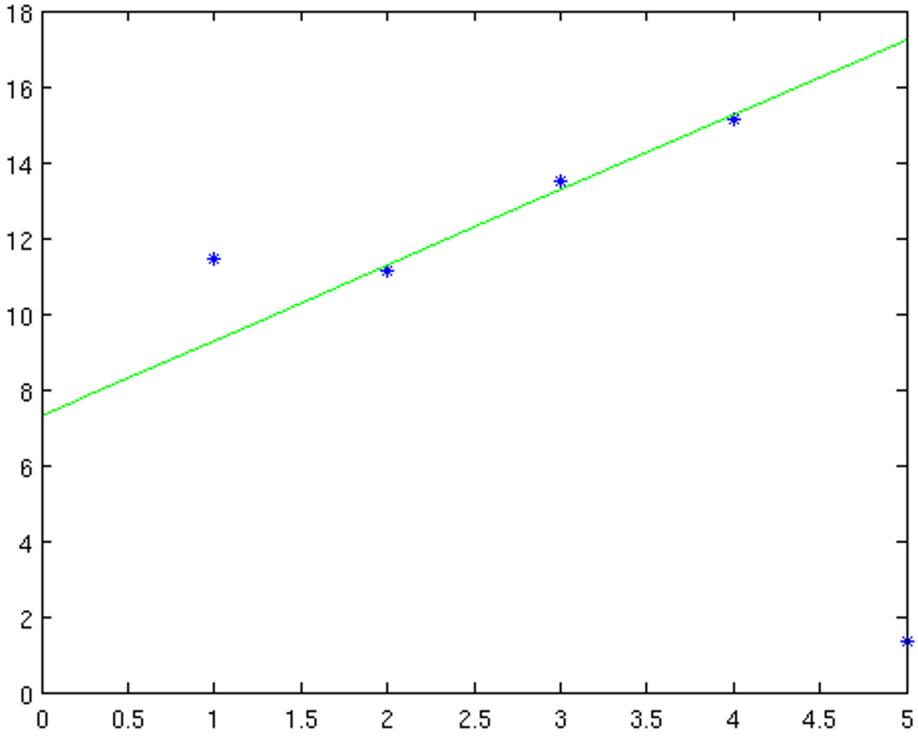
Sammenligning af robuste udjævnings metoder



af
L10MS.02
Jacob Collstrup

Titel:	Sammenligning af robuste udjævningsmetoder
Projekt periode:	1. Februar til 10. Juni 2010
Gruppe:	L10MS.02
Gruppe medlemmer:	Jacob Collstrup
Vejledere:	Jens Peter Cederholm & Jens Juhl
Oplagstal:	4
Side antal:	47
Bilags antal:	72
Bilag på CD:	72

Synopsis

I betragtning af at der eksisterer et multiplum af robuste udjævningsmetoder, kan det være relevant at gøre sig tanker om hvilke der er bedst, evt. i hvilke situationer. For at besvare dette spørgsmål, laves der et oprids over nogle få tilgængelige robuste udjævningsmetoder. Disse præsenteres med styrker og svagheder. Herefter opridses problemformuleringen til projektet.

I næste trin gennemgås to udvalgte robuste udjævningsmetoder i detaljer, hvor der gøres overvejelser om hvordan eventuelle tærskelværdier skal sættes, samt hvilke landinspektør opgaver som udjævningsmetoderne skal testes på. Herefter beskrives hvordan dataene, der skal arbejdes på, er genereret, hvilke overvejelser der er gjort i forbindelse med generering af data. Dette afsnit følges op af et implementeringsafsnit, som beskriver, ved hjælp af figurer, hvordan udjævningsmetoderne er implementeret i matlab.

Herefter foretages der en række forsøg, på baggrund af forskellige datasæt. Rådata, fra forsøgene præsenteres også her. Derefter følger en gennemgang af alle resultaterne, tendenser udledes og overraskende resultater, forsøges forklaret. Til slut præsenteres projektets konklusion, jvf. problemformuleringen.

Abstract

Considering the fact that there exists a lot of robust adjustment methods, it's relevant to consider what method is applied to the problem. To answer this question a short presentation of available robust adjustment methods is presented. These are presented with strengths and weakneses. After this the problemfurmulation for the project is presented.

In the next step two chosen robust adjustment methods are presented in detail along with considerations regarding thresholds for those methods and what surveyor tasks the robust adjustment methods are testet on. After this comes a description of how the test data is generated and what considerations are done in regards to this. This part is followed up by an implementation part, detailing how the chosen robust adjustment methods are implemented in matlab, using flowcharts.

