er begrænset, idet $X_{pred} \in [0,1]$.

Antages det derimod, at $\tau_2 < \tau_1$, fås begrænsningerne:

$$\begin{aligned} \mathbf{X}_{nc}\boldsymbol{\beta}(\tau_2) &\leq \mathbf{X}_{nc}\boldsymbol{\beta}(\tau_1) \Leftrightarrow & (6.4)\\ \mathbf{X}_{nc}\boldsymbol{\beta}(\tau_2) + \mathbf{X}_{nc}(\boldsymbol{\beta}(\tau_1) - \boldsymbol{\beta}(\tau_2)) &= \mathbf{X}_{nc}\boldsymbol{\beta}(\tau_1). & (6.5) \end{aligned}$$

Herved fås følgende generelle begrænsninger:

$$\mathbf{X}_{nc}\boldsymbol{\beta}(\tau_2) + \text{sign}(\tau_1 - \tau_2) \cdot |\mathbf{X}_{nc}(\boldsymbol{\beta}(\tau_1) - \boldsymbol{\beta}(\tau_2))| = \mathbf{X}_{nc}\boldsymbol{\beta}(\tau_1) \tag{6.6}$$

Ved indførelse af disse nye begrænsninger i modellen (5.6), ændres $\mathbf{A}$ nu til:

$$\mathbf{A} = \begin{bmatrix} \mathbf{X} & \mathbf{I} & -\mathbf{I} & \mathbf{0} \\ \mathbf{X}_{nc} & \mathbf{0} & \mathbf{0} & \text{sign}(\tau_1 - \tau_2)\mathbf{I} \end{bmatrix}. \tag{6.7}$$

Endvidere udvides $\mathbf{b}$ med $\mathbf{y}_{nc} = \mathbf{X}_{nc}\boldsymbol{\beta}(\tau_1)$, og $\mathbf{x}$ udvides med $\mathbf{r}_{nc} = |\mathbf{X}_{nc}(\boldsymbol{\beta}(\tau_1) - \boldsymbol{\beta}(\tau_2))|$. Der gælder altså nu at:

$$\mathbf{Ax} = \begin{bmatrix} \mathbf{X} & \mathbf{I} & -\mathbf{I} & \mathbf{0} \\ \mathbf{X}_{nc} & \mathbf{0} & \mathbf{0} & \text{sign}(\tau_\mathbf{1} - \tau_\mathbf{2})\mathbf{I} \end{bmatrix} \cdot \begin{bmatrix} \boldsymbol{\beta}(\tau_2) \\ \mathbf{r}^+ \\ \mathbf{r}^- \\ |\mathbf{X}_{nc}(\boldsymbol{\beta}(\tau_1) - \boldsymbol{\beta}(\tau_2))| \end{bmatrix} = \begin{bmatrix} \mathbf{y} \\ \mathbf{X}_{nc}\boldsymbol{\beta}(\tau_1) \end{bmatrix}. \tag{6.8}$$

Derudover ændres $\mathbf{c}$ til:

$$\mathbf{c} = \begin{bmatrix} \mathbf{0}_K \\ \tau_2\mathbf{e} \\ (1-\tau_2)\mathbf{e} \\ \mathbf{0}_{N_{nc}} \end{bmatrix}, \tag{6.9}$$

hvor N_{nc} er antallet af punkter, hvori der tjekkes for krydsninger.

Udvidelsen af $\mathbf{A}$ leder naturligvis til en tilsvarende udvidelse i $\mathbf{B}$, og ved at konstruere matricerne $\mathbf{X_s}$ og $\mathbf{P_s}$, kan $\mathbf{B}$ antage formen (5.13):

$$\mathbf{B} = \begin{bmatrix} \mathbf{X}_s(h) & \mathbf{0} \\ \mathbf{X}_s(\overline{h}) & \mathbf{P_s} \end{bmatrix}, \tag{6.10}$$

hvor

$$\mathbf{X_s} = \begin{bmatrix} \mathbf{X} \\ \mathbf{X}_{nc} \end{bmatrix}, \ \mathbf{r_s} = \begin{bmatrix} \mathbf{r} \\ \mathbf{r}_{nc} \end{bmatrix}, \tag{6.11}$$

og hvor $\mathbf{P_s}$ er en diagonal-matrix med $\text{sign}(\mathbf{r_s}(\overline{h}))$.

Bemærk, at h stadig tilhører $\mathbb{N}^K$, men at index-værdierne nu kommer fra mængden $\{1,\ldots,N+N_{nc}\}$, og at $\overline{h}$ derved indeholder index-værdierne $\{1,\ldots,N+N_{nc}\}\setminus\{h\}$.

Via omskrivningen af $\mathbf{B}$ kan $\mathbf{B}^{-1}$ beregnes via metoden beskrevet i afsnit 5.1.

I vores tilfælde, med vindkraftdataene, er udfaldsrummet for responsvariablen begrænset, og vi lader nu $[y_{\min}, y_{\text{maks}}]$ definerer det afgrænsede interval for denne variabel. Idet responsvariablen herved ikke kan antage værdier uden for intervallet, bør det samme gælde for de estimerede fraktiler. Dette giver anledning til indførelsen af grænsebetingelser for fraktilerne.

Indførelse af grænsebetingelser

Lad $[y_{\min}, y_{\text{maks}}]$ definere intervallet for responsvariablen. De estimerede fraktil-kurver bør, som sagt, ikke bevæge sig uden for dette interval, hvorfor der nu indføres følgende grænsebetingelser for fraktilerne:

$$\mathbf{X}_{nc}\boldsymbol{\beta}_j \leq y_{\text{maks}}\mathbf{e}, \tag{6.12}$$

$$\mathbf{X}_{nc}\boldsymbol{\beta}_j \geq y_{\min}\mathbf{e}, \quad j=1,\ldots,l. \tag{6.13}$$

hvor l angiver antallet af fraktiler-kurver, som ønskes bestemt. Tilføjes disse betingelser til systemet fra før, fås:

$$\mathbf{A} = \begin{bmatrix} \mathbf{X} & \mathbf{I} & -\mathbf{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{X}_{nc} & \mathbf{0} & \mathbf{0} & \text{sign}(\tau_1-\tau_2)\mathbf{I} & \mathbf{0} & \mathbf{0} \\ \mathbf{X}_{nc} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{I} & \mathbf{0} \\ \mathbf{X}_{nc} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & -\mathbf{I} \end{bmatrix}. \tag{6.14}$$

Derudover udvides $\mathbf{x}$ med $\mathbf{r}_{bc}$, og $\mathbf{b}$ udvides med $\mathbf{y}_{bc}$, således at:

$$\mathbf{x} = \begin{bmatrix} \boldsymbol{\beta}(\tau_2) \\ \mathbf{r}^+ \\ \mathbf{r}^- \\ |\mathbf{X}_{nc}(\boldsymbol{\beta}(\tau_1)-\boldsymbol{\beta}(\tau_2))| \\ |y_{\text{maks}}\mathbf{e}-\mathbf{X}_{nc}\boldsymbol{\beta}(\tau_2)| \\ |y_{\min}\mathbf{e}-\mathbf{X}_{nc}\boldsymbol{\beta}(\tau_2)| \end{bmatrix}, \mathbf{b} = \begin{bmatrix} \mathbf{y} \\ \mathbf{X}_{nc}\boldsymbol{\beta}(\tau_1) \\ \mathbf{y}_{\text{maks}}\mathbf{e}_{N_{nc}} \\ \mathbf{y}_{\min}\mathbf{e}_{N_{nc}} \end{bmatrix} \tag{6.15}$$

hvilket giver følgende ændring i $\mathbf{c}$:

$$\mathbf{c} = \begin{bmatrix} \mathbf{0}_K \\ \tau_2\mathbf{e} \\ (1-\tau_2)\mathbf{e} \\ \mathbf{0}_{3N_{nc}} \end{bmatrix}. \tag{6.16}$$

Matricen $\mathbf{B}$ kan igen bringes på formen (5.13):

$$\mathbf{B} = \begin{bmatrix} \mathbf{X_s}(h) & \mathbf{0} \\ \mathbf{X_s}(\overline{h}) & \mathbf{P_s} \end{bmatrix}, \tag{6.17}$$

hvor

$$\mathbf{X_s} = \begin{bmatrix} \mathbf{X} \\ \mathbf{X}_{nc} \\ \mathbf{X}_{nc} \\ \mathbf{X}_{nc} \end{bmatrix}, \quad \mathbf{r_s} = \begin{bmatrix} \mathbf{r} \\ \mathbf{r}_{nc} \\ \mathbf{r}_{bc} \end{bmatrix}, \tag{6.18}$$

og hvor $\mathbf{P_s}$ er diagonal-matricen med $\text{sign}(\mathbf{r_s}(\overline{h}))$.

Strukturen i $\mathbf{B}$ leder til følgende form af $\mathbf{x}_\mathcal{C}$ og $\mathbf{C}$.

$$\mathbf{x}_\mathcal{C} = \begin{bmatrix} \mathbf{r_s}(h_r)^+ \\ \mathbf{r_s}(h_r)^- \\ \mathbf{r_s}(h_{nc}) \\ \mathbf{r_s}(h_{bc,max}) \\ \mathbf{r_s}(h_{bc,min}) \end{bmatrix}, \quad \mathbf{C} = \begin{bmatrix} \mathbf{I} & -\mathbf{I} & 0 & 0 & 0 \\ 0 & 0 & sign(\tau_1 - \tau_2)\mathbf{I} & 0 & 0 \\ 0 & 0 & 0 & \mathbf{I} & 0 \\ 0 & 0 & 0 & 0 & -\mathbf{I} \end{bmatrix}, \tag{6.19}$$

hvor h_r, h_{nc}, $h_{bc,max}$, $h_{bc,max} \in h$.

Fordelen ved denne non-crossing-metode, hvad enten det er med eller uden grænsebetingelser, er, at fraktil-krydsninger undgås, og dermed giver det samlede resultat mening rent statistisk. Modellen har dog den ulempe, at resultatet afhænger af, hvilken fraktil der benyttes som begyndelses-fraktil. Årsagen til dette er, at begyndelses-fraktilen herved er fastlagt, hvorved de øvrige fraktiler skal tilpasse sig denne. Af denne grund vælges ofte 50%−fraktilen, som begyndelses-fraktil, idet den typisk vil være forholdsvist præcis grundet mængden af omkringliggende data, og man håber derved på at mindske effekten fra valg af begyndelses-fraktil [9]. I [9] argumenteres der også for brugen af 50%−fraktilen som begyndelses-fraktil. Det skal dog bemærkes, at startgættet skal være en mulig løsning, dvs. hvis 50%−fraktilen, beregnet i R, overskrider de eventuelle grænsebetingelser, er man nødsaget til at finde en anden begyndelsesfraktil.

I [9] foreslås det desuden, at man, efter at have beregnet alle fraktil-kurverne via ovenstående metode, fortsætter med at estimere kurverne, ved at benytte den nederste og den øverste fraktil, man lige har beregnet, til at estimere et nyt sæt af fraktiler. Princippet bag dette er, at man håber på at mindske den dårlige indflydelse, som begyndelses-fraktilen kan have på resten af fraktil-kurverne. Denne metode beskrives i det følgende afsnit.

6.2 Udvidet fremgangsmåde for den simple non-crossing-model

Det antages i dette delafsnit, at metoden NC_{single} allerede er benyttet, således at man har et estimat af den nederste og øverste fraktil-kurve. Deres tilhørende fraktil-koefficienter kaldes hhv. $\boldsymbol{\beta}_{\text{nederst}}$ og $\boldsymbol{\beta}_{\text{øverst}}$. Først benyttes $\boldsymbol{\beta}_{\text{nederst}}$ som begyndelses-fraktil, således at de øvrige fraktiler beregnes i forhold til denne. Dette nye estimat af fraktil-koefficienterne angives ved $\boldsymbol{\beta}_{op_1}, \ldots, \boldsymbol{\beta}_{op_K}$. Derefter benyttes $\boldsymbol{\beta}_{\text{øverst}}$ som begyndelses-fraktil, og de øvrige fraktiler beregnes med udgangspunkt i denne; disse nye fraktil-koefficienter angives ved $\boldsymbol{\beta}_{ned_1}, \ldots, \boldsymbol{\beta}_{ned_K}$.
Det endelige estimat af $\boldsymbol{\beta}_i$ er således gennemsnittet af $\boldsymbol{\beta}_{op_i}$ og $\boldsymbol{\beta}_{ned_i}$. Denne metode kaldes

fremover for $NC_{singleON}$, hvor NO er en forkortelse af "Nedefra" og "Oppefra", og ved benyttelsen af denne metode, håber man på at mindske indflydelsen fra valget af startfraktil yderligere, så man derved ender op med et bedre estimat. Om dette er tilfældet, testes på vores data i afsnit 10.2.

Heldigvis kan man udvide modellen yderligere, således at det endelige fraktil-estimat er fuldstændigt uafhængigt af noget begyndelses-valg. Denne model beregner samtlige fraktiler på én gang. Inden denne metode beskrives, gives her en redegørelse for implementeringen af `NonCrossSingle`, som ligger til grund for metoderne NC_{single} og $NC_{singleNO}$.

6.3 Algoritme 2 - NonCrossSingle

I dette afsnit gives en redegørelse for opbygningen af `NonCrossSingle` på baggrund af de enkelte Simplex-skridt beskrevet i afsnit 5.2. Under redegørelsen inddrages dele af den skrevne `Matlab`-kode.

Følgende funktion er egenhændigt udviklet på baggrund af modellen opstillet i afsnit 6 samt af funktionen `CrossQuant`.

`NonCrossSingle` har følgende funktionskald:

```
[Beta,T,H] = NonCrossSingle(X,beta,y,Xnc,tau,lb,ub,h0);
```

Her ses en kort præsentation af input samt output-argumenterne for funktionen:

Input argumenter:

`X` : Designmatrix for datapunkter.

`beta0` : En vektor med koefficienterne for begyndelses-fraktilen.

`y` : En datavektor for responsvariablen.

`Xnc` : Designmatrix for udvalgte kontrolpunkter.

`tau` : En vektor bestående af værdier $\tau \in [0,1]$, hvor der skal gælde, at $\tau_1 \leq \tau_2 \leq, \ldots, \leq \tau_n$, eller $\tau_1 \geq \tau_2 \geq, \ldots, \geq \tau_n$

`lb` : Nedre grænse.

`ub` : Øvre grænse.

`h0` : Indeks-mængden, h, for begyndelses-fraktilen. Dette input kan udelades.

Output argumenter:

`Beta` : En matrix, hvis i'te søjle angiver $\hat{\boldsymbol{\beta}}(\tau_i)$.

`T` : En vektor, hvis i'te element indeholder tabet for den τ_i'te fraktil.

`H` : En matrix bestående af de endelig indeks-mængder h for hver fraktil, hvor den i'te søjle i H angiver indeks-mængden, h, for den τ_i'te fraktil.

Funktionen `NonCrossSingle` er en udvidelse af `CrossQuant` og beregner koefficienterne

tilhørende fraktil-kurverne enkeltvis under den forudsætning, at disse ikke krydser. Inputvektoren, $\boldsymbol{\tau}$, angiver fraktil-kurverne, for hvilke koefficienterne ønskes beregnet.

Funktionen bestemmer, ud fra en given start $\hat{\boldsymbol{\beta}}_0(\tau_1)$, koefficienterne $\hat{\boldsymbol{\beta}}(\tau_2)$, hvorefter $\hat{\boldsymbol{\beta}}(\tau_2)$ benyttes som startgæt til beregning af $\hat{\boldsymbol{\beta}}(\tau_3)$. Dette gentages for alle $\tau_i \in \boldsymbol{\tau}$, hvor der skal gælde, at $\tau_1 < \tau_2 <, \ldots, \tau_n$ eller $\tau_1 > \tau_2 >, \ldots, > \tau_n$. Herefter returneres en matrix, `Beta`, hvis i'te søjle angiver koefficientvektoren $\hat{\boldsymbol{\beta}}(\tau_i)$, en vektor, `T`, indeholdende tabet for de enkelte fraktiler, samt en matrix, `H`, bestående af de endelige indeks-mængder, h, for hver af de beregnede fraktiler.

Implementering af NonCrossSingle

Skridt 1

Givet de observerede data, `y`, designmatricen, `X`, samt koefficienterne for begyndelsesfraktilen, `beta0`, beregnes residualerne, `r`. Herefter sammensættes matricen, `XS` (X_s), af den sædvanlige designmatrix, `X` samt af tre identiske matricer, `Xnc`, som her agerer kontrolpunkter for eventuelle krydsninger imellem fraktilerne og for eventuelle krydsninger imellem en fraktil og den øvre eller den nedre grænse, $y_{max} = ub$ og $y_{min} = lb$ - se Box 1.

Vektoren, `rS` (r_s), konstrueres herefter, hvor

- `r` angiver afvigelsen mellem fraktil-kurven og observationerne, y.
- `zeros(Nnc,1)` angiver afvigelsen mellem fraktil-kurven, $Q(\tau_i; \mathbf{x})$, og $Q(\tau_{i+1}; \mathbf{x})$, som indledningsvist sættes til nul, idet det antages at $Q(\tau_{i+1}; \mathbf{x})$ udspringer af $Q(\tau_i; \mathbf{x})$, og derfor er sammenfaldende til at begynde med.
- `ub-Xnc*beta` er afvigelsen mellem fraktil-kurven og den øvre grænse.
- `lb-Xnc*beta` er afvigelsen mellem fraktil-kurven og den nedre grænse.

Herved antager både `XS` og `rS` formen (6.18). Det pågældende stykke `Matlab`-kode ses af Box 1.

Box 1

```
10 % Beregner residualerne.
11 r = y - X*beta0;
12
13 % Samler den samlede designmatrix for forudsigelsen og kontrolpunkterne.
14 XS = [X; Xnc; Xnc; Xnc];
15
16 % Samler residualerne i vektoren 'rS':
17 rS = [r ; zeros(Nnc,1); ub-Xnc*beta0; lb-Xnc*beta0];
18 %    1) 'r' er residualerne mellem den 'tau(i)'te-fraktil og 'y'.
19 %    2) 'zeros(Nnc,1)' er residualerne mellem den 'tau(i)'te-fraktil og
```

```
20 %         den 'tau(i+1)'te-fraktil. Residualerne er '0', da vi antager, at
21 %         disse fraktiler indledningsvist er sammenfaldende.
22 %     3) 'ub-Xnc*beta' er residualerne mellem den 'tau(i)'te-fraktil og den
23 %        øvre grænse, y = ub.
24 %     4) 'lb-Xnc*beta' er residualerne mellem den 'tau(i)'te-fraktil og den
25 %        nedre grænse, y = lb.
```

Herefter beregnes indeks-mængderne `h` og `non_h` (h og $\overline{h}$) - se Box 2.

Gives `h` som input i funktionen beregnes `non_h` via `h`, hvis ikke, benyttes funktionen `find_h`, som blev beskrevet i forrige afsnit, til beregningen af både `h` og `non_h`. Disse indeks-mængder knytter sig kun til residualerne, `r`, og som en konsekvens heraf konstrueres en vektor `non_h_S`, som, udover at indeholde `non_h`, også inddrager indeks-værdierne for de øvrige elementer i `rS`.

Endvidere samles vektoren, `xB` ($\mathbf{x}_{\mathcal{B}}$), bestående af koefficienterne for begyndelses-fraktilen, `beta0`, samt de absolutte residualer, `abs(rS(non_h_S))`, som her repræsenterer slack-variablene.

Box 2

```
27 % Bestemmer indeksmængderne 'h' og 'non_h'.
28 if nargin < 8
29     % Hvis 'h' ikke gives som input, beregnes 'h', 'P' og 'non_h' via
30     % funktionen 'find_h'.
31     [h, non_h, P] = find_h(X, r, beta0, N);
32
33     % Samlerne 'non_h'erne for både 'r' og de øvrige residualer i 'rS' i
34     % vektoren 'non_h_S'.
35     non_h_S = [non_h N+1:N+3*Nnc];
36 else
37     % Hvis 'h' gives som input, beregnes 'non_h' og 'P' via 'h'.
38     index1 = 1:N+3*Nnc;
39     h = h0;
40     % Finder hvilke indekser i mængden {1,..,N+3*Nnc}, som indgår i 'h'.
41     % Dem som ikke indgår i 'h' udgør 'non_h'.
42     Ih = ismember(index1, h);
43     non_h_S = index1(~Ih);
44     non_h = non_h_S(non_h_S<=N);
45     P = sign(r(non_h));
46 end
47
48 % Samler residual-vektoren, som repræsenterer slackvariablene.
49 xB = [beta0 ; abs(rS(non_h_S))];
50 tau1 = tau(1);   % 'tau'-værdien for den 'tau(i)'te-fraktil.
51 tau2 = tau(2);   % 'tau'-værdien for den 'tau(i+1)'te-fraktil.
```

Dernæst beregnes antallet af elementer i `h`, som hører til hhv. `r`, de givne non-crossing-betingelser og grænsebetingelserne. Tilsvarende optælling foretages for `non_h_S` - se Box 3.

Matricen `PS` er en diagonalmatrix bestående værdierne -1 og 1 afhængigt af fortegnene af elementerne i `rS`.

`B_inv` ($\mathbf{B}^{-1}$) beregnes som anvist nedenfor, hvor den særlige struktur i $\mathbf{B}$ udnyttes, se afsnit 5.1.

Box 3

```
53 % Tæller antallet af 'h', 'non_h', 'h_nc', 'non_h_nc', 'non_h_ncy1' og
54 % 'non_h_ncy0', for holde styr på hvor elementerne i hhv. 'h' og 'non_h'
55 % kommer fra; om det er fra 'r', non-crossing betingelser eller fra grænse
56 % betingelser.
57 ant_h = sum(h<=N);
58 ant_nh = N-ant_h;
59 ant_hnc = sum(h<=N+Nnc)-ant_h;
60 ant_nhnc = Nnc-ant_hnc;
61 ant_hncy1 = sum(h<=N+2*Nnc)-(ant_h+ant_hnc);
62 ant_nhncy1 = Nnc-ant_hncy1;
63 ant_hncy0 = length(h)-(ant_h+ant_hnc+ant_hncy1);
64 ant_nhncy0 = Nnc-ant_hncy0;
65
66 % Konstruerer en matrix 'PS', som repræsenterer fortegnene for residual-
67 % erne i 'rS'. 'PS' konstrueres først som en vektor med 1-taller og
68 % modificeres herefter.
69 PS = ones(ant_nh+ant_nhnc+ant_nhncy1+ant_nhncy0,1);
70 PS(1:ant_nh) = P;                                % Fortegn for 'r'.
71 PS(ant_nh+1:ant_nh+ant_nhnc) = sign(tau1-tau2);% Fortegn for 'Crossing'.
72 PS(ant_nh+ant_nhnc+ant_nhncy1+1:end) = -1;      % Fortegn for nedre grænse.
73 PS = diag(PS);
74
75 % Beregner B_inv
76 B11 = eye(size(XS(h,:)))/XS(h,:);
77 B21 = -PS*XS(non_h_S,:)/XS(h,:);
78 B12 = zeros(K,N+3*Nnc-K);
79 B22 = PS;
80 B_inv = [B11 B12 ; B21 B22];
```

Skridt 2

Den beskrevne kode ovenfor indgår i initialiseringen af funktionen og køres én gang for hvert funktionskald. De følgende stykker kode, som beskrives under de resterende Simplex-skridt, indgår i en `for`-løkke, som gentages fra `j = 1` til antallet af fraktiler (`length(tau)`), som ønskes bestemt.

I dette skridt beregnes vektoren, `rho`, som repræsenterer den asymmetriske tabsfunktion.
Endvidere bregnes matricen `C`, som grundet det store antal nul-elementer, konstrueres som en `mC` $\times$ `nC` sparse matrix angivet på *triplet*-form (mere herom i afsnit 7.1). Ved

triplet-form angives tre vektorer, `IC`, `JC` og `Cval`, hvor de to førstnævnte vektorer angiver række- og søjleplaceringen og den sidstnævnte angiver værdien af alle ikke-nul-elementer.

Box 4

```
104    % Beregner koefficienterne i objektfunktionen.
105    rho = diag(PS(1:ant_nh,1:ant_nh));
106    rho(rho==-1) = 1-tau2;
107    rho(rho==1) = tau2;
108
109    % Beregner herunder den sparse matrice 'C' angivet på triplet-form.
110
111    % Alle ikke-nul-elementer lagres i vektoren 'Cval'.
112    Cval = [repmat([1 -1], [1 ant_h])'; sign(tau1-tau2)*...
113        ones(ant_hnc,1); ones(ant_hncy1,1); -ones(ant_hncy0,1) ];
114
115    % Søjle-indekser.
116    JC = (1:2*ant_h+ant_hnc+ant_hncy1+ant_hncy0)';
117    JC(1:2:2*ant_h) = 1:ant_h;
118    JC(2:2:2*ant_h) = ant_h+1:2*ant_h;
119
120    % Række-indekser.
121    IC = repmat(1:ant_h, [2 1]);
122    IC = [IC(:); (ant_h+1:ant_h+ant_hnc+ant_hncy1+ant_hncy0)'];
123
124    % Størrelsen af 'C'.
125    mC = N+3*Nnc;
126    nC = 2*ant_h+ant_hnc+ant_hncy1+ant_hncy0;
127
128    % Konstruerer 'C'.
129    C = sparse(IC,JC, Cval, mC, nC);
```

Herefter beregnes vektoren, `d` (se Box 5), hvis elementerne repræsenterer de relative ændringer i objektfunktionen fra $\mathbf{x}^{(K)}$, og der undersøges herefter om `d` ≥ 0 via en `while`-løkke.

Er dette tilfældet, afbrydes funktionen, hvis ikke, udvælges et negativt element, `d(s)`, i `d` samt dets tilhørende indeks, `s`. Hvordan `s` og `d(s)` vælges, beskrives i det følgende.

Box 5

```
131    % Beregner 'd', som angiver det relative gain for alle retninger.
132    cC = [tau2*ones(ant_h,1) ; (1-tau2)*ones(ant_h,1) ; zeros(ant_hnc+...
133        ant_hncy1+ant_hncy0,1)];
134    cB = [zeros(K,1); rho; zeros(ant_nhnc + ant_nhncy1 + ant_nhncy0,1)];
135    g = B_inv'*cB;
136    d = cC - C'*g;
```

Skridt 3

I modsætning til `CrossQuant` udvælges her *alle* negative elementer i `d` samt deres tilhørende indeks-værdier, som lagres i vektorerne `ds` og `s` - se Box 6. Som nævnt i forrige afsnit angiver de negative elementer i `d` det relative fald i objektfunktionen for en given retning, hvorimod det *egentlige* fald i objektfunktionen *også* afhænger af skridtlængden, `alpha` (α), hvorfor disse værdier bestemmes for alle `s(l)`, for `l = 1,...,length(ds)`.

For at bestemme `alpha` beregnes først `sigma(i)` (σ_i) for alle `hdir(i)`. Skridtlængden vælges til `alpha = min(sigma)`, således at ingen slackvariable i `xB` bliver negative, og det tilsvarende indeks i `sigma`, `q`, gemmes. Til beregningen af `sigma` anvendes ikke en `for`-løkke, som det var tilfældet i `CrossQuant`; grundet `sigma`'s øgede længde, som en konsekvens af de indførte non-crossing- og grænsebetingelser, beregnes `sigma` via vektorer.

Ethvert sæt `alpha`, `q` og `s(l)` lagres i den i'te række i den preallokerede matrix `a_q_s` med henblik på senere brug.

Box 6

```
141         % Tæller antallet af aftagende retninger i 'd' og lagrer dem i en
142         % vektoren 'ds'. Deres tilhørende indeksværdier lagres i 's'.
143         ant_neg_d = sum(d<0);
144         [ds s]= sort(d);
145         ds = ds(1:ant_neg_d);
146         s = s(1:ant_neg_d);
147         a_q_s = zeros(ant_neg_d,3);
148         index3 = 1:ant_neg_d;
149
150         % For alle aftagende retninger, beregnes selve retningen, 'hdir'.
151         for l = 1:ant_neg_d
152             hdir = B_inv*C(:,s(l));
153
154             if sum(hdir > 0) > 0
155                 % Beregner 'alpha', 'q' og 's' for alle aftagende ret-
156                 % ninger. Disse værdier lagres i den pre-allokerede matrix
157                 % 'a_q_s'.
158                 sigma = zeros(length(hdir)-K,1);
159                 xB_down = xB(K+1:end);          % Slackvariablene i 'xB'.
160                 hdir_down = hdir(K+1:end);      % De tilsvarende i 'hdir'.
161                 sigma(hdir_down>tol) = xB_down(hdir_down>tol)./...
162                     hdir_down(hdir_down>tol);
163                 sigma(hdir_down <= tol) = inf;
164
165                 % Her gemmes 'alpha', 'q' og 's' i matricen 'a_q_s'.
166                 a_q_s(l,1) = min(sigma);                    % 'alpha'.
167                 q = index2(sigma==min(sigma)); q = q(1);
168                 a_q_s(l,2) = q;                             % 'q'
169                 a_q_s(l,3) = s(l);                          % 's'
170             else
171                 disp('Problemet er ubegrænset')
172             end
```

```
173         end
```

Skridt 4

For alle de potentielle skridtlængder i matricen `a_q_s`, bestemmes nu det *egentlige* fald i objektfunktionen. Hvis `a_q_s(:,1)= 0`, dvs. alle skridtlængder har værdien nul, vælges den første af de mulige skridtlængder, `a_q_s(1,1)`, samt dennes tilhørende `q = a_q_s(1,2)` og `s = a_q_s(1,3)`, idet disse værdier hører til den stejlest aftagende retning - se Box 7.
Hvis én eller flere af skridtlængderne er større end nul beregnes, via disse, det *egentlige* fald i objektfunktionen. Skridtlængden tilhørende det største *egentlige* fald vælges, samt dennes `s` og `q`.

Via disse tre værdier bestemmes `hdir` påny.

Box 7

```
161         if  sum(a_q_s(:,1) > 0) == 0
162             % Finder 's' og 'q', hvis alle alpha = 0.
163             alpha = a_q_s(1,1);
164             q = a_q_s(1,2);
165             s = a_q_s(1,3);
166         else
167             % Finder 's' og 'q', hvis ikke alle alpha = 0. Dette gøres ved
168             % at vælge det element i 'd', som giver det største egentlige
169             % fald i objektfunktionen. Det egentlige fald for alle alpha
170             % beregnes i 'gain'.
171             gain = ds.*a_q_s(:,1);
172             Igain = index3(gain == min(gain)); Igain = Igain(1);
173             alpha = a_q_s(Igain,1);
174             q = a_q_s(Igain,2);
175             s = a_q_s(Igain,3);
176         end
177
178         % Retningen, 'hdir', bestemmes for den valgte 's' og 'q'.
179         hdir = B_inv*C(:,s);
180
```

Skridt 5

Herefter ombyttes elementet `non_h_temp(q)` med `h_temp(s)`, hvor `non_h_temp` og `h_temp` er midlertidige variable, dog identiske med `non_h` og `h` (Box 8). Ombytningen foretages ikke direkte i de sidstnævnte, idet der undersøges om denne ombytning i fortsat opfylder at:

$$\texttt{rank(X(h\_temp,:))= K}, \tag{6.20}$$

dvs. om ranken af de K udvalgte rækker i X fortsat er K. Hvis kravet ikke er opfyldt, sættes d(s)= 1, og herved påtvinges valget af et nyt, negativt element i d.

Simplex-skridtene to til fire repeteres indtil rank(X(h_temp,:)= K eller d >= 0.

Hvis der derimod gælder at rank(X(h_temp,:))= K, sættes non_h = non_h_temp og h = h_temp svarende til at foretage en direkte ombytning af non_h(q) og h(s).

Box 8

```
195         % Skifter en slackvariabel ind og ud af basis. Ændringen foretages
196         % ikke direkte i 'h' og 'non_h', men i to midlertidige variable
197         % 'h_temp' og 'non_h_temp'.
198         non_h_temp = non_h_S;
199         h_temp = h;
200         non_h_ud = non_h_S(q);
201         if s>ant_h
202             h_ud = h(s-ant_h);
203             non_h_temp(q) = h_ud;
204             h_temp(s-ant_h) = non_h_ud;
205         else
206             h_ud = h(s);
207             non_h_temp(q) = h_ud;
208             h_temp(s) = non_h_ud;
209         end
```

Skridt 6

Matricen PS opdateres på baggrund af ombytningen af non_h(q) og h(s), således at denne angiver fortegnet for den nytilkomne slackvariabel i xB. Som det ses af nedenstående kode benyttes variablene ant_h, ant_nh, ant_hnc, etc., for at vide, hvor den nytilkomne slackvariabel hører til; om det er i r, i non-crossing- eller i grænsebetingelserne.

Box 9

```
287         % 'h_temp' og 'non_h_temp' beholdes, idet der nu gælder at
288         % 'XS(h_temp,:)' = K.
289         h = h_temp;
290         non_h_S = non_h_temp;
291
291         % Ændrer fortegn i PS, så det stemmer med fortegnet for
293         % residualet, som ryger ind i basis.
294         if s<=ant_h
295             PS(q,q) = 1;
296         elseif s > ant_h && s <= 2*ant_h
297             PS(q,q) = -1;
298         elseif s > 2*ant_h && s <= 2*ant_h + ant_hnc
```

```
299             PS(q,q) = sign(tau1-tau2);
300         elseif s > 2*ant_h+ant_hnc && s <= 2*ant_h+ant_hnc+ant_hncy1
301             PS(q,q) = 1;
302         elseif s > 2*ant_h+ant_hnc+ant_hncy1
303             PS(q,q) = -1;
304         end
```

Sidst men ikke mindst opdateres `ant_h`, `ant_nh`, `ant_hnc` etc. på baggrund af den nytilkomne slackvariabel i basis [7] , og `xB` opdateres.

Box 10

```
367         % Opdaterer xB.
368         xB = xB - alpha*hdir; xB(q+K) = alpha;
```

Simplex-skridtene 2 - 6 gentages indtil $d \geq 0$, dvs. en optimal løsning for den pågældende fraktil er fundet, hvorefter koefficienter, `beta`, lagres i **Beta**, tabet lagres i **T**, og indeksmængden, `h`, lagres i **H**. Denne fraktil benyttes nu til at beregne den næste fraktil, og da disse fraktiler, som nævnt, er sammenfaldende indledningsvist, sættes `xB(K+ant_nh+1: K+ant_nh+ant_nhnc)= 0`, svarende til sætte residualerne for non-crossing-betingelserne lig nul. Idet vi allerede har `h`, `non_h_S`, `PS`, `xB` mm. udelades Simplex-skridt 1, og den næste fraktil beregnes ved at gentage skridtene 2 - 6.

Metoderne, NC_{single} og $NC_{singleNO}$, har, som tidligere nævnt, den ulempe, at fraktilestimaterne afhænger af, hvilken fraktil der benyttes som begyndelses-fraktil. I det følgende udvikles en model, hvis resultat er uafhængigt af begyndelses-fraktilen, idet alle de ønskede fraktiler beregnes samtidigt.

[7] Opdatering af `ant_h`, `ant_nh` mf. er ikke medtaget grundet kodestykkets længde, se A.4.

7 Beregning af alle fraktiler på én gang

Er skrevet med inspiration fra [6].

I dette afsnit opstilles en model til beregning af alle de ønskede fraktiler på én gang, og samtidigt undgå fraktil-krydsninger. Denne metode kaldes for *Huge*.

Først opstilles en model til beregning af alle fraktiler, hvor der endnu ikke tages højde for krydsninger.

De fraktiler, der ønskes bestemt, angives via indeks-værdierne $1, 2, \ldots, l$, hvor l er det samlede antal fraktiler, og det antages at $\tau_1 < \tau_2 < \ldots < \tau_l$.
Modellen for det samlede system antager formen (5.6), men der må nu gælde, at:

$$\mathbf{A} = \begin{bmatrix} \mathbf{X} & \mathbf{0} & \ldots & \mathbf{0} & [\mathbf{I}, -\mathbf{I}] & \mathbf{0} & \ldots & \mathbf{0} \\ \mathbf{0} & \mathbf{X} & \mathbf{0} & \vdots & \mathbf{0} & [\mathbf{I}, -\mathbf{I}] & \mathbf{0} & \vdots \\ \vdots & \mathbf{0} & \ddots & \vdots & \vdots & \mathbf{0} & \ddots & \vdots \\ \mathbf{0} & \ldots & \mathbf{0} & \mathbf{X} & \mathbf{0} & \ldots & \mathbf{0} & [\mathbf{I}, -\mathbf{I}] \end{bmatrix} = [\mathbf{X}_l, \mathbf{L}], \tag{7.1}$$

og

$$\mathbf{c} = \begin{bmatrix} \mathbf{0}_{lK} \\ \tau_1 \mathbf{e} \\ (1-\tau_1)\mathbf{e} \\ \tau_2 \mathbf{e} \\ (1-\tau_2)\mathbf{e} \\ \vdots \\ \tau_l \mathbf{e} \\ (1-\tau_l)\mathbf{e} \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} \boldsymbol{\beta}_1 \\ \vdots \\ \boldsymbol{\beta}_l \\ \mathbf{r}_1^+ \\ \mathbf{r}_1^- \\ \vdots \\ \mathbf{r}_l^+ \\ \mathbf{r}_l^- \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} \mathbf{y} \\ \vdots \\ \mathbf{y} \end{bmatrix}. \tag{7.2}$$

Idet hensigten er at undgå fraktil-krydsninger, indføres nu non-crossing-betingelserne:

$$\mathbf{X}_{nc}\boldsymbol{\beta}_{j+1} \geq \mathbf{X}_{nc}\boldsymbol{\beta}_j \Leftrightarrow \tag{7.3}$$

$$\mathbf{X}_{nc}(\boldsymbol{\beta}_{j+1} - \boldsymbol{\beta}_j) \geq \mathbf{0}, \quad j = 1, \ldots, l-1. \tag{7.4}$$

Endvidere indføres følgende grænsebetingelser:

$$\mathbf{X}_{nc}\boldsymbol{\beta}_1 \geq y_{\min}\mathbf{e}, \tag{7.5}$$

$$-\mathbf{X}_{nc}\boldsymbol{\beta}_l \geq -y_{\text{maks}}\mathbf{e}, \tag{7.6}$$

hvor $y \in [y_{\min}, y_{\text{maks}}]$.

Betingelserne (7.3)-(7.6) opstilles på matrix-form på følgende måde:

$$\mathbf{X}_{NC}\begin{bmatrix}\boldsymbol{\beta}_1\\ \vdots\\ \boldsymbol{\beta}_l\end{bmatrix} = \begin{bmatrix} \mathbf{X}_{nc} & \mathbf{0} & & & \\ -\mathbf{X}_{nc} & \mathbf{X}_{nc} & \mathbf{0} & & \\ \mathbf{0} & -\mathbf{X}_{nc} & \mathbf{X}_{nc} & \ddots & \\ & \ddots & \ddots & \ddots & \mathbf{0} \\ & & & -\mathbf{X}_{nc} & \mathbf{X}_{nc} \\ & & & \mathbf{0} & -\mathbf{X}_{nc} \end{bmatrix}\begin{bmatrix}\boldsymbol{\beta}_1\\ \vdots\\ \boldsymbol{\beta}_l\end{bmatrix} \geq \mathbf{b}_{nc}, \tag{7.7}$$

hvor

$$\mathbf{b}_{nc} = \begin{bmatrix} y_{\min}\mathbf{e}_{nc} \\ \mathbf{0}_{nc} \\ \vdots \\ \mathbf{0}_{nc} \\ -y_{\text{maks}}\mathbf{e}_{nc} \end{bmatrix}. \tag{7.8}$$

Sammensættes dette med systemet fra før ((7.18)-(7.2)), opnås den samlede model:

$$\min\{\mathbf{c_s}^T\mathbf{x_s} : \mathbf{A_s}\mathbf{x_s} = \mathbf{b_s}\}, \tag{7.9}$$

hvor

$$\mathbf{A_s} = \begin{bmatrix} \mathbf{X}_l & \mathbf{L} & \mathbf{0} \\ \mathbf{X}_{NC} & \mathbf{0} & -\mathbf{I} \end{bmatrix}, \tag{7.10}$$

og

$$\mathbf{c_s} = \begin{bmatrix} \mathbf{0}_{lK} \\ \tau_1\mathbf{e} \\ (1-\tau_1)\mathbf{e} \\ \tau_2\mathbf{e} \\ (1-\tau_2)\mathbf{e} \\ \vdots \\ \tau_l\mathbf{e} \\ (1-\tau_l)\mathbf{e} \\ \mathbf{0} \end{bmatrix}, \quad \mathbf{x_s} = \begin{bmatrix}\mathbf{x}\\ \mathbf{r}_{nc}\end{bmatrix} = \begin{bmatrix} \boldsymbol{\beta}_1 \\ \vdots \\ \boldsymbol{\beta}_l \\ \mathbf{r}_1^+ \\ \mathbf{r}_1^- \\ \vdots \\ \mathbf{r}_l^+ \\ \mathbf{r}_l^- \\ \mathbf{r}_{nc} \end{bmatrix}, \quad \mathbf{b_s} = \begin{bmatrix}\mathbf{b}\\ \mathbf{b}_{nc}\end{bmatrix} = \begin{bmatrix} \mathbf{y} \\ \vdots \\ \mathbf{y} \\ y_{\min}\mathbf{e}_{nc} \\ \mathbf{0}_{nc} \\ \vdots \\ \mathbf{0}_{nc} \\ -y_{\text{maks}}\mathbf{e}_{nc} \end{bmatrix}. \tag{7.11}$$

Der gælder her, at:

$$\mathbf{r}_{nc} = \begin{bmatrix} |y_{\min}\mathbf{e}_{nc} - \mathbf{X}_{nc}\boldsymbol{\beta}_1| \\ |\mathbf{X}_{nc}(\boldsymbol{\beta}_1 - \boldsymbol{\beta}_2)| \\ \vdots \\ |\mathbf{X}_{nc}(\boldsymbol{\beta}_{l-1} - \boldsymbol{\beta}_l)| \\ |\mathbf{X}_{nc}\boldsymbol{\beta}_l - y_{\text{maks}}\mathbf{e}_{nc}| \end{bmatrix}. \tag{7.12}$$

Den samlede model er således opstillet, men inden modellen kan benyttes, er vi nødt til først at betragte matricen, $\mathbf{B_s}$, og dens inverse, $\mathbf{B_s}^{-1}$. I den forbindelsen dannes den samlede matrix, $\mathbf{X}_s$:

$$\mathbf{X}_s = \begin{bmatrix} \mathbf{X}_l \\ \mathbf{X}_{NC}, \end{bmatrix}. \tag{7.13}$$

hvorved $\mathbf{B}_s$ kan komme på formen (5.13):

$$\mathbf{B}_s = \begin{bmatrix} \mathbf{X}_s(h) & \mathbf{0} \\ \mathbf{X}_s(\overline{h}) & \mathbf{P}_s \end{bmatrix}, \tag{7.14}$$

hvor $\mathbf{P}_s$ er en diagonal matrix med $\text{sign}(\mathbf{r}_s(\overline{h}))$ i diagonalen.

Det skal her nævnes, at $\mathbf{r}_s$ er den samlede vektor med residualer, dvs.

$$\mathbf{r}_s = \begin{bmatrix} \mathbf{r}_1 \\ \vdots \\ \mathbf{r}_l \\ \mathbf{r}_{nc} \end{bmatrix}. \tag{7.15}$$

Ydermere bør det pointeres, at $h \in \mathbb{N}^{Kl}$, og indeholder indeks-værdier fra mængden $\{1, \ldots, Nl + N_{nc}(l+1)\}$, og at $\overline{h}$ derved indeholder indeks-værdierne $\{1, \ldots, Nl + N_{nc}(l+1)\} \setminus \{h\}$.

Strukturen i $\mathbf{B}_s$ leder til følgende form af $\mathbf{x}_\mathcal{C}$ og $\mathbf{C}$.

$$\mathbf{x}_\mathcal{C} = \begin{bmatrix} \mathbf{r}_s(h_r)^+ \\ \mathbf{r}_s(h_r)^- \\ \mathbf{r}_s(h_{nc}) \end{bmatrix}, \quad \mathbf{C} = \begin{bmatrix} \mathbf{I} & -\mathbf{I} & 0 \\ 0 & 0 & -\mathbf{I} \end{bmatrix}, \tag{7.16}$$

hvor h_r, $h_{nc} \in h$.

Da $\mathbf{B}_s$ er på formen (5.13), kan $\mathbf{B}_s^{-1}$ altså beregnes via fremgangsmetoden beskrevet i afsnit 5.1, hvilket dog besværliggøres af matricens størrelse. Vektoren $\overline{h}$ indeholder $l(N + N_{NC} - K) + N_{NC}$ elementer, hvilket betyder, at $\mathbf{X}_s(\overline{h})$ bliver en $(l(N + N_{NC} - K) + N_{NC}) \times Kl$ matrix. Dette medfører, i vores tilfælde, at $\mathbf{X}_s(\overline{h})$ undervejs i udregningerne opnår en størrelse på ca. 80.000×800, idet $\mathbf{B}_s^{-1}$ er endnu større, fås altså matricer, der indeholder flere milliarder elementer!
For overhovedet at kunne implementere metoden *Huge*, kræves en speciel håndtering af matricerne, hvilket betragtes nærmere i det følgende afsnit.

7.1 Sparse matricer i Matlab

I det følgende beskrives kort egenskaberne ved sparse matricer i MATLAB samt nødvendigheden af disse i henhold til implementeringen af `NonCrossHuge`, som bygger på modellen *Huge*. Undervejs klarlægges, ud fra et programmeringsteknisk synspunkt, de umiddelbare fordele og ulemper ved sparse matricer.

Ved en sparse matrix forstås en matrix, som primært er besat af nul-elementer. I MATLAB lagres en sparse matrix ved dens ikke-nul-elementer, hvorfor man ved vektor-matrix-multiplikationer undgår unødvendige operationer med nul-elementer samt lagringen af disse. Af denne grund mindskes både beregningstiden samt lagringspladsen, som for sparse matricer vokser lineært som funktion af antallet af ikke-nul-elementerne [4].

En simpel måde at konstruere en sparse matrix er ved angive den på *triplet*-form, dvs. ved at angive tre vektorer `I`, `J` og `values`, som angiver række- og søjleindeks samt værdierne.

Værdier	`values`	0.5	-1	28	-2.3	8	10
Rækkeindeks	`I`	1	1	2	3	4	4
Søjleindeks	`J`	1	3	2	1	3	4

Ved funktionskaldet `A = sparse(I,J,values)` fås matricen `A`:

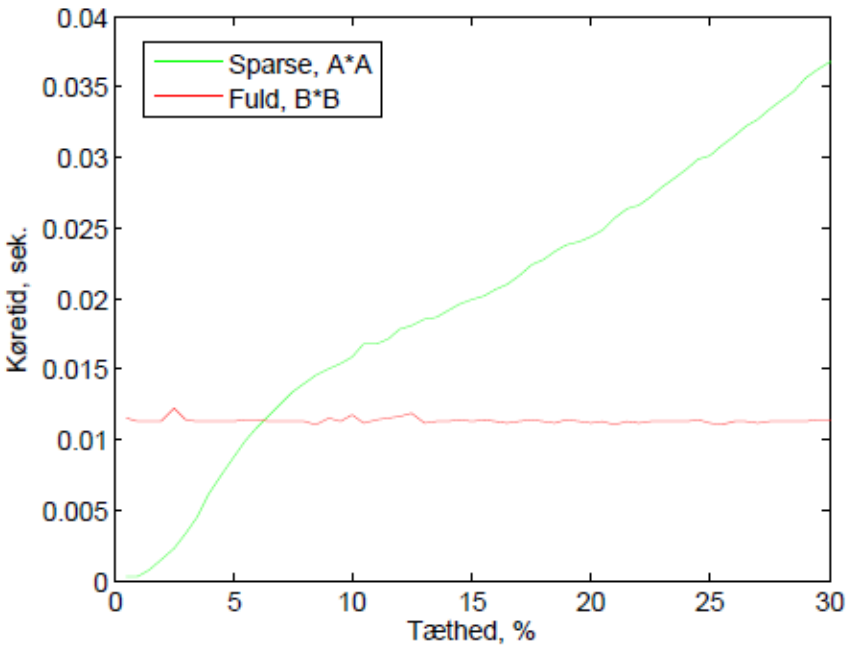
$$\mathbf{A} = \begin{bmatrix} 0.5 & & -1 & \\ & 28 & & \\ -2.3 & & & \\ & & 8 & 10 \end{bmatrix} \tag{7.17}$$

Dimensionerne af `A` afhænger her af de tre ovennævnte vektorer. Alternativt prædefineres størrelsen af `A` ved f.eks. `m = 6` og `n = 6`, og af `A = sparse(I,J,values, m, n)`, fås nu:

$$\mathbf{A} = \begin{bmatrix} 0.5 & & -1 & & \\ & 28 & & & \\ -2.3 & & & & \\ & & 8 & 10 & \\ & & & & \\ & & & & \end{bmatrix} \tag{7.18}$$

Enhver ikke-sparse matrix, `B`, kan gøres sparse ved `A = sparse(B)`. Ligeledes kan en sparse matrix, `A`, konverteres til en fuld matrix ved `B = full(A)`.

I MATLAB opnås en tidsmæssig gevinst ved multiplikationer med sparse matricer, hvis ikke matricernes tæthed overstiger 5% [4]. Med andre ord må antallet af ikke-nul-elementer ikke overstige 5% af det samlede antal elementer i den fulde matrix. I figur 7.1 ses beregningstiden for matrix-multiplikationerne $\mathbf{A} \cdot \mathbf{A}$ og $\mathbf{B} \cdot \mathbf{B}$, som funktion af tætheden, hvor



Figur 7.1: Her ses beregningstiden for matrix-multiplikationerne $\mathbf{A} \cdot \mathbf{A}$ og $\mathbf{B} \cdot \mathbf{B}$.

$\mathbf{A}$ er en sparse-matrix og $\mathbf{B}$ er en fuld matrix. Begge matricer har størrelsen 500×500. Beregningstiden for fulde matricer er $\mathcal{O}(n^3)$, og idet matricernes størrelser holdes fast ($n = 500$), er beregningstiden for $\mathbf{B} \cdot \mathbf{B}$, som forventet, konstant. Anderledes gælder der for $\mathbf{A} \cdot \mathbf{A}$, at beregningstiden, som tidligere nævnt, vokser tilnærmelsesvist lineært.

Under implementeringen af `NonCrossHuge` har vi udnyttet egenskaberne ved sparse matricer. Disse er taget i brug, dels for at mindske beregningstiden, men i langt højere grad for at mindske lagringspladsen. Inden vi fortsætter med en redegørelse af implementeringen af `NonCrossHuge`, gives her en forklaring på hvordan matricen $\mathbf{B^{-1}}$ konstrueres i den pågældende funktion. I tabel 7.1 er vindkraft-datasættet for 24-timers horisonten[8] benyttet under første iteration af `NonCrossHuge`.