After this a series of experiments are conducted on the test data. The raw data from these experiments are presented here. After that there is a follow up on the results, detailing tendencies and unexpected results are sought to be explained. The project is rounded off with a conclusion tying to the problemformulation.

Forord

Dette afgangsprojekt er skrevet af gruppe L10ms02 bestående af Jacob Collstrup, på landinspektør uddannelsens Measurement Science program 2010. Jeg vil i den forbindelse gerne sige tak til vejledere, Peter Cederholm og Jens Juhl, som har været meget behjælpelige i forbindelse med arbejdet af dette projekt. I dette projekt er der anvendt punktum, som decimal separator. Det vil sige at ½ skrives som 0.5 i decimaler. Referencer til litteratur er lavet som fodnoter.

Bilagshenvisninger er lavet som kantet parentes, med ordet 'Bilag' efterfulgt af et nr. En henvisning til bilag 1, vil fx se således ud [Bilag 1]

Indholdsfortegnelse

1 Indledning....4
2 Foranalyse....7
2.1 Mindste kvadraters princip....7
2.2 Data snooping....7
2.3 IRLS....9
2.3.1 L1-normen....9
2.3.2 Huber....11
2.3.3 Bisquare....11
2.4 RANSAC....12
2.5 Andre estimatorere....13
2.6 Robust kvalitetsvurdering....13
2.7 Vurderingspunkter....14
3 Problemformulering....15
3.1 Hvilke kriterier indgår i projektet....15
3.2 Udjævningsopgaver....16
Opgaver i projektet....18
3.3 Udjævningsmetoder....18
4 Teori....19
4.1 Beskrivelser af de udvalgte udjævningsmetoder....19
4.2 IRLS....19
4.3 RANSAC....19
5 Data udvælgelse....23
5.1 Data til udjævning af afstand....23
5.2 Data til udjævning af ret linje i 2d....23
5.3 Data til udjævning af 2d transformation....24
6 Implementering....25
6.1 Mindste Kvadraters Metode + data snooping....25
6.2 IRLS....26
6.3 RANSAC....27
6.4 RANSAC + IRLS + MKP....29
7 Forsøg....31
7.1 Forsøgsparametre....32
7.2 Afstand....32
7.3 Ret linje....34
7.4 Helmert....35
8 Resultater....37
8.1 Succesrate....37
8.2 Problemsøgning....41
8.2.1 Normalisering af residualer....41
8.2.2 Vægtfunktionen....41
8.2.3 Andre forhold....45
8.3 Data Snooping resultater....46

8.4 IRLS resultater....46
8.5 RANSAC resultater....48
8.6 RIL resultater....49
8.7 Samlet overblik....50
9 Konklusion....51

1 Indledning

En del af arbejdet i landinspektørfaget består, i at modellere virkeligheden, gennem kort og 3d modeller. For at lave disse kort og 3d modeller indsamles der data, med forskellige instrumenter. Et fælles karaktertræk for alle disse instrumenter, er at deres målinger er påvirket af fejl. Disse fejl inddeles i tre kategorier, da de behandles på forskellige måder: Systematiske fejl, tilfældige fejl og grove fejl. Systematiske fejl er karakteriseret ved, at de kan modelleres, og derved kan deres indflydelse på dataene fjernes. Alternativ kan nogen af disse systematiske fejls indflydelse, på målingen, fjernes ved at anvende en bestemt målemetode. Tilfældige fejl kan der ikke gøres noget ved og de kan ikke fjernes fra dataene helt. Tilfældige fejl er karakteriseret ved at være normalfordelte og deres indflydelse på målingen kan minimeres, gennem udjævning. Tilfældige fejl kan også forklares som målestøj. De grove fejl skyldes bl.a. brugerfejl, forkert metode eller anden menneskelig indflydelse, eller hardware fejl. Problemet med disse fejl er, at de ødelægger dataene og dermed resultatet af målingen. Derfor er det vigtigt at få dem fjernet, inden at dataene beregnes. Problemet er at det kan være svært, at finde disse fejl.

Når data er indsamlet, beregnes de ofte, efter mindste kvadraters princip. Normalt når der måles i landinspektørfaget, måles der ikke direkte på den størrelse der søges. En undtagelse kan være afstandsmåling og GPS-måling. Her kan det godt være afstanden eller koordinaterne i sig selv, som er interessante. En situation kan være at man måler en masse punkter i 2d, som ligger på en ret linje. Det kan være en skelgrænse, som skal måles.