$\mathbf{B^{-1}}$ består af fire delmatricer, $\mathbf{B_{11}}$, $\mathbf{B_{12}}$, $\mathbf{B_{21}}$ og $\mathbf{B_{22}}$, som her er gjort sparse, hvor $\mathbf{B_{12}}$ er en nul-matrix. Grundet størrelsen af $\mathbf{B_{12}}$, $\mathbf{B_{21}}$ og $\mathbf{B_{22}}$ kan lagringen af disse kun muliggøres ved brugen af sparse matricer - hvis disse forsøges lagret som fulde matricer på de computer, som er blevet anvendt i projektets forløb, fås meddelelsen *"Out of Memory"*.

Tætheden for $\mathbf{B_{11}}$ og $\mathbf{B_{21}}$ overstiger de førnævnte 5%. Idet sidstnævnte matrix kun indgår i sammensætningen af $\mathbf{B^{-1}}$, og ikke benyttes i en matrix-multiplikation har tætheden ingen indflydelse på beregningstiden. Anderledes står det til med $\mathbf{B_{11}}$, som ikke er gjort

[8] Datasættet forklares nærmere i afsnit 10.

	Størrelsen	**Antal ele.**	**Antal ikke-nul-ele.**	**Tæthed**
$\mathbf{B_{11}}$	392×392	153.664	22.508	14.65%
$\mathbf{B_{12}}$	392×78.060	30.599.520	0	0%
$\mathbf{B_{21}}$	78.060×392	30.599.520	4.387.038	14.34%
$\mathbf{B_{22}}$	78.060×78.060	$6.0934e+9$	78.060	$1.2811e-5\%$
$\mathbf{B^{-1}}$	78.452×78.452	$6.1547e+9$	4.487.606	$7.2913e-4\%$

Tabel 7.1: Her ses størrelsen, antallet af elementer, antallet af ikke-nul-elementer samt tætheden for $\mathbf{B^{-1}}$ og for de matricer, som indgår i beregningen af denne.

sparse med henblik på lagring af denne, men fordi den indgår i beregningen af $\mathbf{B_{12}}$. Hvis $\mathbf{B_{12}}$ skal blive sparse, skal operatorerne, som indgår i beregningen af denne også være sparse - derfor er $\mathbf{B_{11}}$ gjort sparse.

Det ses, at brugen af sparse matricer har muliggjort implementeringen af `NonCrossHuge` uden yderligere ændring af struktur og lign. af de implicerede matricer. Sparse matricer er her anvendt dels for at mindske beregningstiden, men i langt højere grad for at mindske lagringspladsen, så det overhovedet er muligt at lagre $\mathbf{B}^{-1}$.
I det følgende afsnit redegøres der for selve implementeringen af `NonCrossHuge`.

7.2 Algoritme 3 - NonCrossHuge

I dette afsnit gives en redegørelse for opbygningen af `NonCrossHuge` på baggrund af de enkelte Simplex-skridt beskrevet i afsnit 5.2. Under redegørelsen inddrages dele af den skrevne `Matlab`-kode.

Følgende funktion er udviklet på egen hånd på baggrund af modellen opstillet i 7 samt af funktionen `NonCrossSingle`.

`NonCrossHuge` har følgende funktionskald:

```
[Beta,T] = NonCrossHuge(X,beta0,y,Xnc,tau,lb,ub);
```

Her ses en kort præsentation af input samt output-argumenterne for funktionen:

Input argumenter:

`X` : Designmatrix for datapunkter.

`beta0` : En matrix med startgæt for alle fraktiler, som ønskes beregnet.

`y` : En datavektor for responsvariablen.

`Xnc` : Designmatrix for udvalgte kontrolpunkter.

`tau` : En vektor bestående af værdier $\tau \in [0,1]$, hvor $\tau_1 \leq \tau_2 \leq, \ldots, \leq \tau_n$.

`lb` : Nedre grænse.

`ub` : Øvre grænse.

Output argumenter:

`Beta` : En matrix, hvis i'te søjle angiver $\hat{\boldsymbol{\beta}}(\tau_i)$.

`T` : En vektor, hvis i'te element indeholder tabet for den τ_i'te fraktil.

Funktionen `NonCrossHuge` er en udvidelse af `NonCrossSingle`. Funktionen beregner koefficienterne for hver af de ønskede fraktil-kurver, således at disse ikke krydser.

Funktionen bestemmer, ud fra en givent startgæt, $\hat{\boldsymbol{\beta}}_0$, en matrix bestående af samtlige sæt af koefficienter, $\hat{\boldsymbol{\beta}}(\tau_i)$, for alle τ_i i input vektoren, $\boldsymbol{\tau}$, hvor τ angiver fraktilerne, for hvilke koefficienterne ønskes beregnet.
I modsætning til `NonCrossSingle`, hvor koefficienterne, $\hat{\boldsymbol{\beta}}(\tau_i)$, bestemmes enkeltvist, og $\hat{\boldsymbol{\beta}}(\tau_{i+1})$ bestemmes på baggrund af $\hat{\boldsymbol{\beta}}(\tau_i)$, bestemmes samtlige sæt af $\hat{\boldsymbol{\beta}}(\tau_i)$ samtidigt i `NonCrossHuge`.

Herefter returneres en matrix, `Beta`, hvis i'te søjle angiver koefficientvektoren $\hat{\boldsymbol{\beta}}(\tau_i)$ samt en vektor, `T`, indeholdende tabet for de enkelte fraktiler.

Implementeringen af NonCrossHuge

Skridt 1

Givet de observerede data, `y`, designmatricen, `X` samt startgættet, `beta0`, beregnes residualerne, `rs` ($\mathbf{r}_s$), bestående af `r` og `r_nc` ($\mathbf{r}_{nc}$) - se Box 1. Vektoren `r` angiver afvigelsen mellem `y`, og de enkelte fraktiler, og `r_nc` angiver den indbyrdes afstand imellem fraktilerne, hvor den øvre og nedre grænse, under opsætningen af grænsebetingelserne, behandles som fraktiler.

Til beregningen af `r` benyttes MATLAB-funktionen, `repmat`, som konstruerer en matrix, hvis søjler består af `y`. Herved fås en vektor, `r`, bestående af residualerne for samtlige fraktiler.
For matricen, `r_nc`, beregnes søjlerne ved:

$$\mathbf{lb} \;-\; Q(\tau_1;\mathbf{x}_{pred}) \tag{7.19}$$
$$Q(\tau_i;\mathbf{x}_{pred}) \;-\; Q(\tau_{i+1};\mathbf{x}_{pred}), \quad i=1,\ldots,l-1 \tag{7.20}$$
$$Q(\tau_l;\mathbf{x}_{pred}) \;-\; \mathbf{ub}. \tag{7.21}$$

hvor lb er den nedre grænse, og ub er den øvre grænse. Også `r_nc` omskrives til en vektor, for herved at sammensætte `rs`, som repræsenterer slackvariablene for Simplex-modellen, hvor `r_nc` er slackvariable for alle non-crossing-betingelser.

Box 1

```
% Residualerne imellem fraktilerne og 'y' opstilles i en vektor 'r'.
% Den indbyrdes afstand imellem fraktilerne opstilles i en vektor 'rnc'.
r = repmat(y,1,ant_frak)-X*beta0;
r = r(:);        % Fra matrix til vektor.
r_nc = [lb*ones(Nnc,1) Xnc*beta0]-[Xnc*beta0 ub*ones(Nnc,1)];
r_nc = r_nc(:);  % Fra matrix til vektor.

% 'r' og 'rnc' opstilles i en vektor 'rs'.
rs = [r; r_nc];
```

Herefter beregnes indeks-mængderne `h` og `non_h` (h og $\overline{h}$) - se Box 2.

Til formålet benyttes funktionen `find_h_Huge` (afs. A.7), som i lighed med `find_h`, først bestemmer indeks-mængden, `I`, for hvilke der gælder at `abs(rs(I))< tol`. Også her er tolerancen, `tol = 1e-05`, fundet ved "*trial and error*" og sættes til en værdi, som aftvinger mindst `ant_frak*K` elementer i `I`, hvor `ant_frak` er antallet af fraktiler, som

ønskes bestemt.
Herefter benyttes funktion, `find_PartialXS` (afsnit A.8), med funktionskaldet,

```
XS_I = find_PartialXS(X,Xnc,I,ant_frak),
```

som konstruerer en designmatrix, `XS_I = XS(I,:)`, på samme form som $\mathbf{X}_l$ i (7.18). Fra denne designmatrix udvælges `K*ant_frak` lineært uafhængige rækker, og indeksværdierne for disse udgør `h`, og via `h` bestemmes `non_h`. I modsætning til `NonCrossSingle`, hvor der skelnes imellem `non_h` og `non_h_S`, bruges i `NonCrossHuge` kun `non_h`, hvis elementer både knytter sig til `r` og `r_nc`.

Endvidere samles vektoren, `xB` ($\mathbf{x}_{\mathcal{B}}$), bestående af stargættet, `beta0`, samt de absolutte residualer, `abs(rs(non_h))`.

Box 2

```
20 % Bestemmer 'h' for alle fraktilerne.
21 [h, P, non_h] = find_h_huge(X, Xnc, rs, ant_frak);
22
23 % Samler xB bestående af beta0 og slackvariablene.
24 xB = [beta0(:) ; abs(rs(non_h))];
```

Dernæst beregnes antallet af elementer i `h`, som hører til hhv. `r` og `r_nc`. Tilsvarende optælling foretages for `non_h` - se Box 3.

Outputtet, `P`, fra funktionen, `find_h_Huge`, angiver fortegnene for residualerne. I modsætning til `CrossQuant` og `NonCrossSingle` konstrueres via `P` en vektor, `Pvektor`, samt en sparse matrix, `Pmatrix`, på triplet-form. Årsagen er, at det, grundet `P`'s størrelse, er beregningstungt at konvertere `P` fra en vektor til en matrix og omvendt, som det ellers blev gjort i de to foregående funktioner.

Via `h` og `non_h` benyttes igen `find_PartialXS`, for at konstruere de sparse matricer `XS_h` ($\mathbf{X}_s(h)$) og `XS_nh` ($\mathbf{X}_s(\overline{h})$), som benyttes til beregningen af `B_inv`.

Box 3

```
26 % Her tælles antallet af 'h', 'non_h', 'h_nc', og 'non_h_nc', for at holde
27 % styr på hvor elementerne i hhv. 'h' og 'non_h' kommer fra; om det er fra
28 % 'r' eller 'rnc'.
29 ant_h = sum(h<=N*ant_frak);
30 ant_nh = N*ant_frak-ant_h;
31 ant_hnc = length(h)-ant_h;
32 ant_nhnc = length(non_h)-ant_nh;
33
34 % Via diagonalmatricen, P, konstrueres både 'Pvektor' og 'Pmatrix'.
35 P(end-ant_nhnc+1:end) = -1;
36 lp = length(P); lR = length(R);
37 Pvector = P; Pmatrix = sparse(1:lp, 1:lp, P);
38
```

```
39 % For hvert sæt af 'h_i' og 'non_h_i', tilhørende fraktil 'Q_i', udtrækkes
40 % de tilsvarende rækker i 'XS'.
41 XS_h = find_PartialXS(X,Xnc,h,ant_frak);
42 XS_nh = find_PartialXS(X,Xnc,non_h,ant_frak);
43 mXSh = length(XS_h);
44
45 % Beregner B_inv
46 B11 = sparse(eye(size(XS_h)))/XS_h;
47 B12 = sparse(mXSh,lp);
48 B21 = -Pmatrix'*XS_nh*B11;
49 B22 = Pmatrix;
50 B_inv = [B11 B12 ; B21 B22];
```

Skridt 2

I linjerne 54 - 56 beregnes hvor mange af residualerne i `r`, som hører til de enkelte fraktiler. Til dette formål bestemmes en vektor, `non_h_X`, bestående af elementerne i `non_h`, som er mindre end `ant_frak*N`. Med andre ord indeholder `non_h_X` indeksværdierne for hver af de slackvariable, som er samlet i `r`. Der beregnes nu, via `non_h_X`, en vektor `X_frak_nr`, som angiver fraktilnummeret for hvilken, elementerne i `non_h_X` knytter sig til. Via MATLAB-funktionen, `histc` beregnes hvor hyppigt hvert af fraktilnumrene i `X_frak_nr` opstår. Herved fås vektoren, `ant_res_frak`, hvis `i`'te element, angiver antallet af residualer i `r`, som hører til den `tau(i)`'te fraktil.

Herefter at beregnes, via `ant_frak_res`, de asymmetriske tabsfunktioner `rho`, `rho_c_p` og `rho_c_n` gældende for residualerne i `r`, de positive og de negative residualer uden for basis, dvs. i $\mathbf{x}_{\mathcal{C}}$.

Box 4

```
52 % Beregner antallet af residualer fra 'r' for hvert af fraktilerne for
53 % herefter at kunne beregne 'rho'.
54 non_h_X = non_h(non_h<=ant_frak*mX);
55 X_frak_nr = ceil(non_h_X/mX);
56 ant_res_frak = histc(X_frak_nr,1:ant_frak);
57
58 rho = Pvector(1:ant_nh);
59 start = 0;
60 for i = 1:ant_frak
61    rho_temp = rho(start+1:start+ant_res_frak(i));
62    rho_temp(rho_temp == -1) = 1-tau(i);
63    rho_temp(rho_temp == 1) = tau(i);
64    rho(start+1:start+ant_res_frak(i)) = rho_temp;
65    start = start + ant_res_frak(i);
66 end
67
68 % Beregner rho_c_p og rho_c_n, dvs. koefficienterne for objektfunktionen
69 % for både de positive og negative residualer i xC.
70 rho_c_p = zeros(ant_h,1);
```

```
71 ant_c_frak = length(y)-ant_res_frak;
72 start = 0;
73
74 for i = 1:ant_frak;
75    rho_c_temp = ones(ant_c_frak(i),1)*tau(i);
76    rho_c_p(start+1:start+ant_c_frak(i)) = rho_c_temp;
77    start = start + ant_c_frak(i);
78 end
79 rho_c_n = 1-rho_c_p;
```

Endvidere bregnes matricen `C`, som grundet det store antal nul-elementer, konstrueres som en `mC` $\times$ `nC` sparse matrix angivet på *triplet*-form. Ved *triplet*-form angives tre vektorer, `IC`, `JC` og `Cval`, hvor de to førstnævnte angiver række/søjle-placeringen og den sidstnævnte angiver værdien.

Box 5

```
 81 % Beregner her den sparse matrix C angivet på triplet-form.
 82
 83 % Alle ikke-nul-elementer lagres i vektoren 'Cval'.
 84 Cval = [repmat([1 -1], [1 ant_h])'; -ones(ant_hnc,1)];
 85
 86 % Søjle-indekser.
 87 JC = (1:2*ant_h+ant_hnc)';
 88 JC(1:2:2*ant_h) = 1:ant_h;
 89 JC(2:2:2*ant_h) = ant_h+1:2*ant_h;
 90
 91 % Række-indesker.
 92 IC = repmat(1:ant_h, [2 1]);
 93 IC = [IC(:); (ant_h+1:ant_h+ant_hnc)'];
 94
 95 % Størrelsen af 'C'.
 96 mC = lR;
 97 nC = 2*ant_h+ant_hnc;
 98
 99 % Konstruerer 'C'.
100 C = sparse(IC(:), JC, Cval, mC, nC);
```

Herefter beregnes vektoren, `d` (se Box 6), hvis elementer repræsenterer de relative ændringer i objektfunktionen fra $\mathbf{x}^{(K)}$, og der undersøges herefter om `d` ≥ 0.

Er dette tilfældet afbrydes funktionen, hvis ikke, udvælges et negativt element `d(s)` i `d` samt dets tilhørende indeks, `s`. Hvordan `s` og `d(s)` vælges, beskrives i det følgende.

Box 6

```
102 % Beregner 'd', som angiver det relative fald for alle retninger.
103 cC = [rho_c_p; rho_c_n; zeros(ant_hnc,1)];
104 cB = [zeros(ant_frak*K,1); rho; zeros(ant_nhnc,1)];
105 g = B_inv'*cB;
106 d = cC - C'*g;
```

Skridt 3

I lighed med `NonCrossSingle` udvælges her *alle* negative elementer i `d` samt deres tilhørende indeks-værdier, som lagres i vektorerne `ds` og `s` - se Box 7. Som nævnt i forrige afsnit angiver de negative elementer i `d` det relative fald i objektfunktionen for en given retning, hvorimod det *egentlige* fald i objektfunktionen *også* afhænger af skridtlængden, `alpha` (α), hvorfor disse værdier bestemmes for alle `s(i)`.

For at bestemme `alpha` beregnes først `sigma(i)` (σ_i) for alle `hdir(i)`. For at slackvariablene i `xB` ikke skal blive negative, vælges `alpha = min(sigma)`, og det tilsvarende indeks i `sigma`, `q`, gemmes. I `NonCrossSingle` beregnes `sigma` enkeltvist for hver aftagende retning. I `NonCrossHuge`, derimod, beregnes *alle* `sigma`'er for alle aftagende retninger *samtidigt*. `sigma` bliver herved en matrix, hvis søjler angiver de mulige skridtlængder for hver retning. Til beregningen af `sigma` anvendes ikke en `for`-løkke, som det var tilfældet i `CrossQuant`; grundet `sigma`'s øgede længde, som en konsekvens af de indførte non-crossing-betingelser, beregnes `sigma` via vektorer.

Ethvert sæt `alpha`, `q` og `s(i)` lagres i den i'te række i den pre-allokerede matrix `a_q_s` med henblik på senere brug.

Box 7

```
110     % Tæller antallet af aftagende retninger i 'd' og lagrer dem i en
111     % vektoren 'ds'. Deres tilhørende indeksværdier lagres i 's'.
112     ant_neg_d = sum(d<0);
113     [ds s]= sort(d);
114     ds = ds(1:ant_neg_d);
115     s = s(1:ant_neg_d);
116     a_q_s = zeros(ant_neg_d,3);
117     index3 = 1:ant_neg_d;
118
119     % For alle aftagende retninger, beregnes en matrix med retninger.
120     hdir = B_inv*C(:,s);
121
122     % Hvis problemer er begrænset beregnes 'alpha', 'q' og 's' for alle
123     % aftagende retninger. Disse værdier lagres i 'a_q_s'.
124     if  sum(hdir > 0) > 0
125         % 'alpha' beregnes udfra 'sigma', som bestemmes her.
126         sigma = zeros(size(hdir,1)-ant_frak*K,ant_neg_d);
127         xB_nedre = repmat(xB(ant_frak*K+1:end),1,ant_neg_d);
128         hdir_nedre = hdir(ant_frak*K+1:end,:);
129         Ihdir = hdir_nedre>tol;
130         sigma(Ihdir) = xB_nedre(Ihdir)./...
131             hdir_nedre(Ihdir);
132         sigma(~Ihdir) = inf;
133
134         % Her gemmes 'alpha', 'q' og 's' i matricen 'a_q_s'.
135         [alpha q] = min(sigma);
136         a_q_s(:,1) = alpha;
137         a_q_s(:,2) = q;
138         a_q_s(:,3) = s;
```

```
139     end
```

Skridt 4

For alle de potentielle skridtlængder i matricen `a_q_s`, bestemmes nu det *egentlige* fald i objektfunktionen. Hvis `a_q_s(:,1)= 0`, dvs. alle skridtlængder har værdien nul, vælges den første af de mulige skridtlængder, `a_q_s(1,1)` samt dens tilhørende `q = a_q_s(1,2)` og `s = a_q_s(1,3)`, da disse hører til den mest aftagende retning - se Box 8
Hvis én eller flere af skridtlængderne er større end nul beregnes, via disse, det *egentlige* fald i objektfunktionen. Skridtlængden tilhørende det største *egentlige* fald vælges, samt dennes `s` og `q`.

Via disse tre værdier udvælges én retning ud af matricen `hdir`.

Box 8

```
141     if  sum(a_q_s(:,1) > 0) == 0
142         % Finder 's' og 'q', hvis alle alpha = 0.
143         alpha = a_q_s(1,1);
144         q = a_q_s(1,2);
145         s = a_q_s(1,3);
146     else
147         % Finder 's' og 'q', hvis ikke alle alpha = 0. Dette gøres ved at
148         % vælge det element i 'd', som giver det største egentlige fald i
149         % objektfunktionen. Det egentlige fald for alle alpha'er beregnes
    i
150         % 'gain'.
151         gain = ds.*a_q_s(:,1);
152         [gain Igain] = min(gain);
153         alpha = a_q_s(Igain,1);
154         q = a_q_s(Igain,2);
155         s = a_q_s(Igain,3);
156     end
157
158     % Retningen bestemmes for den valgte 's' og 'q'.
159     Ihh = index3(a_q_s(:,3) == s); Ihh = Ihh(1);
160     hdir = hdir(:,Ihh);
```

Skridt 5

Herefter ombyttes elementet `non_h(q)` med `h(s)` (se Box 9). Vektoren `Pvektor` opdateres på baggrund af ombytningen af `non_h(q)` og `h(s)`, således at denne angiver fortegnet for den nytilkomne slackvariabel i `xB`. Som det ses af nedenstående kode benyttes variablene `ant_h`, `ant_nh`, `ant_hnc`, etc., for at vide, hvor den nytilkomne slackvariabel hører til; om det er i `r` eller `r_nc`.

Box 9

```
% Skifter en slackvariabel ind og ud af basis.
non_h_ud = non_h(q);
if s>ant_h
    h_ud = h(s-ant_h);
    non_h(q) = h_ud;
    h(s-ant_h) = non_h_ud;
else
    h_ud = h(s);
    non_h(q) = h_ud;
    h(s) = non_h_ud;
end

% Ændrer fortegn i P, så det stemmer med fortegnet for residualet, som
% ryger ind i basis.
if  s<=ant_h
    Pvector(q) = 1;
else
    Pvector(q) = -1;
end
```

Skridt 6

Sidst men ikke mindst opdateres `ant_h`, `ant_nh`, `ant_hnc` etc. på baggrund af den nytilkomne slackvariabel i basis, og `xB` opdateres.

Box 10

```
% Opdaterer antallet af slackvariable inde og udenfor basis.
if s <= 2*ant_h && q > ant_nh
    ant_hnc = ant_hnc + 1;
    ant_nhnc = ant_nhnc - 1;
    ant_h = ant_h - 1;
    ant_nh = ant_nh + 1;
elseif s > 2*ant_h && q <= ant_nh
    ant_h = ant_h + 1;
    ant_nh = ant_nh - 1;
    ant_hnc = ant_hnc - 1;
    ant_nhnc = ant_nhnc + 1;
end

% Opdaterer xB.
xB = xB - alpha*hdir; xB(q+ant_frak*K) = alpha;
```

Herefter gentages Simplex-skridtene 2-6 indtil $\mathbf{d} \geq \mathbf{0}$, dvs. en optimal løsning for de pågældende fraktiler er fundet.

Del III
Resultater og analyse

8 Introduktion

I dette afsnit anvendes metoderne, $Cross$, NC_{single}, $NC_{singleNO}$ og $Huge$, til at bestemme fraktil-modeller på formen 2.7. Modellerne bestemmes via de implementerede funktioner, `CrossQuant`, `NonCrossSingle` og `NonCrossHuge`, som er blevet beskrevet i de foregående afsnit, og de estimerede modeller vil blive analyseret og diskuteret efterfølgende.

I afsnittet betragtes indledningsvist et simpelt lineært tilfælde, hvor den teoretiske fordeling er kendt - afsnit 9. Dernæst testes implementeringerne på et udleveret sæt af vindkraft-data fra Tunø Knob Offshore Wind Farm, som bla. indeholder målinger af prædikterede vindkraft-produktioner, de faktiske vindkraft-produktioner og tidshorisonten for prædiktionerne - afsnit 10.

I begge afsnit benyttes de ovennævnte funktioner til at beregne fraktil-koefficienterne, $\hat{\boldsymbol{\beta}}$, for alle de ønskede fraktil-modeller, som i fællesskab udgør én samlet model, der beskriver den betingede fordelingen af responsvariablen. Til formålet benyttes et såkaldt træningssæt, som er en datamængde, der er udvalgt tilfældigt fra det givne datasæt. Dernæst testes de estimerede fraktil-modeller vha. et testsæt, som er en anden tilfældigt udvalgt datamængde fra samme datasæt.

I sammenligningen af de estimerede fraktil-modellers præstation, benyttes testsættet til beregning af det samlede tab for alle fraktil-modeller. Denne fremgangsmåde benyttes, idet de sande fraktil-kurver altid vil have det laveste tab sammenlignet med estimerede fraktil-kurver, hvis der testes med nok data. Derfor forventes det, at jo lavere et tab, desto større er sandsynligheden for, at de estimerede fraktiler ligger tæt på de sande. Desuden argumenteres der for i [8], at netop tabet er det bedste mål for, hvor god en fraktil-model er.
Under sammenligningen af de forskellige modeller, undersøges det desuden, hvor gode fraktil-kurverne rent faktisk er til at dele punkterne korrekt op, da det selvsagt er en vigtig egenskab inden for fraktil-regression.

For overskuelighedens skyld gives her et kort oprids af de fire metoder, der benyttes undervejs:

- **Cross**:
  I denne metode benyttes funktionen, `CrossQuant`, som beregner fraktilerne enkeltvist, uden at tage hensyn fraktil-krydsninger. Denne metode benytter 50%-fraktilen beregnet vha. R, som startgæt, jvf. afsnit 5.

- **$\mathbf{NC_{single}}$**:
  Denne metode anvender funktionen `NonCrossSingle`, som beregner fraktilerne enkeltvist, hvor der tages højde for fraktil-krydsninger. Begyndelses-fraktilen, dvs. 50%-fraktilen, beregnes via R, og gives som input i funktionen, hvorefter alle fraktilerne over, og dernæst under, beregnes. Jvf. afsnit 6.1.

- **$NC_{singleNO}$**:
  I lighed med NC_{single} benyttes her funktionen, `NonCrossSingle`, hvor 50%−fraktilen gives som input, og fraktilerne over og under beregnes, uden at de krydser. Dernæst benyttes den nederste og den øverste fraktil, som begyndelse-fraktil, således at fraktil-modellen estimeres nedefra og op, og dernæst oppefra og ned. Det endelige estimat fås ved at beregne gennemsnittet af de to kørsler. Der er altså her tale om en udvidelse af NC_{single}. Jvf. afsnit 6.2.

- **Huge**:
  I denne metode benyttes funktionen `NonCrossHuge`, som beregner alle fraktil-kurverne samtidigt, mens der sørges for at disse ikke krydser. Startgættet til denne metode fås via `NonCrossSingle`. Jvf. afsnit 7.

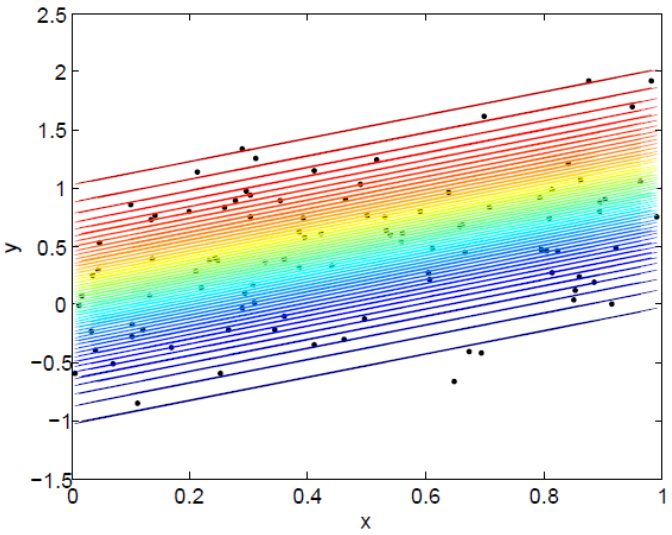
9 Fraktil-modeller for et lineært eksempel

9.1 Introduktion

I dette afsnit testet de tre metoder $Cross$, NC_{single}, og $Huge$ på et simpelt lineært eksempel, hvor selve fordelingen er kendt. Data består her af nogle x- og y-værdier, hvor X er uniformt fordelt mellem 0 og 1, og Y er normalfordelt med middelværdien x og standardafvigelsen 1/2. Dermed stiger middelværdien for den betingede fordeling af Y lineært, og det samme gælder de betingede fraktiler. Fraktil-kurverne bør altså være rette linjer, og der fittes derfor med lineære kurver.
Selve datapunkterne er genereret i R, og der vælges et tilfældigt træningssæt på 100 datapunkter. Da intervallet, Y kan ligge i, ikke er begrænset, sættes der ingen grænsebetingelser for fraktil-kurverne. Dette gøres rent praktisk ved at sætte den nedre grænse og den øvre grænse så hhv. lavt og højt, at de aldrig får nogen indflydelse.

9.2 Resultater

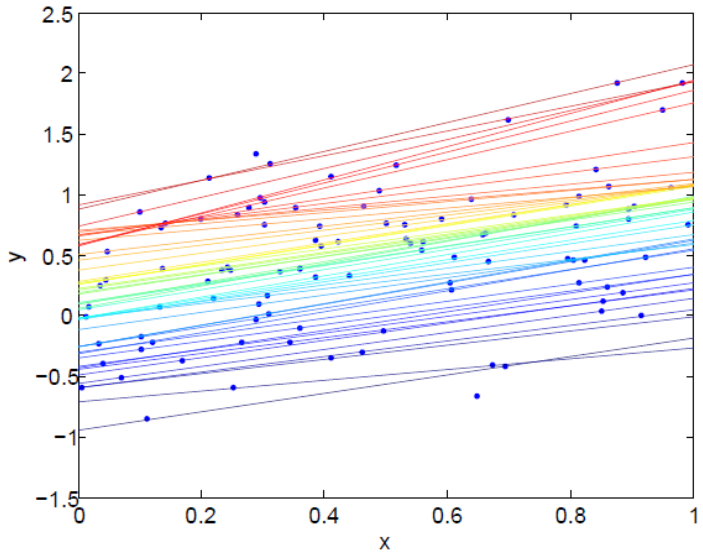
Idet fordelingen er kendt, kan de sande fraktiler-kurver beregnes. Dette gøres via funktionen, `qnorm`, i R for 2%−fraktilen, 4%−fraktilen, 6%−fraktilen, osv. op til 98%−fraktilen, således at 49 forskellige fraktil-kurver for den betingede fordeling af Y fås. Et plot af de disse kurver ses i figur 9.1 sammen med punkterne i træningssættet.



Figur 9.1: De sande fraktiler beregnet i R via `qnorm`.

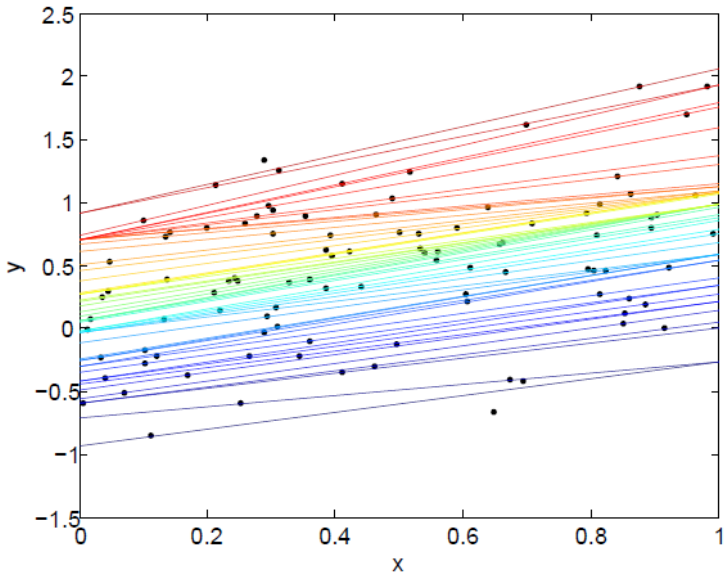
Nu benyttes metoden $Cross$, hvor koefficienterne, $\hat{\beta}$, beregnes for de ovennævnte fraktiler. De tilhørende fraktil-kurver ses af figur 9.2, hvor det tydeligt ses, at fraktil-krydsning forekommer.

Benyttes i stedet metoden NC_{single}, ses det af figur 9.3, at de tydelige fraktiler-krydsninger



Figur 9.2: Fraktil-kurverne beregnet via "*Cross*-metoden".

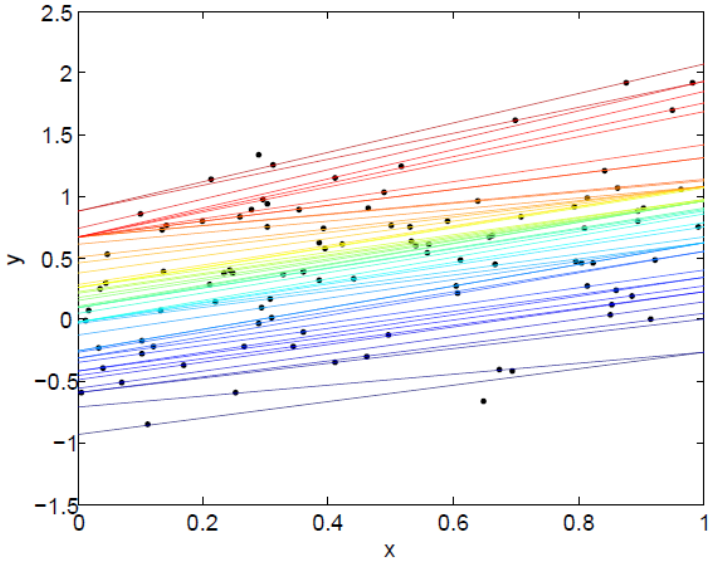
fra før nu er undgået, som forventet.



Figur 9.3: Fraktil-kurverne beregnet via "NC_{single}-metoden".

Benyttes metoden *Huge* fås fraktil-kurverne i figur 9.4, hvor det ligeledes ses, at fraktil-krydsningerne er undgået.

På figur 9.3 og 9.4 ser det ud til, at der, i dette tilfælde, ikke er særlig stor forskel



Figur 9.4: Fraktil-kurverne beregnet via "*Huge*-metoden".

	Sande	**Cross**	**NC_{Single}**	**Huge**
Træning	**737.4023**	726.9189	727.3831	727.1549
Test	**69716.0544**	70913.4744	70877.737	70833.150

Tabel 9.1: Det samlede tab for de tre metoder, samt tabet for de sande fraktiler. Tallene markeret med rødt angiver det højeste tab for metoderne, mens det laveste er markeret med blåt.

på metoderne NC_{single}, og *Huge*. For at se nærmere på hvilke fraktil-modeller der rent faktisk præsterer bedst, betragtes nu det samlede tab, når et testsæt benyttes, for hver af de tre modeller. Dvs. at fraktil-kurverne stadig er de samme, mens datasættet er nyt.

Testsættet er generet på samme måde som træningssættet, men dette nye sæt indeholder 10.000 datapunkter, eftersom et større testsæt giver større sikkerhed for, at modellen med det laveste tab ligner den sande fraktil-model mest. Tabene ses i tabel 9.1 sammen med tabene fra træningssættet.

Tallene i tabellen viser, at *Cross*-metoden giver det laveste tab, når træningssættet benyttes. Dette er som forventet, da fraktilernes placering ikke har nogen begrænsninger her, og dermed har fraktilerne bedre mulighed for at gøre tabet mindre. Det ses desuden, at *Huge* klarer sig bedre end de to øvrige metoder, når testsættet benyttes. Dette giver god mening, da fraktilerne ikke krydser her, og dermed bør ligne de teoretiske fraktiler mere end dem fra *Cross*-metoden. Desuden virker det fornuftigt, at *Huge* klarer sig en smule bedre end NC_{single}, da man i *Huge*-metoden ikke er afhængig af, hvilken fraktilkurve man starter med at estimere.

Ydermere ses det af tabellen, at de sande fraktiler er dårligere end de estimerede fraktiler for træningssættet, hvilket ikke er overraskende, da de estimerede fraktiler er beregnet

ud fra netop træningssættet. Det ses også, at de sande fraktiler klarer sig betydeligt bedre end de estimerede metoder på testsættet. Dette stemmer overens med vores forventning, da de sande fraktiler bør få det laveste tab, hvis testsættet er stort nok.

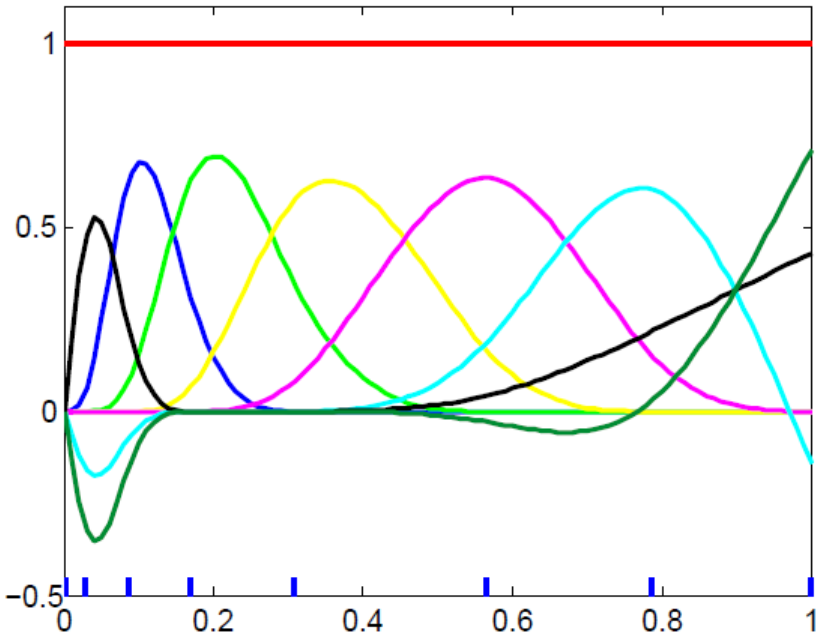
Dette simple eksempel illustrerer, hvordan non-crossing metoderne NC_{single} og $Huge$ undgår fænomenet med fraktil-krydsninger.

10 Fraktil-modeller for vindkraftdata

10.1 Introduktion

I følgende afsnit benyttes de implementerede funktioner, `CrossQuant`, `NonCrossSingle` og `NonCrossHuge`, til at sammenligne de fire metoder, $Cross$, NC_{single}, $NC_{singleNO}$ og $Huge$. Via disse metoder bestemmes koefficienterne, $\hat{\beta}$, som indgår i fraktil-modellerne. Ovenstående metoder benyttes alle på vindkraft-data fra Tunø Knob Offshore Wind Farm. Dette data indeholder målinger af den faktiske vindkraft-produktion, den prædikterede vindkraft-produktion samt tidshorisonten for prædiktionerne, der givet ud fra meteorologiske forudsigelser. Tidshorisonterne angiver, hvor langt tid der går, fra selve prædiktionen gives, til den faktiske produktion finder sted. Produktionerne er normaliserede, og ligger således i intervallet 0 til 1. Derfor er 0 og 1 hhv. den nederste og øverste grænse i non-crossing-metoderne.

Fælles for alle metoderne i dette afsnit er, at de estimerede fraktiler er naturlige splines, som er dannet via en linearkombination af 8 spline-basisfunktioner, som er beregnet i R. På figur 10.1 ses de 8 basisfunktioner.



Figur 10.1: Her ses de 8 spline-basisfunktioner. De blå mærker på førsteaksen angiver knudeplaceringer. Knudeplaceringer samt basisfunktioner varierer en smule afhængigt af datasættet.

Benyttes samme notation som i afsnit 3, ses det, at antallet af knudepunkter er $v + 1 =$

$7 + 1 = 8$, hvilket giver $v + 1 = 8$ basisfunktioner. Der skal altså bestemmes 8 koefficienter, én for hver af basisfunktionerne, for hver fraktil-kurve, hvorfor $\hat{\boldsymbol{\beta}}$ er en vektor med 8 elementer.

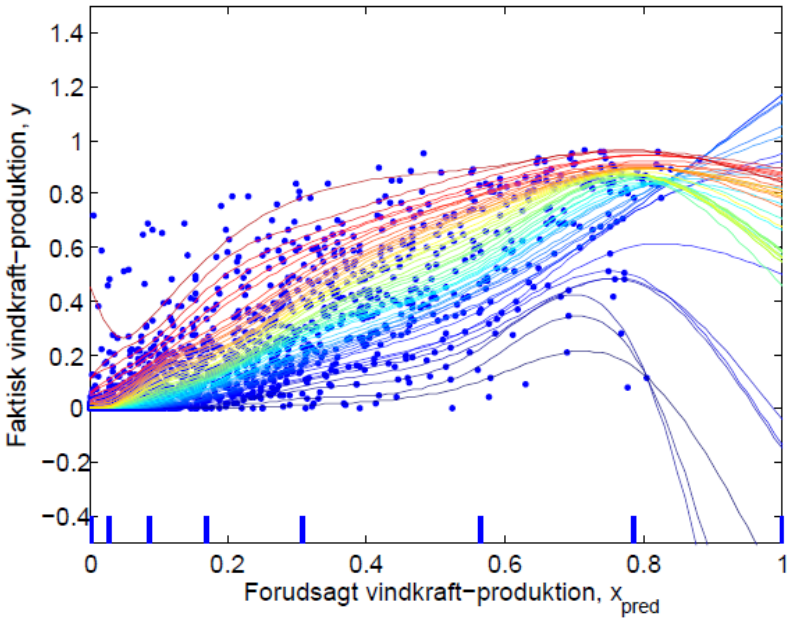
På figuren ses desuden placeringen af knudepunkterne; der er er 6 indre knudepunkter og 2 ydre, som ligger i 0 og 1. Antallet af knudepunkter er valgt ved *"trial and error"*. Ved anvendelsen af for få knudepunkter mangler kurverne fleksibilitet, og ved anvendelsen af for mange knudepunkter "over-fitter" kurverne, dvs. at der opstår absurde udsving, idet de forsøger at fitte tendenser, som ikke eksisterer [2].
De indre knudepunkter er placeret på en sådan måde, at 10% af datapunkterne ligger til venstre for den første knude, 30% til venstre for den anden, 50% for den tredje, 70% for den fjerde, 90% for den femte og 99% ligger til venstre for det sjette knudepunkt. Herved placeres flere knudepunkter i områder med meget data, men hvor der samtidig er sørget for, at der ikke er alt for store intervaller uden nogle knuder, hvorfor 99% er valgt.

I og med at knudepunkternes placeringer afhænger af, hvor data ligger, så ligger knudepunkterne ikke de samme steder for forskellige datasæt. Knudepunkterne med tilhørende basisfunktioner vist på figur 10.1 er for det datasæt, der har en tidshorisont på 24 timer.

10.2 Resultater

I dette afsnit præsenteres resultaterne fra de fire forskellige metoder, beskrevet i afsnit 8. Gennem hele afsnittet er horisonten valgt til 24 timer.

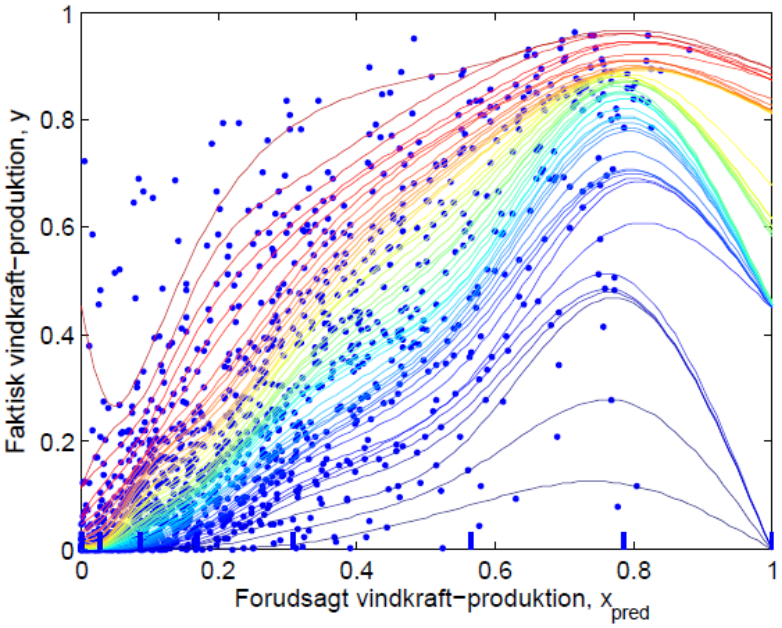
Først benyttes metoden *Cross*, med funktionen `CrossQuant`, hvor vi beregner koefficienterne $\hat{\beta}$, for 2%−fraktilen, 4%−fraktilen, 6%−fraktilen, osv. op til 98%−fraktilen. Således fås 49 forskellige fraktil-kurver for den betingede fordeling af vindkraft-produktionen givet en prædiktion. På figur 10.2 ses disse fraktilkurver samt det benyttede træningssæt bestående af 1498 datapunkter.



Figur 10.2: Fraktil-kurverne beregnet via *Cross*-metoden. Det ses tydeligt, at mange fraktilkrydsninger opstår. De blå mærker på førsteaksen angiver knudepunkterne.

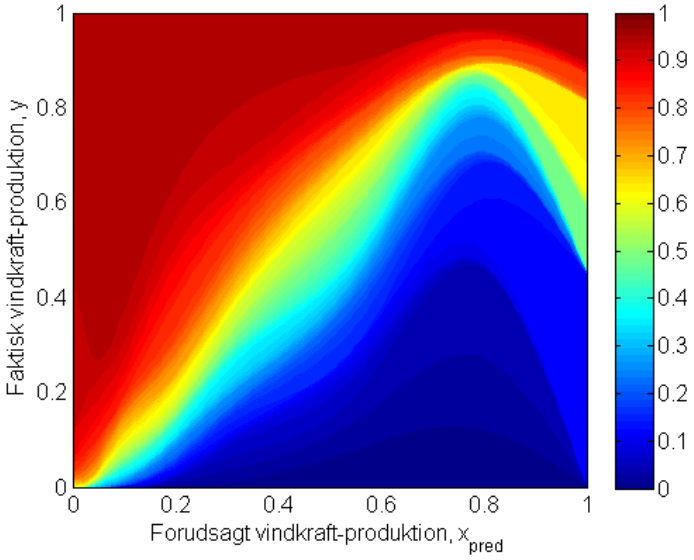
Af figuren ses det, at fraktilerne både krydser hinanden, og antager negative værdier, hvilket ikke er optimalt, da det hverken giver mening, at fraktilerne krydser, eller at en produktionen kan blive negativ.

Nu benyttes i stedet metoden, NC_{single}, til at beregne de ovennævnte fraktiler. Dette gøres vha. funktionen `NonCrossSingle`. På figur 10.3 ses disse fraktilkurver samt træningssættet.
Denne figur viser, som forventet, at de tydelige krydsninger fra før er undgået. Desuden antager fraktil-kurverne nu kun værdier mellem 0 og 1, hvilket også er forventet grundet de indførte grænsebetingelser i metoden. Fraktil-kurverne beregnet via denne metode, giver altså umiddelbart mere mening, end fraktil-kurverne på figur 10.2.
På figur 10.4 ses den kumulative fordelingsfunktion for den betingede fordeling af Y, hvis



Figur 10.3: Fraktil-kurverne beregnet via NC_{single}-metoden. Fraktil-krydsningerne er her undgået. De blå mærker på førsteaksen angiver knudepunkterne.

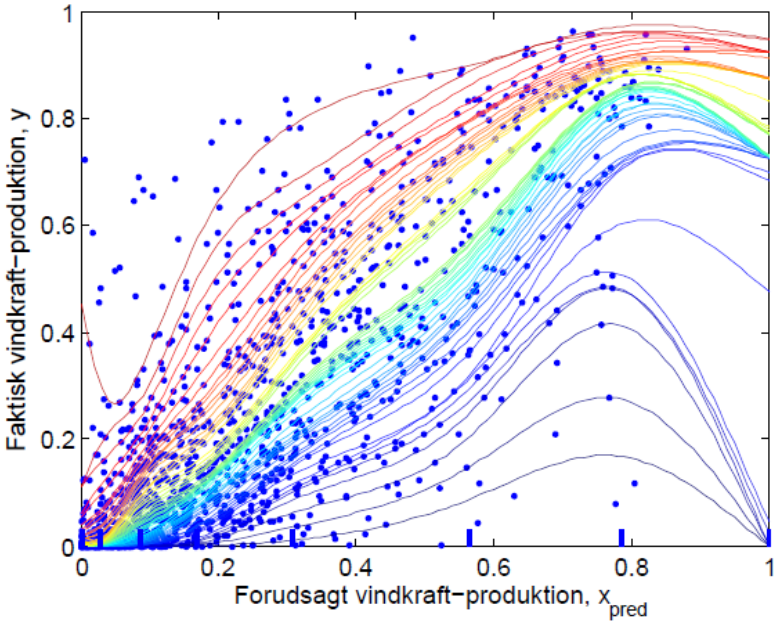
man følger farveskalaen, som går fra 0 til 1.



Figur 10.4: Denne kumulative fordelingsfunktion for den betingede fordeling af Y, beregnet via NC_{single}-metoden. Fraktil-krydsninger er her undgået.

På denne figur fornemmes det også, at der ikke er fraktil-krydsninger. Endvidere ses det, at modellen forventer mange målinger i det område, der er gult/grønt, idet farven her skifter hurtigt, hvilket betyder, at den kumulative fordelingsfunktion er stejl.

Hernæst benyttes metoden $NC_{singleNO}$ til at estimere fraiktil-kurverne for det samme datasæt. Disse kurver ses på figur 10.5.

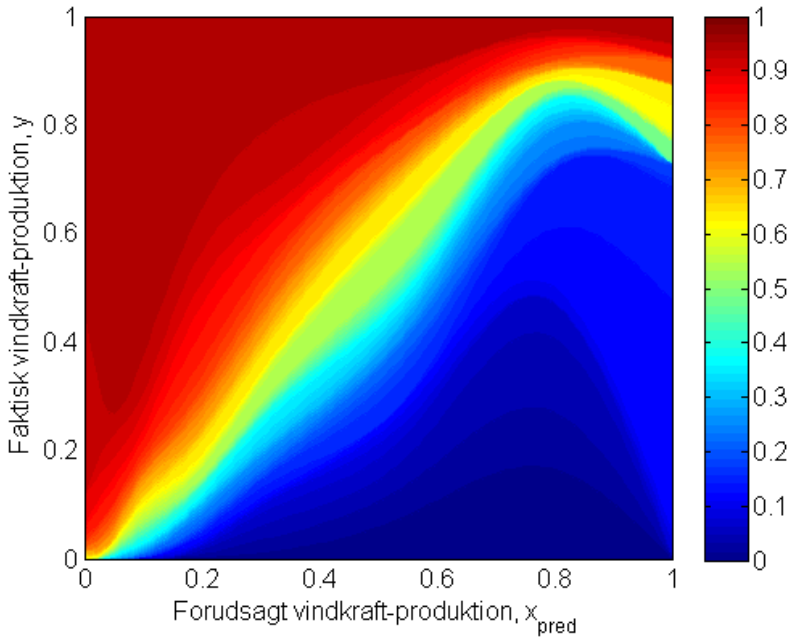


Figur 10.5: Fraktil-kurverne beregnet via $NC_{singleNO}$-metoden. Fraktil-krydsninger er her undgået. De blå mærker på førsteaksen angiver knudepunkterne.

Af figuren ses det, at fraktil-kurverne ligner kurverne i figur 10.3, med undtagelse af nogle af kurverne, som nu er skubbet en anelse opad i højre side. Når fraktilerne beregnes fra midten og ned, bliver de nederste fraktiler presset nedad af de midterste fraktiler, da de jo ikke må krydse hinanden. Når man efterfølgende beregner nedefra og op, har de nederste fraktiler nu mulighed for at at skubbe sig opad igen, og det må formodes at være årsagen til, at nogle af fraktilerne ligger længere oppe, når $NC_{singleNO}$-metoden benyttes. I afsnit A.9 ses to plot; et af de estimerede fraktil-modeller beregnet nedefra og op, og et andet for de estimerede fraktil-modeller beregnet oppefra og ned.

Ydermere set det, at nogle af kurverne i midten ligger meget tæt, hvilket umiddelbart ikke virker fornuftigt, da der dermed opstår store mellemrum mellem de øvrige midterfraktiler, på trods af de mange datapunkter i mellem dem. Det er svært at sige, hvorfor dette underlige fænomen opstår, men det kunne tyde på, at det altså har en stor betydning, hvilken rækkefølge fraktil-modellerne estimeres i.

I figur 10.6 tegnes nu den kumulative fordelingsfunktion for den betingede fordeling af Y, beregnet via fraktil-kurverne i figur 10.5.

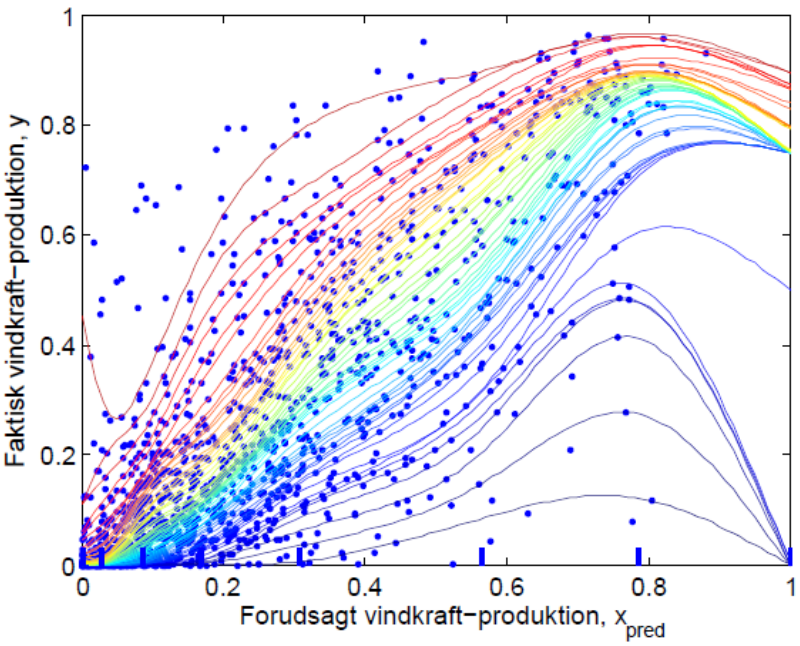


Figur 10.6: Denne kumulative fordelingsfunktion for den betingede fordeling af y, beregnet via $NC_{singleNO}$-metoden.

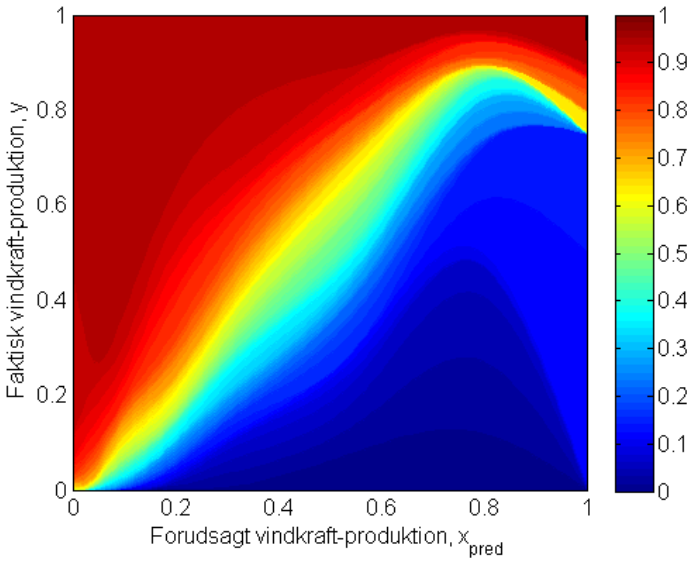
Dette plot ligner meget plottet på figur 10.4, men nu er det gule område til højre skubbet opad, hvilket har en stor betydning, for den kumulative fordelingsfunktion for Y, givet de højeste prædiktioner. Desuden ses det, at området med de gul/grønne farve ser lidt anderledes ud, netop på grund af det underlige mellemrum mellem de midterste fraktiler, beskrevet ovenfor.

Nu benyttes den sidste metode, $Huge$, til at estimere fraktil-kurverne. Denne metode beregner fraktilerne vha. funktionen `NonCrossHuge`. På figur 10.7 ses de estimerede fraktil-kurver.
Disse fraktil-kurver ligner umiddelbart fraktil-kurverne i figur 10.3, men i denne figur ses det, at nogle af fraktilerne er løftet opad i højre side. Dermed kunne de, ved første øjekast, ligne dem i figur 10.5, men fraktil-kurverne, beregnet via $Huge$, er pænt fordelt hen over midten. Når man benytter $Huge-$metoden, bliver de nederste fraktiler ikke nødvendigvis skubbet ned af de midterste fraktiler, da alle fraktilerne beregnes samtidig. Dette har den fordel, at man ikke behøver at fastlægge, eksempelvis, 50%−fraktilen på forhånd, og risikere at lade denne påvirke resten af fraktilerne negativt. Derved har de nederste fraktiler nu mulighed for at lægge sig lidt højere oppe, hvis det er til fordel for tabsfunktionen. Alt i alt ser fraktil-kurverne på figur 10.7 meget rimelige ud, da de ikke krydser hinanden, men er jævnt fordelt, og da der ikke opstår pludselige afbøjninger i højre side.

På figur 10.8 ses den kumulative fordelingsfunktion for den betingede fordeling af Y, beregnet via fraktil-kurverne i figur 10.7.
For bedre at kunne vurdere forskellene i resultaterne fra de anvendte metoder, betragtes



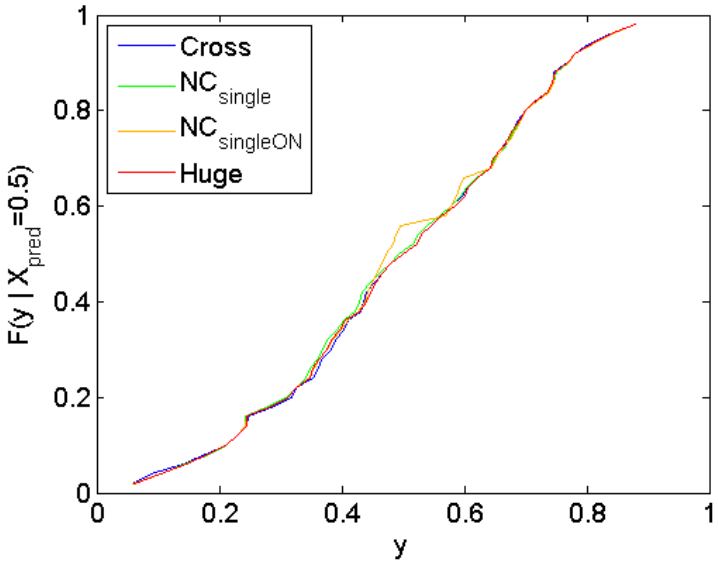
Figur 10.7: Fraktil-kurverne beregnet via *Huge*-metoden, uden fraktil-krydsninger. De blå mærker på førsteaksen angiver knudepunkterne.



Figur 10.8: Denne kumulative fordelingsfunktion for den betingede fordeling af Y, beregnet via *Huge*−metoden.

nu den kumulative fordeling af Y givet forskellige værdier af x_{pred}. I figur 10.9 ses de

kumulative fordelingsfunktioner, givet $X_{pred} = 0.50$, for de fire forskellige metoder.



Figur 10.9: De kumulative fordelingsfunktioner for den betingede fordeling af Y, givet $X_{pred} = 0.50$.

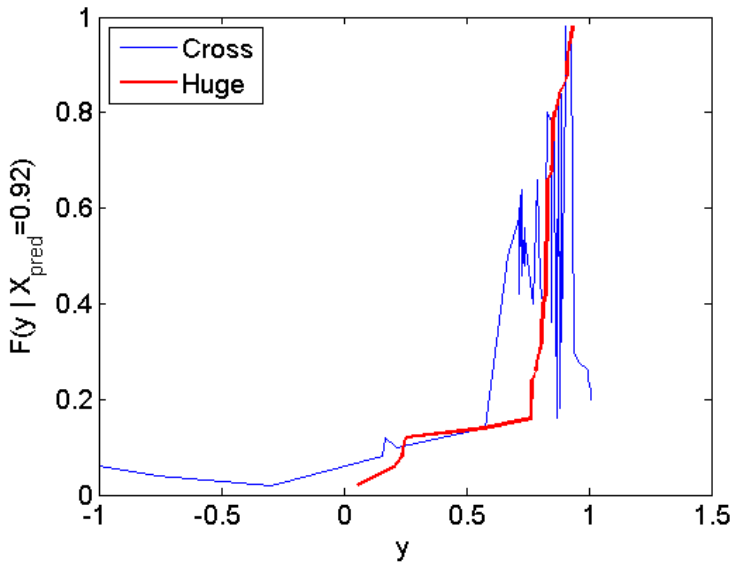
Figuren viser, at der for $X_{pred} = 0.5$, er meget lille forskel på de forskellige fraktil-kurver, ud over at grafen for $NC_{singleNO}$-metoden har nogle anderledes udsving, dér hvor fraktilerne ligger meget tæt.

Nu betragtes i stedet figur 10.10, som viser den kumulative fordelingsfunktion, givet $X_{pred} = 0.92$, for $Cross$ og $Huge$. Her ses det, at der er stor forskel på fraktilerne for større værdier af X_{pred}.
Det er tydeligt, at der er mange krydsninger med $Cross-$metoden, da kurven stiger og aftager meget pludseligt. At kurven aftager i en kumulativ fordelingsfunktion giver ingen mening, da det indikerer, at der er negative sandsynligheder, hvilket selvfølgelig ikke giver statistisk mening. Dette understreger problematikken omkring fraktil-krydsninger.

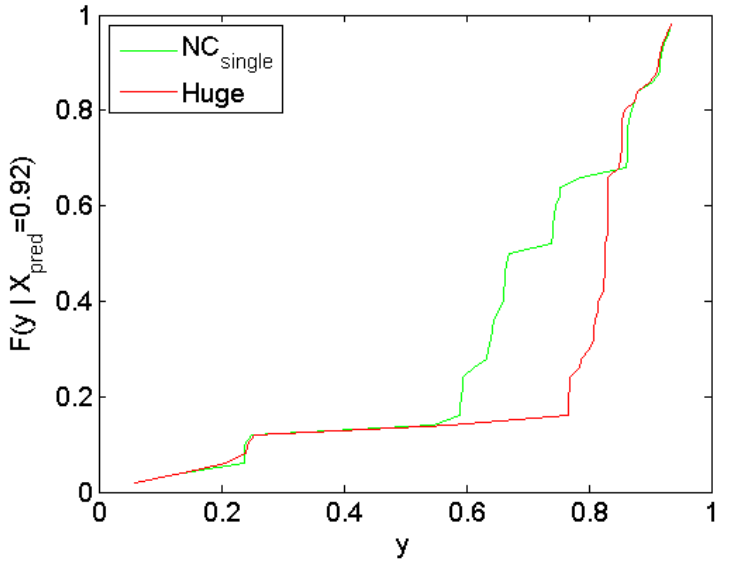
Derimod ses det, at kurven for $Huge$-metoden vokser i hele intervallet, når Y stiger, hvilket indikerer, at der ikke opstår fraktil-krydsninger.

For at se nærmere på forskellen mellem NC_{single} og $Huge$, betragtes nu den kumulative fordelingsfunktion, givet $X_{pred} = 0.92$, for disse to metoder (fig. 10.11).
På trods af de mange ligheder imellem fraktil-kurverne i figurene 10.3 og 10.7, er der alligevel stor forskel på den kumulative fordelingsfunktion for Y, for specielt de højde værdier af X_{pred}.

I det følgende undersøges der, hvilke modeller der præsterer bedst på et sæt testdata bestående af 1496 datapunkter (se plot heraf i afsnit A.10). Hertil benyttes de fraktilmodeller, vi har beregnet via træningsdata, præsenteret ovenfor. Disse modeller testes på



Figur 10.10: De kumulative fordelingsfunktioner for den betingede fordeling af Y, givet $X_{pred} = 0.92$.



Figur 10.11: De kumulative fordelingsfunktioner for den betingede fordeling af Y, givet $X_{pred} = 0.92$.

test-datasættet, dvs. at fraktil-kurverne stadig er de samme, mens datasættet er nyt. Det samlede tab for testsættet beregnes for alle fire metoder, og resultatet ses i tabel 10.1, hvor tabet for træningssættet desuden er vist.

	Cross	**NC$_{\text{Single}}$**	**NC$_{\text{SingleNO}}$**	**Huge**
Train	2764,6940	2766,5311	2769,0355	2765,5780
Test	2279,9874	2280,8322	2282,6399	2279,0048

Tabel 10.1: Det samlede tab med hhv. træningssæt og testsæt for de fire metoder. Det laveste tab i hver række er markeret med blå skrift.

Ud fra tabellen ses det altså, at tabet for træningssættet er lavest med $Cross-$metoden, hvilket er som forventet, da fraktilernes placering ikke har nogen begrænsninger her, og dermed har fraktilerne bedre mulighed for at gøre tabet mindre for træningssættet.

Det ses desuden, at tabet for testsættet er mindst for $Huge-$metoden, hvilket giver god mening, da fraktilerne ikke krydser, og dermed bør disse fraktiler ligne de rigtige fraktiler mere end fraktil-kurverne for $Cross-$metoden. Det virker desuden fornuftigt at tabet er lidt mindre for $Huge$ end for NC_{Single}, idet man i $Huge-$metoden ikke er afhængig af hvilken fraktil-kurve man starter med at estimere.

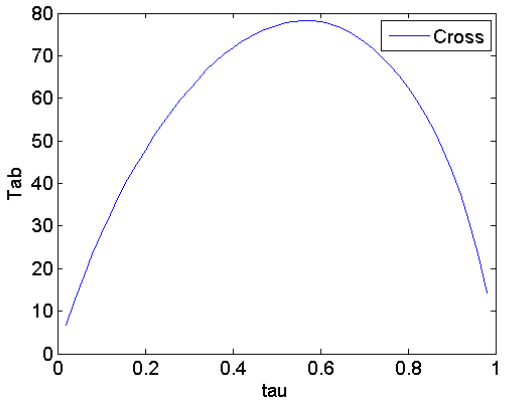
Det bør dog bemærkes, at forskellen mellem de forskellige tab er meget lille i forhold til størrelsen af de samlede tab. Det skyldes højst sandsynligt, at der ikke er ret mange datapunkter, der hvor der virkelig er stor forskel på fraktil-kurverne. Derfor kan det godt være tilfældigt, hvilke modeller der viser sig, at have det laveste tab. Det bør dog noteres her, at selv hvis der ikke er den store forskel i tabsfunktionerne, så har de modeller, der undgår krydsninger, stadig en stor fordel, i og med at de rent faktisk giver mening rent statistisk.

Figur 10.12 viser en kurve over tabet for hver fraktil for $Cross$ med træningsdata (figur 10.12a), samt kurver over forskellen mellem tabet for $Cross$ og tabet for hhv. NC_{Single}, $NC_{SingleNO}$ og $Huge$ (figur 10.12b). Da forskellen er angivet som tabet for hhv. NC_{Single}, $NC_{SingleNO}$ og $Huge$ trukket fra tabet for $Cross$, betyder en positiv afvigelse altså, at tabet for den valgte non-crossing-model er større end for $Cross$. Figuren viser, at tabet, som forventet, er større for alle non-crossing-modeller end for $Cross$ på træningsdata. Desuden ses det, at kurven for $NC_{SingleNO}$ antager de højeste værdier, og dette gøres omkring de midterste fraktiler-kurver, da disse, som tidligere beskrevet, ligger meget tæt.

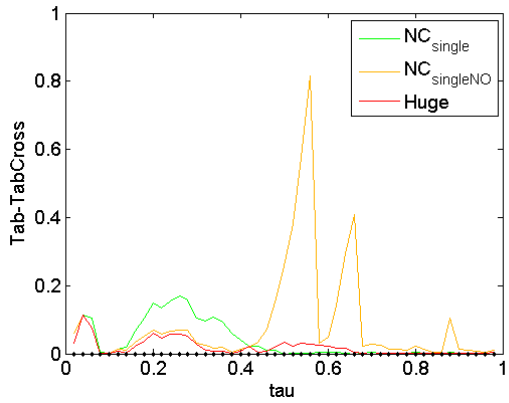
I figur 10.13 se en tilsvarende sammenligning af tabet for metoderne, dog under anvendelsen af testdata. På disse kurver ses det, at $Huge$ har et lavere tab en $Cross$ for langt de fleste fraktiler. Desuden tyder det på, at $Huge$ klarer sig bedre end de øvrige non-crossing-modeller for næsten alle fraktiler på dette testdata.

Opdeling af datapunkter

Det undersøges nu, hvor gode de forskellige fraktil-modeller rent faktisk er til at dele punkterne i træningssættet og testsættet op, da dette selvsagt er en vigtig egenskab inden for fraktil-regression.

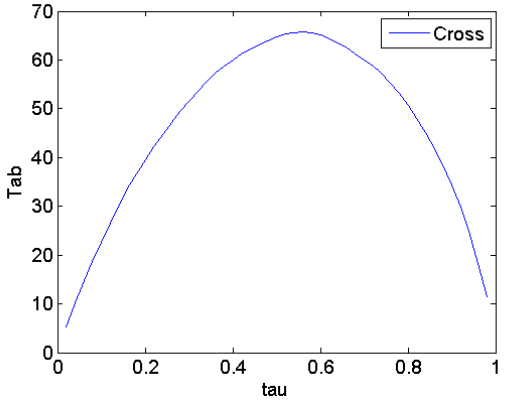


(a) Tabet for *Cross* med træningsdata.

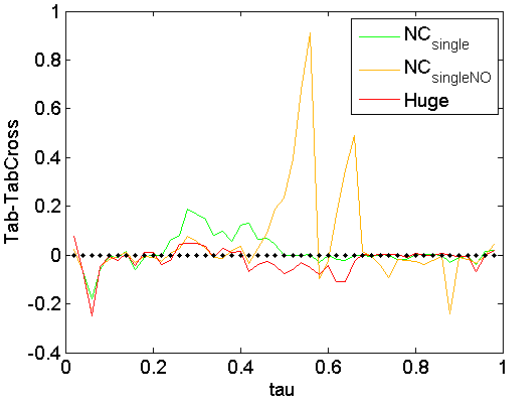


(b) Forskellen mellem tabet for *Cross* og tabet for de tre andre metoder for træningsdata.

Figur 10.12: Sammenligning af tabet for de forskellige non-crossing-metoder, for hver fraktilkurve med træningsdata.



(a) Tabet for *Cross* med testdata.



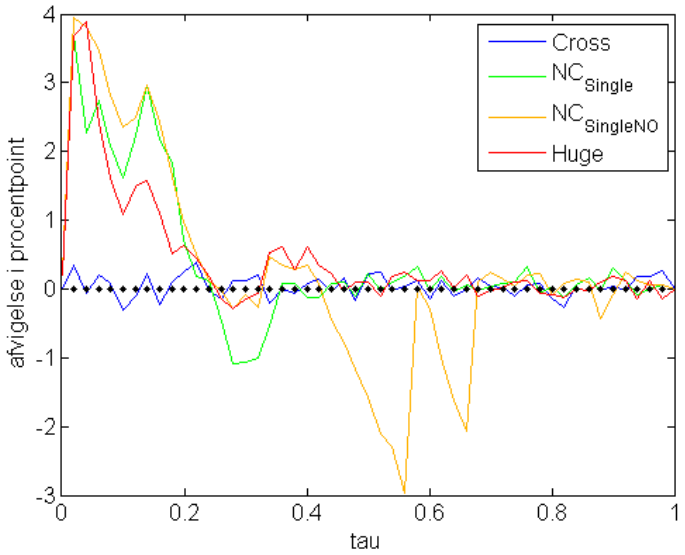
(b) Forskellen mellem tabet for *Cross* og tabet for de tre andre metoder for testdata.

Figur 10.13: Sammenligning af tabet for de forskellige non-crossing-metoder, for hver fraktilkurve med testdata.

Punkter der er placeret oven på en fraktil-kurve[9], er opdelt således, at halvdelen er punktet tildeles området under kurven, mens den anden halvdel tildeles området over kurven. Ligger et punkt oven på flere kurver, deles punktet op imellem hver af fraktilerne. Hvis et punkt således ligger oveni to kurver, hører en tredjedel af punktet til området under den første kurve, en anden tredjedel af punktet hører til området mellem de to kurver, og den sidste tredjedel hører til området over den sidste kurve.
Figur 10.14 viser afvigelsen i opdelingen af træningsdata, for de fire forskellige metoder.

[9]Eller punkter, som ligger meget tæt på: tolerance = 10^{-6}.

En positiv afvigelse betyder, at der er for mange punkter under fraktil-kurven, og en negativ afvigelse betyder, at der er for få.



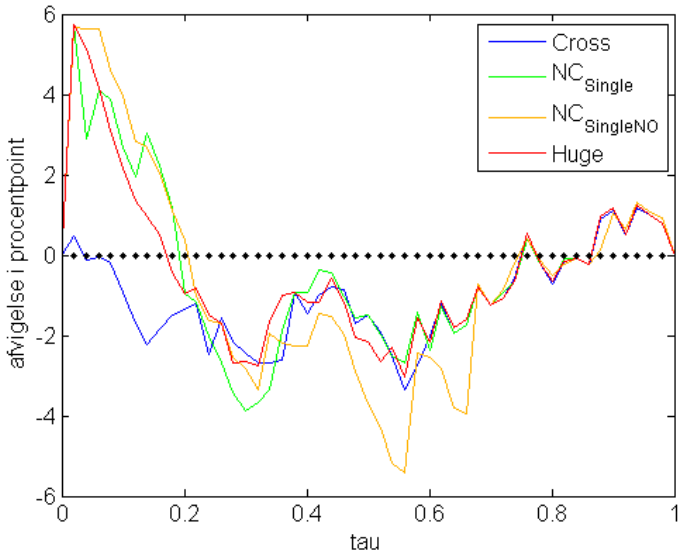
Figur 10.14: Afvigelsen i opdelingen af træningsdata for de fire metoder.

Kurven for *Cross*-metoden viser, at denne metode tilnærmelsesvist deler træningssættet korrekt op. Dette er forventet, da koefficienterne, $\hat{\beta}$, i denne fraktil-model er fundet som løsningen til ligning (2.9) i afsnit 2, uden nogen tilføjede begrænsninger. *Huge* og NC_{single} klarer sig begge fint, undtagen ved de første fraktiler. At de har problemer hér kan skyldes, at mange y-værdier er lig 0 (ca. 90) i dette datasæt, hvilket besværliggør opdelingen af punkterne grundet de indførte non-crossing-betingelser.
Desuden klarer $NC_{singleNO}$ sig dårligt ved de midterste fraktiler, hvor kurverne ligger meget tæt.

Figur 10.15 viser afvigelsen i opdelingen af testdata, for de fire forskellige metoder.
Det ses, at der ikke er den store forskel på hvor godt *Cross*, NC_{single} og *Huge* klarer sig, udover at *Cross* er lidt bedre i starten, hvilket nok igen skyldes antallet af y-værdier, der er 0.

Ud fra analysen af resultaterne i dette afsnit, ser det umiddelbart ud til, at *Huge* er den non-crossing-model, der klarer sig bedst. Idet vi kun har testet for 24-timers-horisonten, og da forskellen på tabene er meget lille set i relation til størrelsen af det samlede tab, er det svært at komme med en sikker konklusion på, om *Huge*-metoden virkelig giver en bedre model end NC_{single}-metoden gør. Dog har *Huge*, som sagt, den klarer fordel, at man ikke behøver at bekymre sig om, hvilken fraktil man bør starte med.

I forsøget på at komme nærmere en konklusion på, hvilken metode der er bedst, undersøges det, hvordan modellerne klarer sig for andre tidshorisonter.

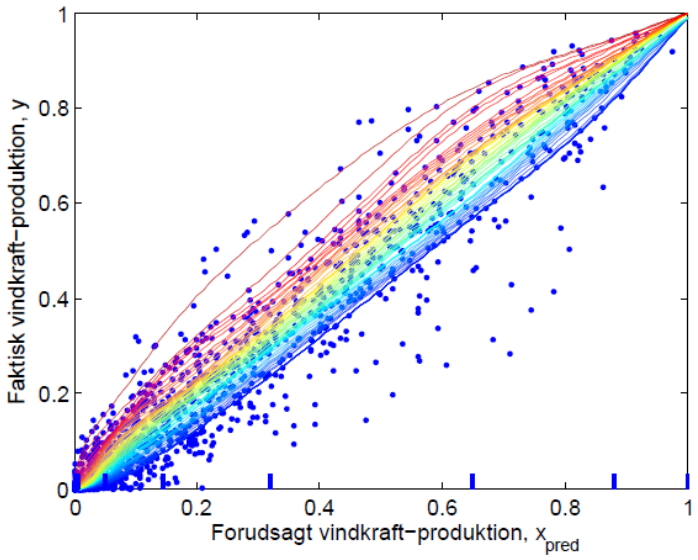


Figur 10.15: Afvigelsen i opdelingen af testdata for de fire metoder.

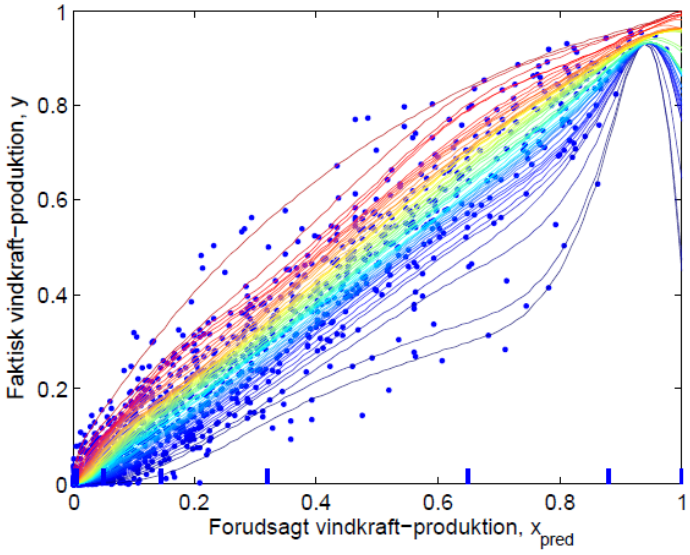
10.3 Resultater for flere tidshorisonter

I dette afsnit testes de tre metoder, *Cross*, NC_{single}, og *Huge* på flere forskellige tidshorisonter. Den fjerde metode, $NC_{singleNO}$, er udeladt her, da det var den non-crossing metode, der præsterede dårligst i ovenstående afsnit.

Metoden NC_{single} benyttes her på træningssættet, der har en horisont på 1 time. Figur 10.16 viser resultatet, som er meget overraskende.
Graferne for de nedre fraktiler er sammenfaldende, i modsætning til de jævnt fordelte øvre fraktiler. Det er svært at sige, hvad dette skyldes. Årsagen kan være et numerisk problem, men det kan også være, at den anvendte naturlige spline-basis ikke egner sig til denne situation. Derfor forsøger vi nu i stedet at fitte med almindelige kubiske splines i stedet for naturlige kubiske splines. Til dette formål beregnes, vha. R, en spline-basis for disse nye splines, og NC_{single} benyttes igen på træningssættet med en tidshorisont på 1 time, se figur 10.17.
Det ses nu, at det er lykkedes fraktil-kurverne at fordele sig bedre denne gang. Det kunne derfor godt tyde på, at vi stiller for strenge krav til fraktilerne, ved at bruge naturlige kubiske splines. Ved brugen af de nye almindelige kubiske splines, ses dog den effekt vi prøvede at undgå ved at benytte naturlige splines; nemlig at fraktil-kurverne har meget store udsving ved yderpunkterne. Endvidere har der været den ulempe ved alm. kubiske splines, at udsvingene i yderpunkter, kan give en 50%−fraktil, som overskrider grænserne $y = 0$ og $y = 1$, hvorfor denne fraktil ikke er et gyldigt begyndelses-estimat for `NonCrossSingle`, som herved ikke kan producere et gyldigt startgæt for `NonCrossHuge`. Da de naturlige splines ikke fungerer i dette tilfælde, med disse knudepunkter, kan det dog være vejen frem at benytte almindelige kubiske splines. Et alternativ kunne være at eksperimentere med knudeplaceringerne samt med antallet af disse. Tiden er dog desværre en



Figur 10.16: Fraktil-kurver for NC_{single}-metoden, hvor horisonten er lig med 1 time, med naturlige splines.



Figur 10.17: Fraktil-kurver for NC_{single}-metoden, hvor horisonten er lig med 1 time, med almindelige kubiske splines i stedet for naturlige splines.

begrænsende faktor mht. dette projekt, så i stedet fortsætter vi med de naturlige splines og nøjes med at betragte de tilfælde, hvor disse splines kan benyttes på tilfredsstillende vis.

Derfor findes i stedet nogle horisonter, for hvilke fraktil-kurverne formår at fordele sig

nogenlunde fornuftigt, frem for at lægge sig oveni hinanden. Det samlede tab, for disse horisonter, for hver af de tre metoder, ses i tabel 10.2.

Horisont	Cross	NC_{Single}	Huge
16 timer	2077.7330	2077.1926	2077.3836
17 timer	2081.3354	2079.5309	2079.9580
18 timer	2106.7783	2104.2969	2104.7116
19 timer	2162.8896	2162.2449	2162.0647
20 timer	2197.8682	2195.4326	2196.2376
21 timer	2250.6051	2250.4963	2249.7381
24 timer	2279.9874	2280.8322	2279.0048
25 timer	2386.1071	2384.8179	2384.4761
27 timer	2445.5291	2450.2842	2445.0874

Tabel 10.2: Det samlede tab med testsæt for de tre metoder for forskellige horisonter. Det højeste tab i hver række er markeret med rød skrift og det laveste med blå.

Det ses i tabellen, at det bestemt ikke er konsistent, hvilken metode der er bedst. Umiddelbart det lader til, at *Cross* generelt præsterer dårligst, mens det er lidt forskelligt om NC_{single} eller $Huge$, der klarer sig bedst. Der tegner sig et billede af, at NC_{single} er bedre ved de lave horisonter, mens $Huge$ er bedre ved de højere. Det er svært at sige, om dette er det rigtige billede, eller om det er et tilfælde. Det skal igen bemærkes, at der er meget lidt data for de høje prædiktioner, og det er netop her, der er stor forskel på modellerne. Så derfor kan der være en del usikkerhed mht., hvilken model der har det laveste tab for testdata. Det er dog igen vigtigt at påpege, at de to non-crossing modeller altid vil have en stor fordel, idet de giver mening rent statistisk, i forhold til *Cross*, lige meget hvad tabsfunktionerne viser.

10.4 Beregningstider og mulige forbedringer

I det følgende afsnit vil vi forsøge at klarlægge nogle af de mulige forbedringer for `NonCrossHuge`, som dog ikke er blevet videre undersøgt grundet den begrænsede tidsramme vi er underlagt. Det er nogle forbedringer i forhold til minimering af beregningstiden. I afsnittet betragtes kun `NonCrossHuge`, idet vi mener, at metoden har et stort potentiale, men at beregningstiden er lang. Her under ses en tabel for de implementerede funktioners beregningstid under anvendelse af 24-timers horisonten:

	Cross	**NC$_{\text{Single}}$**	**NC$_{\text{SingleNO}}$**	**Huge**
Samlet køretid	26.32 min	5.01 min	14.34 min	34.60 min
Iterationer	43259	3965	11482	2645

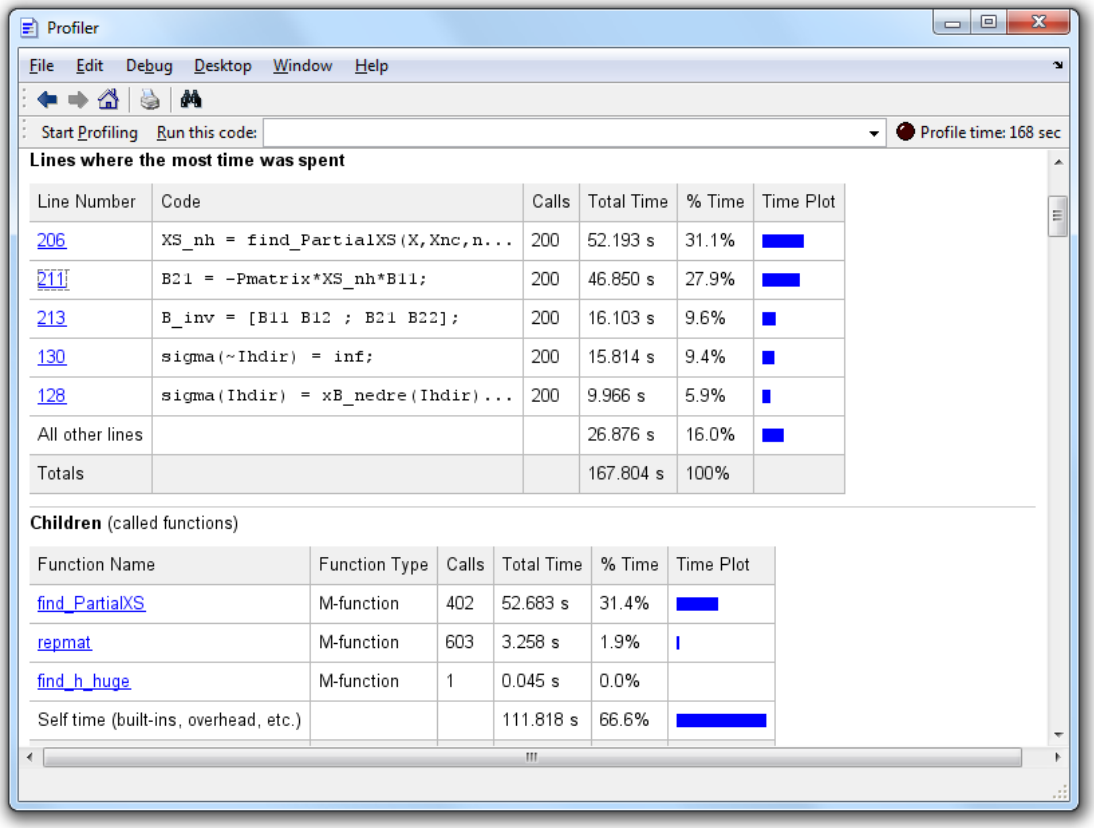
Tabel 10.3: Den samlede køretid samt antallet af iterationer for de forskellige metoder for 24-timers horisonten.

Ikke overraskende er beregningstiden for `Cross` lang, idet alle fraktiler bestemmes ud fra samme startgæt. Beregningstiden for `NonCrossSingleNO` er længere end for `NonCrossSingle`, idet sættet af fraktiler beregnes tre gange i førstnævnte.
Funktionen, `NonCrossHuge`, har, grundet problemets størrelse, den længste beregningstid. For er identificere potentielle forbedringsområder benyttes MATLAB's *profiler*, for at lokalisere eventuelle flaskehalse. *Profileren* udpeger de mest tidskrævende dele af den skrevne kode, som derefter undersøges nøje. I det følgende identificeres mulige forbedringsområder for `NonCrossHuge`, og til formålet benyttes datasættet for 24-timers horisonten. I figur 10.18 herunder ses et skærmprint af MATLAB-profileren for `NonCrossHuge`.

Det ses, at den samlede køretid efter 200 iterationer er 167,804 sek., hvoraf 52,193 sek., svarende til 31,1% af den samlede tid, benyttes til konstruktionen af matricen, $\mathbf{XS}(\overline{h})$ (`XS_nh`), i linje 206. Her under ses strukturen af $\mathbf{XS}(\overline{h})$:

$$\mathbf{XS}(\overline{h}) = \begin{bmatrix} \mathbf{X}(\overline{h}_{\tau_1}) & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \mathbf{X}(\overline{h}_{\tau_2}) & \mathbf{0} & \vdots \\ \vdots & \mathbf{0} & \ddots & \vdots \\ \mathbf{0} & \vdots & \mathbf{0} & \mathbf{X}(\overline{h}_{\tau_n}) \\ \mathbf{X}_{nc}(\overline{h}_{nc,\tau_1}) & \mathbf{0} & \vdots & \mathbf{0} \\ \mathbf{0} & \mathbf{X}_{nc}(\overline{h}_{nc,\tau_2}) & \mathbf{0} & \vdots \\ \vdots & \mathbf{0} & \ddots & \vdots \\ \mathbf{0} & \dots & \mathbf{0} & \mathbf{X}_{nc}(\overline{h}_{nc,\tau_n}) \end{bmatrix}, \tag{10.1}$$

Til konstruktionen af $\mathbf{XS}(\overline{h})$ benyttes, som tidligere nævnt, funktionen `find_PartialXS` (afsnit A.8). Strukturen i matricen kræver en stor mængde bogholderi, idet der beregnes, hvilke fraktiler elementerne i $\overline{h}$ knytter sig til, og hvor mange der knytter sig til



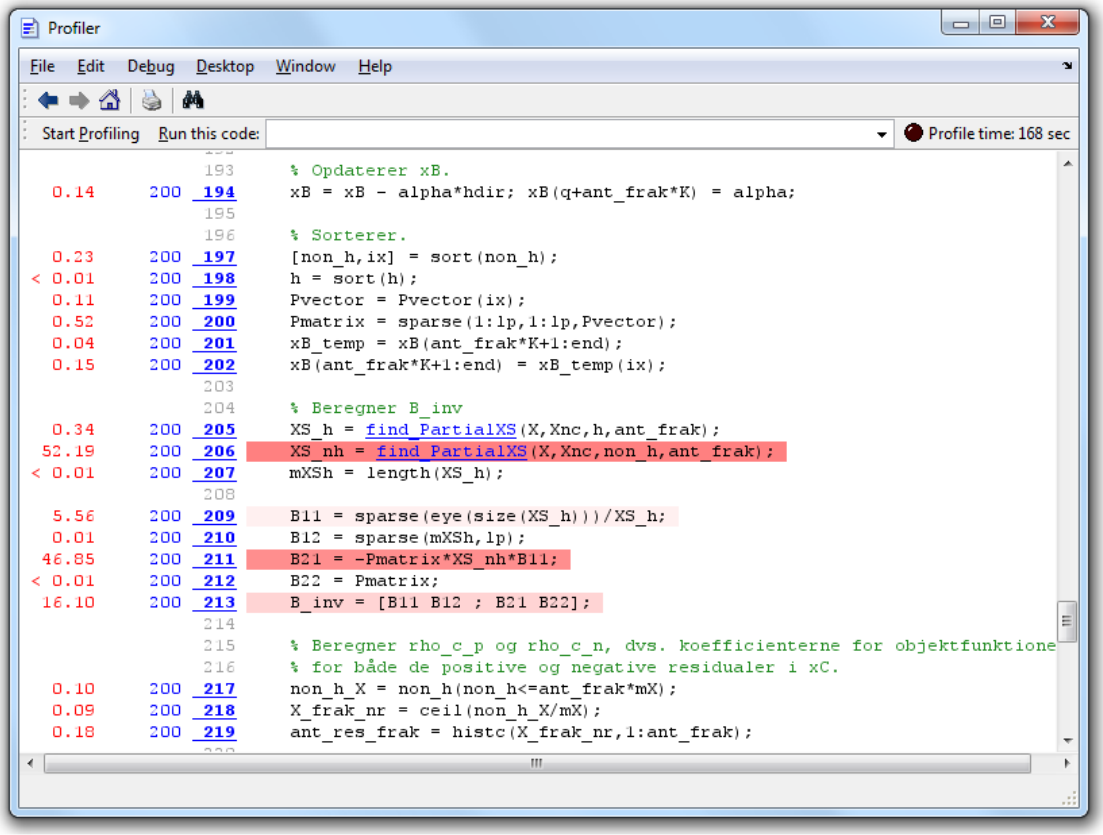
Figur 10.18: MATLAB-profiler for `NonCrossHuge` efter 200 iterationer. Her er datasættet for 24-timers horisonten benyttet.

hver af disse. Der bestemmes for hver fraktil, hvor mange af disse elementer, som knytter til residualerne, $\mathbf{r}$, og hvor mange der knytter sig til slackvariablene for non-crossing-betingelserne, $\mathbf{r}_{nc}$. Endvidere tages der højde for indrykningen og forskydningen af de enkelte $\mathbf{X}(\overline{h})$'er mm. Meget af beregningstiden bruges her, hvorfor implementeringen af `find_PartialXS` bør genovervejes, eller en alternativ opsætning af $\mathbf{XS}(\overline{h})$ bør benyttes i forsøget på at mindske mængden af bogholderi, og derved mindske beregningstiden.

En tilsvarende mængde tid benyttes til at beregne matricen **B21**, som er et produkt af diagonalmatricen, **PS**, $\mathbf{XS}(\overline{h})$ og $\mathbf{B}_{11} = \mathbf{XS}(h)^{-1}$. Her skyldes den lange beregningstid matrix-multiplikationen med $\mathbf{XS}(\overline{h})$ og $\mathbf{B}_{11} = \mathbf{XS}(h)^{-1}$, idet disse matricer med en tæthed over 5% egentlig ikke er velegnede som sparse matricer i MATLAB, jvf. afsnit 7.1. En mulig løsning kan være at omstrukturere $\mathbf{XS}(\overline{h})$ og/eller udtrykke denne ved mindre matricer, således at en sparse lagring kan undgås. Det skal igen understreges, at disse nævnte tiltag ikke er forsøgt implementeret.

Ganske overraskende tager det ingen nævneværdig tid at beregne selve matricen $\mathbf{B}_{11} =$

$\mathbf{XS}(h)^{-1}$ (kodelinje 209), hvor $\mathbf{XS}(h)$, her, har størrelsen 392×392 med samme struktur som $\mathbf{XS}(\overline{h})$. I figur 10.19, ses den samlede beregningstid for B_{11} efter 200 iterationer, som er på små 5.56 sek, svarende til 3.31% af den samlede beregningstid.



Figur 10.19: MATLAB-profiler for `NonCrossHuge` efter 200 iterationer. Her er datasættet for 24-timers horisonten benyttet. Kodelinjerne markeret med rødt, er de mest tidskrævende, og linjen markeret med lyserødt er den næstmest tidskrævende.

11 Konklusion

I denne rapport er der gjort rede for, hvorledes fraktil-regression kan formuleres som et lineært optimeringsproblem, og hvordan dette problem kan løses via Simplex-metoden. En simpel model til at estimere fraktil-kurver enkeltvist er blevet opstillet, hvilket muliggjorde en implementering i MATLAB af Simplex-algoritmen med udgangspunkt i denne model. Ved benyttelsen af algoritmen, risikerer man dog, at det uhensigtsmæssige fænomen med fraktil-krydsninger opstår, da fraktil-kurverne estimeres uafhængigt af hinanden.

Modellen blev derfor udvidet, således at 50%−fraktilen bruges som begyndelses-fraktil, hvorefter nabofraktilerne beregnes en efter en med den betingelse, at de ikke må krydse den forrige fraktil. Dette betyder, at ingen fraktiler krydser hinanden, og derved undgås det åbenlyse problem med den første model. Ulempen ved denne model er dog, at resultatet afhænger af, hvilken fraktil der fastlægges først, da de øvrige fraktiler skal tilpasse sig denne for at undgå krydsninger.

Dette motiverer til endnu en modeludvidelse, der gør det muligt, at beregne alle de ønskede fraktil-kurver samtidigt, uden at de krydser. Derved undgås problematikken angående fraktil-krydsninger, og resultatet afhænger ikke længere af, hvilken fraktil der fastlægges først. For at muliggøre implementeringen af denne model, var det nødvendigt at inddrage viden omkring sparse matricer i MATLAB, da problemet er for stort til at blive håndteret med almindelige matricer. Den klare ulempe ved denne store model er, at beregningstiden, på trods af brugen af sparse-funktioner, klart overstiger beregningstiden for den første non-crossing-model.

Efter implementeringen af de tre algoritmer, benyttedes disse til at estimere et samlet sæt af fraktil-kurver vha. et træningssæt bestående af vindkraftdata for en udvalgt tidshorisont. Dernæst blev et testsæt taget i brug, for at sammenligne det samlede tab for hver af metoderne, og for at undersøge hvor godt datapunkterne blev delt op. Sammenligningen af tabet gav dog ikke konsistente resultater, da det især var meget forskelligt, hvilken af de to non-crossing-modeller der fik det laveste tab. Der tegnede sig tilgengæld et billede af, at crossing-modellen generelt klarer sig dårligst. Selv hvis dette ikke havde været tilfældet, har de to andre modeller, ej at forglemme, den store fordel, at de estimerer et samlet sæt af fraktil-kurver, der er meningsfuldt.

Undervejs i projektet er naturlige kubiske splines anvendt til at estimere fraktil-kurverne, men denne fremgangsmåde viste sig, at have sine begrænsninger, da ingen af metoderne gav tilfredsstillende resultater for visse tidshorisonter. Det kan derfor være interessant at undersøge, om dette problem kan løses ved at ændre placeringen af knudepunkter eller ved at benytte andre typer af splines.

Alt i alt må det konkluderes at fraktil-modeller kan estimeres via løsning af et lineært optimeringsproblem, og at det er muligt at opstille flere forskellige modeller til at undgå fraktil-krydsninger. Specielt er det interessant, at man kan implementere en stor model til beregning af samtlige ønskede fraktil-kurver på én gang.

Litteratur

[1] L. Elden, L. Wittmeyer-Koch, and H.B. Nielsen. *Introduction to numerical computation: analysis and MATLAB® illustrations*. Studentlitteratur, 2004.

[2] L. Keele. *Semiparametric regression for the social sciences*. Wiley Online Library, 2008.

[3] R. Koenker. *Quantile regression*, volume 38. Cambridge Univ Pr, 2005.

[4] GR Lindfield and JET Penny. Using matlab for sparse matrices. *International Journal of Mathematical Education in Science and Technology*, 28(3):427–436, 1997.

[5] Jan Kloppenborg Møller, Henrik Aalborg, orlov 31.07.2008 Nielsen, and Henrik Madsen. *Algorithms for Adaptive Quantile Regression - and a Matlab Implementation*. 2006.

[6] J. K. Møller. Modeling of uncertainty in wind energy forecast. Master's thesis, Informatics and Mathematical Modelling, Technical University of Denmark, DTU, Richard Petersens Plads, Building 321, DK-2800 Kgs. Lyngby, 2006. Supervised by Henrik Madsen and Henrik Aalborg Nielsen, IMM.

[7] H.B. Nielsen. Algorithms for linear optimization, an introduction. *Course note for the DTU course. Optimization and Data Fitting*, 2, 1999.

[8] P. Pinson, H.A. Nielsen, J.K. Møller, H. Madsen, and G.N. Kariniotakis. Nonparametric probabilistic forecasts of wind power: required properties and evaluation. *Wind Energy*, 10(6):497–516, 2007.

[9] Y. Wu and Y. Liu. Stepwise multiple quantile regression estimation using non-crossing constraints. *Statistics and Its Interface*, 2:299–310, 2009.

A Kildekoder

A.1 R-script 1

Kildekode 1: R-kode til simulering af data til afsnit 9.

```
library("quantreg");
library("splines");
library("MASS")


#set.seed(2000)

frak <- seq(0.02,0.98,0.02); #Fraktiler.

antpunkter <- 100; #Antal punkter i træningssættet.

# genererer træningsdata.
xpunkter <- runif(antpunkter,0,1); # genererer x-punkter via uniform
    fordeling
mupunkter <- 0 + 1*xpunkter; # finder middelværdien af Y når X=x.
ypunkter <- rnorm(antpunkter, mupunkter, 1/2) # genererer y-punkter via
    normalfordeling

# genererer testdata.
xpunktertest <- runif(100000,0,1);
mupunktertest <- 0 + 1*xpunktertest;
ypunktertest <- rnorm(100000, mupunktertest, 1/2)


# Beregner de sande fraktilers y-værdier i træningspunkterne.
Frak <- matrix(0,length(xpunkter),length(frak))
for (i in 1:length(frak)) {
  for (j in 1:length(mupunkter)) {
    Frak[j,i] <- qnorm(frak[i], mean = mupunkter[j], sd = 1/2);
  }}

# Beregner de sande fraktilers y-værdier i testpunkterne.
FrakTest <- matrix(0,length(xpunktertest),length(frak))
for (i in 1:length(frak)) {
  for (j in 1:length(mupunktertest)) {
    FrakTest[j,i] <- qnorm(frak[i], mean = mupunktertest[j], sd = 1/2);
  }}


# Plot af sande fraktiler.
plot(xpunkter,ypunkter)
lines(xpunkter,Frak[1:length(xpunkter),1], col = "blue");
lines(xpunkter,Frak[1:length(xpunkter),2], col = "red");
#osv..

Linpunkter <- cbind(xpunkter,ypunkter);
Data = data.frame(cbind(xpunkter,ypunkter));

LinpunkterTEST <- cbind(xpunktertest,ypunktertest);
```

```
DataTEST = data.frame(cbind(xpunktertest,ypunktertest));

# Beregner 50%-fraktilen ved brug af datapunkterne.
frak50 <- rq(ypunkter~xpunkter, tau=0.50, data = Data)
frak50 = frak50$coefficients;

frak50test <- rq(ypunktertest~xpunktertest, tau=0.50, data = DataTEST)
frak50TEST = frak50test$coefficients;

# Plot af 50%-fraktilen ifht. data.
Q50 <- frak50[1]+frak50[2]*xpunkter;
lines(xpunkter,Q50, col = "green");

# Gemmer data, de sande fraktiler og den estimerede 50% fraktil,
# for træningssæt:
write.matrix(Linpunkter, file = "Linpunkter");
write.matrix(Frak,file = "TeoFrak");
write(frak50, file = "frak50R",ncolumns = 1);

# for testsæt:
write.matrix(LinpunkterTEST, file = "LinpunkterTEST");
write.matrix(FrakTest,file = "TeoFrakTEST");
write(frak50TEST, file = "frak50RTEST",ncolumns = 1);
```

A.2 R-script 2

Kildekode 2: R-kode til generering af begyndelses-fraktil samt designmatrix.

```
# Indlæser pakker
library("quantreg");
library("splines");
library("MASS");    # matrix invers


# Indlæser data
imm.pow<-read.table(file="C:/Users/klimData.csv",sep=";",header=TRUE)

AllData <- imm.pow[imm.pow[ ,"hors"] == 1, ]; #Henter data for bestemt
    horisont.

pred <- AllData[ ,"pred"];
meas <- AllData[ ,"meas"];

# inddeler data i træningssæt og testsæt
trainpred <- pred[1:ceiling(length(pred)/2)];
trainmeas <- meas[1:ceiling(length(pred)/2)];
testpred <- pred[(ceiling(length(pred)/2)+1):length(pred)];
testmeas <- meas[(ceiling(length(pred)/2)+1):length(pred)];

Data <- data.frame(cbind(trainmeas,trainpred));

# Fraktiler vil vi beregne.
tau<-seq(0.02,0.98,by=0.02);

# Laver knudevektor.
procenter <- c(0.1, 0.3, 0.5, 0.7, 0.9, 0.99);
knots<-quantile(trainpred,probs=procenter);  #indre knudepunkter

# Laver design-matrix for train-punkterne.
X <- ns(trainpred,knots = knots, Boundary.knots = c(0,1));

# Laver design-matrix for de punkter, hvor der testes for crossings.
xx = seq(0,1,0.01) # punkter hvor der testes for crossings
Xnc <- ns(xx,knots = knots, Boundary.knots = c(0,1));

# Laver design-matrix for test-punkterne.
Xtest <- ns(testpred,knots = knots, Boundary.knots = c(0,1));

# Beregner spline-coefficienter for traindata for alle fraktilerne.
i <- 1;
SC <- matrix(0,dim(X)[2]+1, length(tau));
for (i in 1:length(tau)) {
  C <- rq(trainmeas~X, tau=tau[i],data=Data);
  C <- as.numeric(C$coefficients);
  SC[1:(dim(X)[2]+1),i] <- C;
}

# Tilføjer 1-tals-søjle til design-matricerne, så skæring også er med.
X <- cbind(rep(1,dim(X)[1]),X);
```

```
Xnc <- cbind(rep(1,dim(Xnc)[1]),Xnc);
Xtest <- cbind(rep(1,dim(Xtest)[1]),Xtest);

# Beregner fraktil-kurverne.
Q <- Xnc %*% SC;

#plotter punkter..
plot(trainpred,trainmeas,xlim = c(0 ,1), ylim = c(0 ,1));

# koefficienterne for 50%-fraktilen.
beta50 <- SC[ ,25];

# gemmer design-matricerne, data og koefficinterne til 50%-fraktilen.
write(beta50, file = "beta50_h1",ncolumns = 1);
write.matrix(X, file = "X_h1");
write.matrix(Xnc, file = "Xnc_h1");
write.matrix(Xtest, file = "Xt_h1")
write(trainmeas, file = "y_h1", ncolumns = 1);
write(trainpred, file = "xp_h1", ncolumns = 1);
write(testmeas, file = "yt_h1", ncolumns = 1);
write(testpred, file = "xpt_h1", ncolumns = 1);
```

A.3 Algoritme 1 - CrossQuant

Kildekode 3: MATLAB kode for CrossQuant.

```
function [Beta, T] = CrossQuant(X, beta0, y, tau)

% Bestmmer en række konstanter:
K = length(beta0);
ltau = length(tau);
N = size(X,1);

% Pre-allokerer matricer for at lagre 'beta'-værdier samt tabet for alle
% 'tau(i)'-fraktiler.
Beta = zeros(K,ltau);
T = zeros(ltau,1);

for j = 1:ltau

    % Beregner residualerne.
    r = y - X*beta0;

    % Indeksmængden, 'h', bestemmes via funktionen 'find_h'.
    [h, non_h, P] = find_h(X, r, beta0, N);

    % Samler xB, som indeholder beta og slackvariablene.
    xB = [beta0; abs(r(non_h))];

    % Beregner koefficienterne i objektfunktionen.
    rho = P;
    rho(rho==-1) = 1-tau(j);
    rho(rho==1) = tau(j);

    % Beregner B_inv
    P = diag(P);
    B11 = eye(size(X(h,:)))/X(h,:);
    B12 = zeros(K,N-K);
    B21 = -P*X(non_h,:)/X(h,:);
    B22 = P;
    Binv = [B11 B12; B21 B22];
    P = diag(P);

    % Beregner 'C'.
    C = [eye(K) -eye(K); zeros(N-K,2*K)];

    % Beregner 'd', som angiver det relative fald for alle retninger.
    cC = [tau(j)*ones(K,1); (1-tau(j))*ones(K,1)];
    cB = [zeros(K,1); rho];
    d = cC - C'*Binv'*cB;

    iterates = 0;
    index1 = 1:2*K;
    index2 = 1:N;

    % Så længe der eksisterer en aftagende retning, dvs. så længe der er
    % negative elementer i 'd', optimeres tabsfunktionen.
```

```
    while (sum(d<0)>0)

        % Finder 's', dvs. indeksnummeret for det mest negative element i
        % 'd', som angiver det relative fald i objektfunktionen i den på-
        % gældende retning.
        s = index1(d == min(d));
        s = s(1);

        % Beregner den aftagende retning for den valgte 's'.
        hdir = Binv*C(:,s);
        if (hdir <= 0)
            break;
        end

        % Beregner de mulige skridtlængder i retningen 'hdir'.
        sigma = zeros(length(hdir)-K,1);
        for i = K+1:length(hdir)
            if (hdir(i) > 10^(-6))
                sigma(i-K) = xB(i)/hdir(i);
            else
                sigma(i-K) = inf;
            end
        end

        % Skridtlængden, 'alpha', bestemmes.
        alpha = min(sigma);

        % 'xB(q)' er elementet i 'xB', som bliver nul ved skridtlængden
        % 'alpha' i retningen 'hdir'.
        q = index2(sigma==min(sigma));
        q = q(1);

        % Skifter en slackvariabel ind og ud af basis.
        non_h_ud = non_h(q);
        if s>K
            h_ud = h(s-K);
            non_h(q) = h_ud;
            h(s-K) = non_h_ud;
        else
            h_ud = h(s);
            non_h(q) = h_ud;
            h(s) = non_h_ud;
        end

        % Ændring af residualets fortegn.
        if s>K
            P(q) = -1;
        else
            P(q) = 1;
        end

        % Opdaterer 'xB'.
        xB = xB - alpha*hdir; xB(q+K) = alpha;

        P = diag(P);
```

```
        B11 = eye(size(X(h,:)))/X(h,:);
        B12 = zeros(K,N-K);
        B21 = -P*X(non_h,:)/X(h,:);
        B22 = P;
        Binv = [B11 B12; B21 B22];

        % Beregner koefficienterne i objektfunktionen.
        P = diag(P);
        rho = P;
        rho(rho==-1) = 1-tau(j);
        rho(rho==1) = tau(j);

        % Beregner 'd'.
        cC = [tau(j)*ones(K,1); (1-tau(j))*ones(K,1)];
        cB = [zeros(K,1); rho];
        g = Binv'*cB;
        d = cC - C'*g;

        % Beregner tabet.
        iterates = iterates + 1;
        Tab = rho'*xB(K+1:end)
    end
    Beta(:,j) = xB(1:K);
    T(j) = rho'*xB(K+1:end);
end
```

A.4 Algoritme 2 - NonCrossSingle

Kildekode 4: MATLAB kode for NonCrossSingle.

```
function [Beta,T, H] = NonCrossSingle(X,beta0,y,Xnc,tau,lb, ub, h0)
%% Initialisering

% Bestemmer en række konstanter.
tol = 1e-10;
K = length(beta0);
N = size(X,1);
Nnc = size(Xnc,1);

% Beregner residualerne.
r = y - X*beta0;

% Samler den samlede designmatrix for forudsigelsen og kontrolpunkterne.
XS = [X; Xnc; Xnc; Xnc];

% Samler residualerne i vektoren 'rS':
rS = [r ; zeros(Nnc,1); ub-Xnc*beta0; lb-Xnc*beta0];
%    1) 'r' er residualerne mellem den 'tau(i)'te-fraktil og 'y'.
%    2) 'zeros(Nnc,1)' er residualerne mellem den 'tau(i)'te-fraktil og
%        den 'tau(i+1)'te-fraktil. Residualerne er '0', da vi antager, at
%        disse fraktiler indledningsvist er sammenfaldende.
%    3) 'ub-Xnc*beta' er residualerne mellem den 'tau(i)'te-fraktil og den
%       øvre grænse, y = ub.
%    4) 'lb-Xnc*beta' er residualerne mellem den 'tau(i)'te-fraktil og den
%       nedre grænse, y = lb.

% Bestemmer indeksmængderne 'h' og 'non_h'.
if nargin < 8
    % Hvis 'h' ikke gives som input, beregnes 'h', 'P' og 'non_h' via
    % funktionen 'find_h'.
    [h, non_h, P] = find_h(X, r, beta0, N);

    % Samler 'non_h'erne for både 'r' de øvrige residualerne i 'rS' i
    % vektoren 'non_h_S'.
    non_h_S = [non_h N+1:N+3*Nnc];
else
    % Hvis 'h' gives som input, beregnes 'non_h' og 'P' via 'h'.
    index1 = 1:N+3*Nnc;
    h = h0;
    % Finder hvilke indekser i mængden {1,..,N+3*Nnc}, som indgår i 'h'.
    % Dem som ikke indgår i 'h' udgør 'non_h'.
    Ih = ismember(index1, h);
    non_h_S = index1(~Ih);
    non_h = non_h_S(non_h_S<=N);
    P = sign(r(non_h));
end

% Samler 'xB' bestående af koefficienterne beta0 samt slackvariablene.
xB = [beta0 ; abs(rS(non_h_S))];
tau1 = tau(1);   % 'tau'-værdien for den 'tau(i)'te-fraktil.
tau2 = tau(2);   % 'tau'-værdien for den 'tau(i+1)'te-fraktil.
```

```
% Tæller antallet af 'h', 'non_h', 'h_nc', 'non_h_nc', 'non_h_ncy1' og
% 'non_h_ncy0', for holde styr på hvor elementerne i hhv. 'h' og 'non_h'
% kommer fra; om det er fra 'r', non-crossing betingelser eller fra grænse
% betingelser.
ant_h = sum(h<=N);
ant_nh = N-ant_h;
ant_hnc = sum(h<=N+Nnc)-ant_h;
ant_nhnc = Nnc-ant_hnc;
ant_hncy1 = sum(h<=N+2*Nnc)-(ant_h+ant_hnc);
ant_nhncy1 = Nnc-ant_hncy1;
ant_hncy0 = length(h)-(ant_h+ant_hnc+ant_hncy1);
ant_nhncy0 = Nnc-ant_hncy0;

% Konstruerer en matrix 'PS', som repræsenterer fortegnene for residual-
% erne i 'rS'. 'PS' konstrueres først som en vektor med 1-taller og
% modificeres herefter.
PS = ones(ant_nh+ant_nhnc+ant_nhncy1+ant_nhncy0,1);
PS(1:ant_nh) = P;                                % Fortegn for 'r'.
PS(ant_nh+1:ant_nh+ant_nhnc) = sign(tau1-tau2);% Fortegn for 'Crossing'.
PS(ant_nh+ant_nhnc+ant_nhncy1+1:end) = -1;      % Fortegn for nedre grænse.
PS = diag(PS);

% Beregner B_inv
B11 = eye(size(XS(h,:)))/XS(h,:);
B21 = -PS*XS(non_h_S,:)/XS(h,:);
B12 = zeros(K,N+3*Nnc-K);
B22 = PS;
B_inv = [B11 B12 ; B21 B22];

% Beregner koefficienterne i objektfunktionen.
rho = diag(PS(1:ant_nh,1:ant_nh));
rho(rho==-1) = 1-tau1;
rho(rho==1) = tau1;

% Lagring af 'beta'-værdierne for hver af de beregnede fraktiler, disses
% tab, 'T', samt indeksmængden 'h'.
T = zeros(length(tau),1);
T(1) = [rho; zeros(ant_nhnc+ant_nhncy1+ant_nhncy0,1)]'*xB(K+1:end);
Beta = zeros(K,length(tau));
Beta(:,1) = beta0;
H = zeros(K,length(tau));
H(:,1) = h;

index2 = 1:N+3*Nnc-K;

% Beregning af den tau(i)'te-fraktil.
for j = 1:length(tau)-1;
    iterates = 0;
    tau1 = tau(j);
    tau2 = tau(j+1);

    % Beregner koefficienterne i objektfunktionen.
    rho = diag(PS(1:ant_nh,1:ant_nh));
    rho(rho==-1) = 1-tau2;
```

```
rho(rho==1) = tau2;

% Beregner herunder den sparse matrice 'C' angivet på triplet-form.

% Alle ikke-nul-elementer lagres i vektoren 'Cval'.
Cval = [repmat([1 -1], [1 ant_h])'; sign(tau1-tau2)*...
    ones(ant_hnc,1); ones(ant_hncy1,1); -ones(ant_hncy0,1) ];

% Søjle-indekser.
JC = (1:2*ant_h+ant_hnc+ant_hncy1+ant_hncy0)';
JC(1:2:2*ant_h) = 1:ant_h;
JC(2:2:2*ant_h) = ant_h+1:2*ant_h;

% Række-indekser.
IC = repmat(1:ant_h, [2 1]);
IC = [IC(:); (ant_h+1:ant_h+ant_hnc+ant_hncy1+ant_hncy0)'];

% Størrelsen af 'C'.
mC = N+3*Nnc;
nC = 2*ant_h+ant_hnc+ant_hncy1+ant_hncy0;

% Konstruerer 'C'.
C = sparse(IC,JC, Cval, mC, nC);

% Beregner 'd', som angiver det relative gain for alle retninger.
cC = [tau2*ones(ant_h,1) ; (1-tau2)*ones(ant_h,1) ; zeros(ant_hnc+...
    ant_hncy1+ant_hncy0,1)];
cB = [zeros(K,1); rho; zeros(ant_nhnc + ant_nhncy1 + ant_nhncy0,1)];
g = B_inv'*cB;
d = cC - C'*g;

% Så længe der eksisterer en aftagende retning, dvs. så længe der er
% negative elementer i 'd', optimeres tabsfunktionen.
while (sum(d<0) >0)
    % Tæller antallet af aftagende retninger i 'd' og lagrer dem i en
    % vektoren 'ds'. Deres tilhørende indeksværdier lagres i 's'.
    ant_neg_d = sum(d<0);
    [ds s]= sort(d);
    ds = ds(1:ant_neg_d);
    s = s(1:ant_neg_d);
    a_q_s = zeros(ant_neg_d,3);
    index3 = 1:ant_neg_d;

    % For alle aftagende retninger, beregnes selve retningen, 'hdir'.
    for l = 1:ant_neg_d
        hdir = B_inv*C(:,s(l));

        if sum(hdir > 0) > 0
            % Beregner 'alpha', 'q' og 's' for alle aftagende ret-
            % ninger. Disse værdier lagres i den pre-allokerede matrix
            % 'a_q_s'.
            sigma = zeros(length(hdir)-K,1);
            xB_down = xB(K+1:end);          % Slackvariablene i 'xB'.
            hdir_down = hdir(K+1:end);      % De tilsvarende i 'hdir'.
            sigma(hdir_down>tol) = xB_down(hdir_down>tol)./...
```

```
                hdir_down(hdir_down>tol);
            sigma(hdir_down <= tol) = inf;

            % Her gemmes 'alpha', 'q' og 's' i matricen 'a_q_s'.
            a_q_s(l,1) = min(sigma);                    % 'alpha'.
            q = index2(sigma==min(sigma)); q = q(1);
            a_q_s(l,2) = q;                             % 'q'
            a_q_s(l,3) = s(l);                          % 's'
        else
            disp('Problemet er ubegrænset')
        end
    end

    if  sum(a_q_s(:,1) > 0) == 0
        % Finder 's' og 'q', hvis alle alpha = 0.
        alpha = a_q_s(1,1);
        q = a_q_s(1,2);
        s = a_q_s(1,3);
    else
        % Finder 's' og 'q', hvis ikke alle alpha = 0. Dette gøres ved
        % at vælge det element i 'd', som giver det største egentlige
        % fald i objektfunktionen. Det egentlige fald for alle 'alpha'
        % beregnes i 'gain'.
        gain = ds.*a_q_s(:,1);
        Igain = index3(gain == min(gain)); Igain = Igain(1);
        alpha = a_q_s(Igain,1);
        q = a_q_s(Igain,2);
        s = a_q_s(Igain,3);
    end

    % Retningen, 'hdir', bestemmes for den valgte 's' og 'q'.
    hdir = B_inv*C(:,s);

    % Skifter en slackvariabel ind og ud af basis. Ændringen foretages
    % ikke direkte i 'h' og 'non_h', men i to midlertidige variable
    % 'h_temp' og 'non_h_temp'.
    non_h_temp = non_h_S;
    h_temp = h;
    non_h_ud = non_h_S(q);
    if s>ant_h
        h_ud = h(s-ant_h);
        non_h_temp(q) = h_ud;
        h_temp(s-ant_h) = non_h_ud;
    else
        h_ud = h(s);
        non_h_temp(q) = h_ud;
        h_temp(s) = non_h_ud;
    end

    h_temp = sort(h_temp);

    % Undersøger nu om 'XS(h_temp,:)' har ranken 'K'. Sålænge dette
    % ikke gælder findes en ny retning ved at gentage processen oven-
    % for.
    while rank(XS(h_temp,:))<K;
```

```
% Sætter d(s) = 1, så den samme retning, 'hdir', ikke vælges.
d(s) = 1;

% I det følgende gentages linje 126 - 196, for at finde et sæt
% af 'h_temp', som opfylder at 'XS(h_temp,:)'
if sum(d<0)>0
    antal_neg_d = sum(d<0);
    [dn s]= sort(d);
    dn = dn(1:antal_neg_d);
    s = s(1:antal_neg_d);
    a_q_s = zeros(antal_neg_d,3);
    index3 = 1:antal_neg_d;

    for l = 1:antal_neg_d
        hdir = B_inv*C(:,s(l));
        if sum(hdir > 0) > 0
            sigma = zeros(length(hdir)-K,1);
            xB_down = xB(K+1:end);
            hdir_down = hdir(K+1:end);
            sigma(hdir_down>tol) = xB_down(hdir_down>tol)./...
                hdir_down(hdir_down>tol);
            sigma(hdir_down <= tol) = inf;

            a_q_s(l,1) = min(sigma);
            q = index2(sigma==min(sigma)); q = q(1);
            a_q_s(l,2) = q;
            a_q_s(l,3) = s(l);
        else
            disp('Problemet er ubegrænset');
            break;
        end
    end

    if  sum(a_q_s(:,1) > 0) == 0
        q = a_q_s(1,2);
        s = a_q_s(1,3);
        alpha = a_q_s(1,1);
    else
        gain = dn.*a_q_s(:,1);
        Igain = index3(gain == min(gain)); Igain = Igain(1);
        alpha = a_q_s(Igain,1);
        q = a_q_s(Igain,2);
        s = a_q_s(Igain,3);
    end

    hdir = B_inv*C(:,s);

    non_h_temp = non_h_S;
    h_temp = h;
    non_h_ud = non_h_S(q);
    if s>ant_h
        h_ud = h(s-ant_h);
        non_h_temp(q) = h_ud;
        h_temp(s-ant_h) = non_h_ud;
    else
```

```
                h_ud = h(s);
                non_h_temp(q) = h_ud;
                h_temp(s) = non_h_ud;
            end
        else
            break;
        end
        h_temp = sort(h_temp);
    end

    % Afbryder funktionen, hvis ingen elementer i 'd' er negative.
    if sum(d<=0)==0
        break;
    end

    % 'h_temp' og 'non_h_temp' beholdes, idet der nu gælder at
    % 'XS(h_temp,:)' = K.
    h = h_temp;
    non_h_S = non_h_temp;

    % Ændrer fortegn i PS, så det stemmer med fortegnet for
    % residualet, som ryger ind i basis.
    if s<=ant_h
        PS(q,q) = 1;
    elseif s > ant_h && s <= 2*ant_h
        PS(q,q) = -1;
    elseif s > 2*ant_h && s <= 2*ant_h + ant_hnc
        PS(q,q) = sign(tau1-tau2);
    elseif s > 2*ant_h+ant_hnc && s <= 2*ant_h+ant_hnc+ant_hncy1
        PS(q,q) = 1;
    elseif s > 2*ant_h+ant_hnc+ant_hncy1
        PS(q,q) = -1;
    end

    % Opdaterer antallet af slackvariable inde og udenfor basis.
    if s <= 2*ant_h && q > ant_nh
        if q <= ant_nh + ant_nhnc
            ant_hnc = ant_hnc + 1;
            ant_nhnc = ant_nhnc - 1;
        elseif q <= ant_nh+ant_nhnc+ant_nhncy1
            ant_hncy1 = ant_hncy1 + 1;
            ant_nhncy1 = ant_nhncy1 - 1;
        else
            ant_hncy0 = ant_hncy0 + 1;
            ant_nhncy0 = ant_nhncy0 - 1;
        end
        ant_h = ant_h - 1;
        ant_nh = ant_nh + 1;

    elseif (s > 2*ant_h && s <= 2*ant_h+ant_hnc) && (q <= ant_nh ||...
            q > ant_nh + ant_nhnc)
        if q <= ant_nh
            ant_h = ant_h + 1;
            ant_nh = ant_nh - 1;
        elseif q <= ant_nh+ant_nhnc+ant_nhncy1
```

```
            ant_hncy1 = ant_hncy1 + 1;
            ant_nhncy1 = ant_nhncy1 - 1;
        else
            ant_hncy0 = ant_hncy0 + 1;
            ant_nhncy0 = ant_nhncy0 - 1;
        end
        ant_hnc = ant_hnc - 1;
        ant_nhnc = ant_nhnc + 1;

    elseif (s>2*ant_h + ant_hnc && s <= 2*ant_h+ant_hnc+ant_hncy1)...
            && (q<=ant_nh + ant_nhnc || q>ant_nh+ant_nhnc+ant_nhncy1)
        if q <= ant_nh
            ant_h = ant_h + 1;
            ant_nh = ant_nh - 1;
        elseif q <= ant_nh+ant_nhnc
            ant_hnc = ant_hnc + 1;
            ant_nhnc = ant_nhnc - 1;
        else
            ant_hncy0 = ant_hncy0 + 1;
            ant_nhncy0 = ant_nhncy0 - 1;
        end
        ant_hncy1 = ant_hncy1 - 1;
        ant_nhncy1 = ant_nhncy1 + 1;

    elseif s > 2*ant_h+ant_hnc+ant_hncy1 && q <= ant_nh+ant_nhnc+...
            ant_nhncy1
        if q <= ant_nh
            ant_h = ant_h+1;
            ant_nh = ant_nh-1;
        elseif q <= ant_nh + ant_nhnc
            ant_hnc = ant_hnc + 1;
            ant_nhnc = ant_nhnc - 1;
        else
            ant_hncy1 = ant_hncy1 + 1;
            ant_nhncy1 = ant_nhncy1 - 1;
        end
        ant_hncy0 = ant_hncy0 - 1;
        ant_nhncy0 = ant_nhncy0 + 1;
    end

    % Opdaterer xB.
    xB = xB - alpha*hdir; xB(q+K) = alpha;

    % Sorterer.
    [non_h_S,ixS] = sort(non_h_S);
    h = sort(h);
    PS = diag(PS);
    PS = PS(ixS);
    PS = diag(PS);
    xB_temp = xB(K+1:end);
    xB(K+1:end) = xB_temp(ixS);

    % Opdaterer B_inv
    B11 = eye(size(XS(h,:)))/XS(h,:);
    B21 = -PS*XS(non_h_S,:)/XS(h,:);
```

```
        B12 = zeros(K,N+3*Nnc-K);
        B22 = PS;
        B_inv = [B11 B12 ; B21 B22];

        % Opdaterer koefficienter i objektfunktionen.
        rho = diag(PS(1:ant_nh,1:ant_nh));
        rho(rho==-1) = 1-tau2;
        rho(rho==1) = tau2;

        % Beregner her den sparse matrice C angivet på triplet-form.
        Cval = [repmat([1 -1], [1 ant_h])'; sign(tau1-tau2)*...
            ones(ant_hnc,1); ones(ant_hncy1,1); -ones(ant_hncy0,1) ];
        JC = (1:2*ant_h+ant_hnc+ant_hncy1+ant_hncy0)';
        JC(1:2:2*ant_h) = 1:ant_h;
        JC(2:2:2*ant_h) = ant_h+1:2*ant_h;
        IC = repmat(1:ant_h, [2 1]);
        IC = [IC(:); (ant_h+1:ant_h+ant_hnc+ant_hncy1+ant_hncy0)'];
        mC = N+3*Nnc;
        nC = 2*ant_h+ant_hnc+ant_hncy1+ant_hncy0;
        C = sparse(IC,JC, Cval, mC, nC);

        % Beregner d.
        cC = [tau2*ones(ant_h,1) ; (1-tau2)*ones(ant_h,1) ; zeros(ant_hnc+
            ant_hncy1+ant_hncy0,1)];
        cB = [zeros(K,1); rho; zeros(ant_nhnc + ant_nhncy1 + ant_nhncy0,1)];
        g = B_inv'*cB;
        d = cC - C'*g;

        iterates = iterates + 1;
        tab = [rho; zeros(ant_nhnc+ant_nhncy1+ant_nhncy0,1)]'*xB(K+1:end)

        [s q alpha tab iterates j sum(d<0)'];
    end
    beta = xB(1:K);
    xB(K+ant_nh+1:K+ant_nh+ant_nhnc) = 0;
    T(j+1) = tab;
    Beta(:,j+1) = beta;
    H(:,j+1) = h;
end
```

A.5 Algoritme 3 - NonCrossHuge

Kildekode 5: MATLAB kode for NonCrossHuge.

```
function [Beta,T] = NonCrossHuge(X,beta0,y,Xnc,tau,lb,ub)
tol = 1e-10;
tab = 0;
Nnc = size(Xnc,1);
N = size(X,1);

% Antal beta-parametre og antal fraktiler.
[K ant_frak] = size(beta0);

% Residualerne imellem fraktilerne og 'y' opstilles i en vektor 'r'.
% Den indbyrdes afstand imellem fraktilerne opstilles i en vektor 'rnc'.
r = repmat(y,1,ant_frak)-X*beta0;
r = r(:);        % Fra matrix til vektor.
r_nc = [lb*ones(Nnc,1) Xnc*beta0]-[Xnc*beta0 ub*ones(Nnc,1)];
r_nc = r_nc(:);  % Fra matrix til vektor.

% 'r' og 'rnc' opstilles i en vektor 'rs'.
rs = [r; r_nc];

% Bestemmer 'h' for alle fraktilerne.
[h, P, non_h] = find_h_huge(X, Xnc, rs, ant_frak);

% Samler xB bestående af beta0 og slackvariablene.
xB = [beta0(:) ; abs(rs(non_h))];

% Her tælles antallet af 'h', 'non_h', 'h_nc', og 'non_h_nc', for at holde
% styr på hvor elementerne i hhv. 'h' og 'non_h' kommer fra; om det er fra
% 'r' eller 'rnc'.
ant_h = sum(h<=N*ant_frak);
ant_nh = N*ant_frak-ant_h;
ant_hnc = length(h)-ant_h;
ant_nhnc = length(non_h)-ant_nh;

% Via diagonalmatricen, P, konstrueres både 'Pvektor' og 'Pmatrix'.
P(end-ant_nhnc+1:end) = -1;
lp = length(P); lR = length(rs);
Pvector = P; Pmatrix = sparse(1:lp, 1:lp, P);

% For hvert sæt af 'h_i' og 'non_h_i', tilhørende fraktil 'Q_i', udtrækkes
% de tilsvarende rækker i 'XS'.
XS_h = find_PartialXS(X,Xnc,h,ant_frak);
XS_nh = find_PartialXS(X,Xnc,non_h,ant_frak);
NSh = length(XS_h);

% Beregner B_inv
B11 = sparse(eye(size(XS_h)))/XS_h;
B12 = sparse(NSh,lp);
B21 = -Pmatrix'*XS_nh*B11;
B22 = Pmatrix;
B_inv = [B11 B12 ; B21 B22];
```

```
% Beregner antallet af residualer fra 'r' for hvert af fraktilerne for
% herefter at kunne beregne 'rho'.
non_h_X = non_h(non_h<=ant_frak*N);
X_frak_nr = ceil(non_h_X/N);
ant_res_frak = histc(X_frak_nr,1:ant_frak);

rho = Pvector(1:ant_nh);
start = 0;
for i = 1:ant_frak
    rho_temp = rho(start+1:start+ant_res_frak(i));
    rho_temp(rho_temp == -1) = 1-tau(i);
    rho_temp(rho_temp == 1) = tau(i);
    rho(start+1:start+ant_res_frak(i)) = rho_temp;
    start = start + ant_res_frak(i);
end

% Beregner rho_c_p og rho_c_n, dvs. koefficienterne for objektfunktionen
% for både de positive og negative residualer i xC.
rho_c_p = zeros(ant_h,1);
ant_c_frak = length(y)-ant_res_frak;
start = 0;

for i = 1:ant_frak;
    rho_c_temp = ones(ant_c_frak(i),1)*tau(i);
    rho_c_p(start+1:start+ant_c_frak(i)) = rho_c_temp;
    start = start + ant_c_frak(i);
end
rho_c_n = 1-rho_c_p;

% Beregner her den sparse matrix C angivet på triplet-form.

% Alle ikke-nul-elementer lagres i vektoren 'Cval'.
Cval = [repmat([1 -1], [1 ant_h])'; -ones(ant_hnc,1)];

% Søjle-indekser.
JC = (1:2*ant_h+ant_hnc)';
JC(1:2:2*ant_h) = 1:ant_h;
JC(2:2:2*ant_h) = ant_h+1:2*ant_h;

% Række-indesker.
IC = repmat(1:ant_h, [2 1]);
IC = [IC(:); (ant_h+1:ant_h+ant_hnc)'];

% Størrelsen af 'C'.
mC = lR;
nC = 2*ant_h+ant_hnc;

% Konstruerer 'C'.
C = sparse(IC(:), JC, Cval, mC, nC);

% Beregner 'd', som angiver det relative gain for alle retninger.
cC = [rho_c_p; rho_c_n; zeros(ant_hnc,1)];
cB = [zeros(ant_frak*K,1); rho; zeros(ant_nhnc,1)];
g = B_inv'*cB;
d = cC - C'*g;
```

```
iterates = 0;
while sum(d<0)>0;
    % Tæller antallet af aftagende retninger i 'd' og lagrer dem i en
    % vektoren 'ds'. Deres tilhørende indeksværdier lagres i 's'.
    ant_neg_d = sum(d<0);
    [ds s]= sort(d);
    ds = ds(1:ant_neg_d);
    s = s(1:ant_neg_d);
    a_q_s = zeros(ant_neg_d,3);
    index3 = 1:ant_neg_d;

    % For alle aftagende retninger, beregnes en matrix med retninger.
    hdir = B_inv*C(:,s);

    % Hvis problemer er begrænset beregnes 'alpha', 'q' og 's' for alle
    % aftagende retninger. Disse værdier lagres i 'a_q_s'.
    if  sum(hdir > 0) > 0
        % 'alpha' beregnes udfra 'sigma', som bestemmes her.
        sigma = zeros(size(hdir,1)-ant_frak*K,ant_neg_d);
        xB_nedre = repmat(xB(ant_frak*K+1:end),1,ant_neg_d);
        hdir_nedre = hdir(ant_frak*K+1:end,:);
        Ihdir = hdir_nedre>tol;
        sigma(Ihdir) = xB_nedre(Ihdir)./...
            hdir_nedre(Ihdir);
        sigma(~Ihdir) = inf;

        % Her gemmes 'alpha', 'q' og 's' i matricen 'a_q_s'.
        [alpha q] = min(sigma);
        a_q_s(:,1) = alpha;
        a_q_s(:,2) = q;
        a_q_s(:,3) = s;
    end

    if  sum(a_q_s(:,1) > 0) == 0
        % Finder 's' og 'q', hvis alle alpha = 0.
        alpha = a_q_s(1,1);
        q = a_q_s(1,2);
        s = a_q_s(1,3);
    else
        % Finder 's' og 'q', hvis ikke alle alpha = 0. Dette gøres ved at
        % vælge det element i 'd', som giver det største egentlige fald i
        % objektfunktionen. Det egentlige fald for alle alpha'er beregnes i
        % 'gain'.
        gain = ds.*a_q_s(:,1);
        [gain Igain] = min(gain);
        alpha = a_q_s(Igain,1);
        q = a_q_s(Igain,2);
        s = a_q_s(Igain,3);
    end

    % Retningen bestemmes for den valgte 's' og 'q'.
    Ihh = index3(a_q_s(:,3) == s); Ihh = Ihh(1);
    hdir = hdir(:,Ihh);
```

```
% Skifter en slackvariabel ind og ud af basis.
non_h_ud = non_h(q);
if s>ant_h
    h_ud = h(s-ant_h);
    non_h(q) = h_ud;
    h(s-ant_h) = non_h_ud;
else
    h_ud = h(s);
    non_h(q) = h_ud;
    h(s) = non_h_ud;
end

% Ændrer fortegn i P, så det stemmer med fortegnet for residualet, som
% ryger ind i basis.
if  s<=ant_h
    Pvector(q) = 1;
else
    Pvector(q) = -1;
end

% Opdaterer antallet af slackvariable inde og udenfor basis.
if s <= 2*ant_h && q > ant_nh
    ant_hnc = ant_hnc + 1;
    ant_nhnc = ant_nhnc - 1;
    ant_h = ant_h - 1;
    ant_nh = ant_nh + 1;
elseif s > 2*ant_h && q <= ant_nh
    ant_h = ant_h + 1;
    ant_nh = ant_nh - 1;
    ant_hnc = ant_hnc - 1;
    ant_nhnc = ant_nhnc + 1;
end

% Opdaterer xB.
xB = xB - alpha*hdir; xB(q+ant_frak*K) = alpha;

% Sorterer.
[non_h,ix] = sort(non_h);
h = sort(h);
Pvector = Pvector(ix);
Pmatrix = sparse(1:lp,1:lp,Pvector);
xB_temp = xB(ant_frak*K+1:end);
xB(ant_frak*K+1:end) = xB_temp(ix);

% Beregner B_inv
XS_h = find_PartialXS(X,Xnc,h,ant_frak);
XS_nh = find_PartialXS(X,Xnc,non_h,ant_frak);
NSh = length(XS_h);

B11 = sparse(eye(size(XS_h)))/XS_h;
B12 = sparse(NSh,lp);
B21 = -Pmatrix*XS_nh*B11;
B22 = Pmatrix;
B_inv = [B11 B12 ; B21 B22];
```

```
    % Beregner rho_c_p og rho_c_n, dvs. koefficienterne for objektfunktionen
    % for både de positive og negative residualer i xC.
    non_h_X = non_h(non_h<=ant_frak*N);
    X_frak_nr = ceil(non_h_X/N);
    ant_res_frak = histc(X_frak_nr,1:ant_frak);

    rho = Pvector(1:ant_nh);
    start = 0;
    for i = 1:ant_frak
        rho_temp = rho(start+1:start+ant_res_frak(i));
        rho_temp(rho_temp == -1) = 1-tau(i);
        rho_temp(rho_temp == 1) = tau(i);
        rho(start+1:start+ant_res_frak(i)) = rho_temp;
        start = start + ant_res_frak(i);
    end

    % Beregner rho_c, dvs. koefficienterne for objektfunktionen for de res-
    % idualer tilhørende xC;
    rho_c_p = zeros(ant_h,1);
    ant_c_frak = length(y)-ant_res_frak;
    start = 0;

    for i = 1:ant_frak;
        rho_c_temp = ones(ant_c_frak(i),1)*tau(i);
        rho_c_p(start+1:start+ant_c_frak(i)) = rho_c_temp;
        start = start + ant_c_frak(i);
    end
    rho_c_n = 1-rho_c_p;

    % Beregner her den sparse matrice C angivet på triplet-form.
    Cval = [repmat([1 -1], [1 ant_h])'; -ones(ant_hnc,1)];
    JC = (1:2*ant_h+ant_hnc)';
    JC(1:2:2*ant_h) = 1:ant_h;
    JC(2:2:2*ant_h) = ant_h+1:2*ant_h;
    IC = repmat(1:ant_h, [2 1]);
    IC = [IC(:); (ant_h+1:ant_h+ant_hnc)'];
    nC = 2*ant_h+ant_hnc;
    mC = lR;
    C = sparse(IC(:), JC, Cval, mC, nC);

    % Beregner d.
    cC = [rho_c_p; rho_c_n; zeros(ant_hnc,1)];
    cB = [zeros(ant_frak*K,1); rho; zeros(ant_nhnc,1)];
    g = B_inv'*cB;
    d = cC - C'*g;

    tabg = tab;
    tab = [rho; zeros(ant_nhnc,1)]'*xB(ant_frak*K+1:end);
    iterates = iterates + 1;
    [tab iterates s q alpha tab-tabg]

end
T = tab;
Beta = reshape(xB(1:ant_frak*K),K,ant_frak);
```

A.6 find_h

Kildekode 6: MATLAB kode for find_h.

```
function [h, non_h, P] = find_h(X, r, beta, n)

% Hvis absolutværdien af elementerne i 'r' er mindre end 'tol' betrages
% disse som værende 0.
tol = 1e-6;

% Beregner antallet af 0-elementer i 'r', og sorterer herefter i 'r', som
% gemmes i vektoren 'rsort'. Gemmer ligeledes indeksværdierne ifht. til 'r'
% i 'I'.
ant_nul = sum(abs(r)<tol);
[rsort I] = sort(abs(r));

% Henter indeksværdierne i 'I' hvor de tilsvarende 'r = 0'. Henter herefter
% via disse indeksværdier rækkerne i 'X', hvor 'X(h)*beta = y'.
I = I(1:ant_nul);
Xh = X(I,:);

% Benytter LU-faktoriseringen for at vælge hvilke af indeksværdierne i 'I',
% som skal udgøre 'h' og hvilke der skal kaseres.
[L,U,P] = lu(Xh);
Index = 1:ant_nul;
flag = 0;

for i = length(beta)+1:ant_nul;
    rI(i-length(beta)) = Index(P(i,1:end)==1);
    flag = 1;
end

% De overflødige nul-værdier sættes til 'tol'.
if flag == 1;
r(I(rI)) = tol;
end

% Beregner 'h', 'non_h' og 'P'.
H = 1:n;
h = H(abs(r)<tol)';
non_h = H(abs(r)>=tol);
P = sign(r(non_h));
end
```

A.7 find_h_Huge

Kildekode 7: MATLAB kode for find_h_Huge.

```
function [h, P, non_h] = find_h_huge(X, Xnc, r, ant_frak)

% Hvis absolutværdien af elementerne i 'r' er mindre end 'tol' betrages
% disse som værende 0.
tol = 1e-5;
mR = size(r);
[mX, nX] = size(X);

% Antallet af beta'er for samtlige fraktiler.
ant_var = ant_frak*nX;

% Beregner antallet af 0-elementer i 'r', og sorterer herefter i 'r', som
% gemmes i vektoren 'rsort'. Gemmer ligeledes indeksværdierne ifht. til 'r'
% i 'I'.
ant_nul = sum(abs(r)<tol);
[rsort I] = sort(abs(r));

% Henter indeksværdierne i 'r' for 'r = 0'.
I = I(1:ant_nul);
I = sort(I);

% Konstruere den fulde matrix 'XS' med 'X(h)'er i den øvre diagonalen og
% 'Xnc(hnc)'er i den nedre diagonal.
XS = find_PartialXS(X,Xnc,I,ant_frak);

% Benytter LU-faktoriseringen for at vælge hvilke af indeksværdierne i 'I',
% som skal udgøre 'h' og hvilke der skal kaseres.
[L,U,P] = lu(XS);
Index = 1:ant_nul;
flag = 0;
rI = zeros(ant_nul-ant_var,1);
for i = ant_var+1:ant_nul;
    rI(i-ant_var) = Index(P(i,:)==1);
    flag = 1;
end

% De overflødige indekser for 'r = 0' i 'r' sættes til 'tol'.
if flag == 1;
r(I(rI)) = tol;
end

% Beregner 'h', 'non_h' og 'P'.
H = (1:mR)';
h = H(abs(r)<tol);
non_h = H(abs(r)>=tol);
P = sign(r(non_h));
end
```

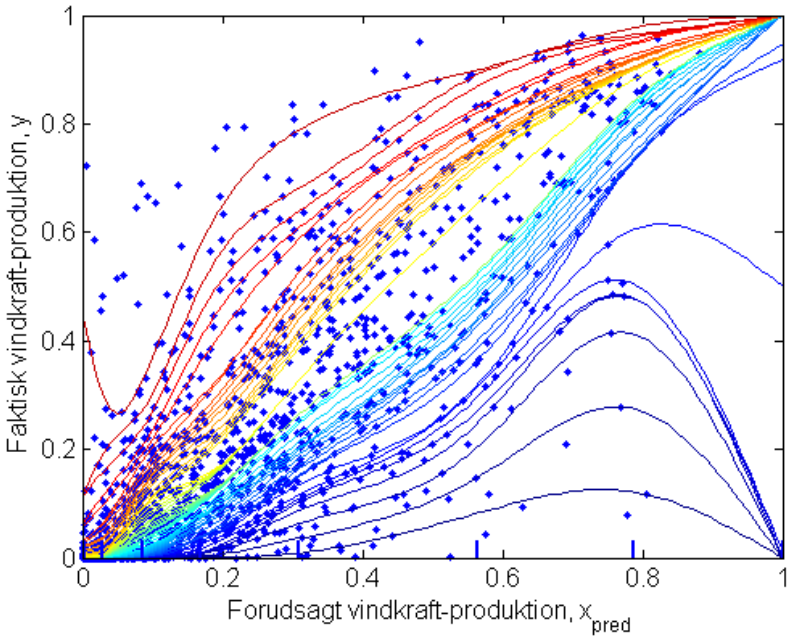
A.8 find_PartialXS

Kildekode 8: MATLAB kode for find_PartialXS.

```
function XS = find_PartialXS(X,Xnc,h,ant_frak)

mX = length(X);
[mXnc nX] = size(Xnc);
ant_rows = length(h);

% Indeler 'h' i to indeksmængder; 'h_X' for de indeksmængder, som tilhører
% 'X', og 'h_Xnc' for
h_X = h(h<=ant_frak*mX);
h_Xnc = h(h>ant_frak*mX);

% Bestemmer hvilke rækker 'h_X' angiver, hvis man ser på de enkelte X'er.
X_index = mod(h_X,mX);
X_index(X_index == 0) = mX;

% Bestemmer hvilke rækker 'h_X' angiver, hvis man ser på de enkelte Xnc'er.
Xnc_index = mod(h_Xnc-ant_frak*mX,mXnc);
Xnc_index(Xnc_index == 0) = mXnc;

% Bestemmer her hvilke 'X'er og 'Xnc'er, hvis rækker er med i 'h_X' og
% 'h_Xnc'.
X_frak_nr = ceil(h_X/mX);
Xnc_frak_nr = ceil((h_Xnc-ant_frak*mX)/mXnc);

% Opbygger her den samlede designmatrix, XS, for samtlige fraktiler. Hvert
% af de enkelte 'X' ligger i den øvre diagonal af 'XS' og 'Xnc' ligger i
% den nedre diagonal. Derfor bestemmes et indryk for hvert af 'X''erne og
% 'Xnc''erne.
X_forskydning = (X_frak_nr-1)*nX;
Xnc_forskydning1 = zeros(sum(Xnc_frak_nr < 3),1);
Xnc_forskydning2 = (Xnc_frak_nr(Xnc_frak_nr>2)-2)*nX;
Xnc_forskydning = [Xnc_forskydning1; Xnc_forskydning2];

% Antal rækker i første og sidste 'Xnc' i den nedre 'XS'.
ant_1_Xnc = sum(Xnc_frak_nr==1);
ant_sidst_Xnc = sum(Xnc_frak_nr==ant_frak+1);

% Angiver her alle ikke-nul-elementer i den nedre 'XS' i en vektor.
Xval = X(X_index,:)';
Xval = Xval(:);

% Angiver her søjle-placeringerne for alle ikke-nul elementerne i den øvre
% 'XS'.
JX = zeros(length(X_index)*nX,1);
for i = 1:length(X_index)
    JX((i-1)*nX+1:i*nX) = X_forskydning(i)+1:X_forskydning(i)+nX;
end

% Angiver her række-placeringerne for alle ikke-nul elementerne i den øvre
% 'XS'
IX = repmat(1:length(X_index),nX,1); IX = IX(:);
```

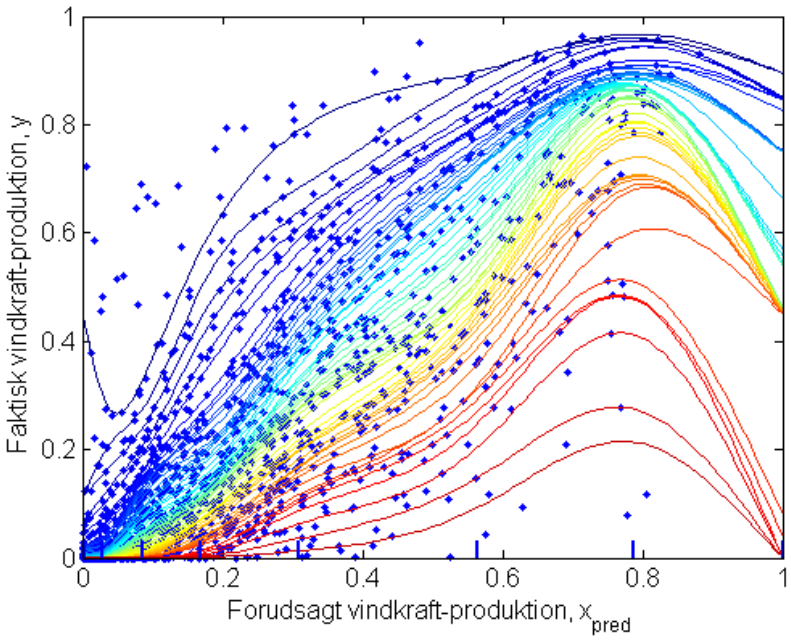
```
% Nedre XS.
% Angiver her alle ikke-nul-elementer i den nedre 'XS' i en vektor.
Xncval1 = Xnc(Xnc_index(1:ant_1_Xnc),:)';
Xncval2 = [-Xnc(Xnc_index(ant_1_Xnc+1:length(Xnc_index)-ant_sidst_Xnc),:)...
    Xnc(Xnc_index(ant_1_Xnc+1:length(Xnc_index)-ant_sidst_Xnc),:)]';
Xncval3 = -Xnc(Xnc_index(length(Xnc_index)-ant_sidst_Xnc+1:length(Xnc_index))
    ,:)';
Xncval = [Xncval1(:); Xncval2(:); Xncval3(:)];

% Angiver her søjle-placeringerne for alle ikke-nul elementerne i den nedre
% 'XS'.
JXnc = zeros(nX*(2*length(Xnc_index)-(ant_sidst_Xnc+ant_1_Xnc)),1);
start = 1;
for i = 1:ant_1_Xnc
    JXnc(start:start+nX-1) = Xnc_forskydning(i)+1:Xnc_forskydning(i)+nX;
    start = start+nX;
end
for i = ant_1_Xnc+1:length(Xnc_index)-ant_sidst_Xnc
    JXnc(start:start+2*nX-1) = Xnc_forskydning(i)+1:Xnc_forskydning(i)+2*nX;
    start = start+2*nX;
end
for i = length(Xnc_index)-ant_sidst_Xnc+1:length(Xnc_index)
    JXnc(start:start+nX-1) = Xnc_forskydning(i)+1:Xnc_forskydning(i)+nX;
    start = start+nX;
end

% Angiver her række-placeringerne for alle ikke-nul elementerne i den nedre
% 'XS'.
IXnc1 = repmat(length(X_index)+1:length(X_index)+ant_1_Xnc,nX,1);
IXnc2 = repmat(length(X_index)+ant_1_Xnc+1:ant_rows-ant_sidst_Xnc,2*nX,1);
IXnc3 = repmat(ant_rows-ant_sidst_Xnc+1:ant_rows,nX,1);
IXnc = [IXnc1(:); IXnc2(:); IXnc3(:)];

% Samler nedre og øvre del af 'XS' på sparse triplet-form.
XS = sparse([IX; IXnc], [JX; JXnc], [Xval; Xncval]);
```

A.9 $NC_{singleNO}$



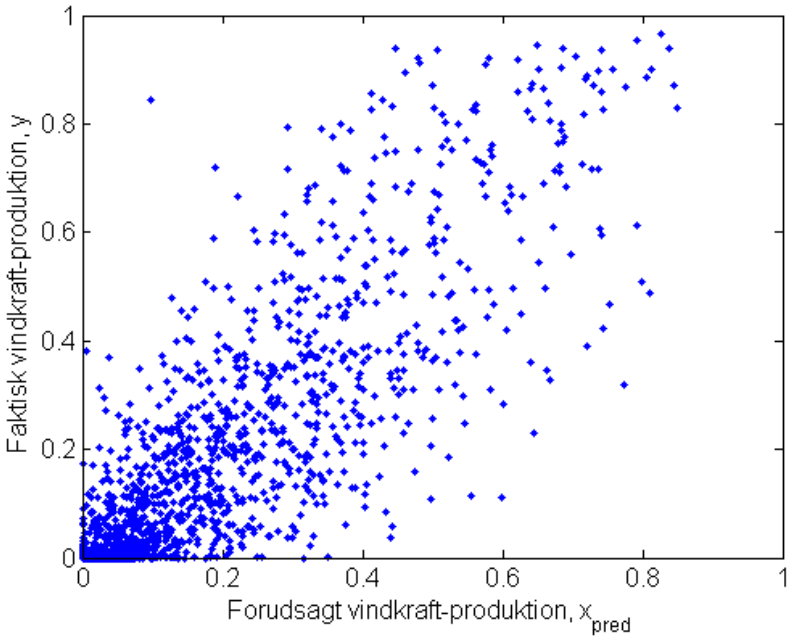
Figur A.1: Metoden $NC_{singleNO}$, hvor der er estimeret nedefra og op.

Det endelige estimatet i figur 10.5 fås ved at tage gennemsnittet af estimaterne i de to figurer. Det ses, at når der estimeres nedefra og op, ligger nogle af de midterste fraktilkurver meget tæt, hvilket skaber stort mellemrum mellem andre midter-fraktiler.



Figur A.2: Metoden $NC_{singleNO}$, hvor der er estimeret oppefra og ned.

A.10 Testsættet for 24-timers-horisonten



Figur A.3: Her ses testsættet for 24-timers horisonten.