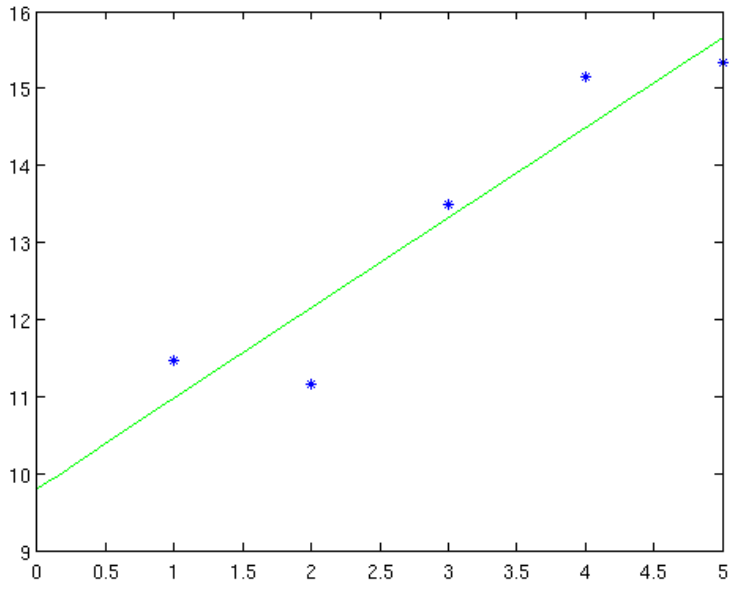


Illustration 1:En typisk udjævningssituation, hvor der ud fra en række punkt koordinater i 2d, skal findes en 2d linie.

På grund af de tilfældige fejl, ligger punkterne ikke helt på linjen. For at finde frem til linjens ligning benyttes mindste kvadraters princip. Gennem mindste kvadraters princip udjævnes der, på en sådan måde at alle punkterne får indflydelse på linjen. Herved kan eventuelle overbestemmelser anvendes.

Et problem for udjævningen er, hvis der opstår en situation, hvor der er grove fejl i datasættet.

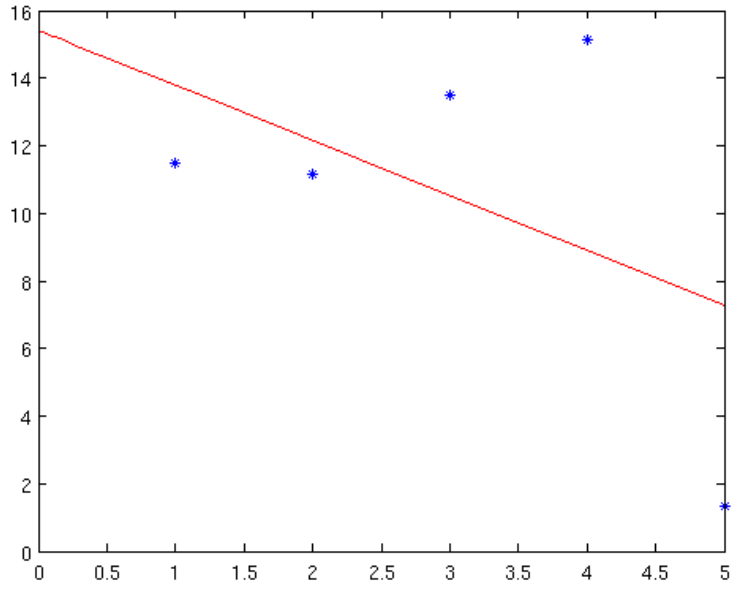


Illustration 2:Denne situation er den samme, som den på forrige side. Her er der dog en grov fejl på det sidste punkt, som ødelægger udjævningen.

I den skitserede situation er der en grov fejl i måledataene. Dette medfører at hvis der foretages udjævning på dette datasæt, uden at denne fejl fjernes, vil resultatet blive forkert:

I denne situation bliver det der kaldes robust udjævning relevant. Robust udjævning er en måde at udjævne på, hvor indflydelsen af grove fejl enten minimeres eller fejlene fjernes helt. Nogle robuste udjævningsmetoder baserer sig på alm. udjævning efter mindste kvadraters princip, mens andre gør brug af helt andre metoder til at løse problemet. Det problem som robust udjævning søger at løse er skitseret nedenunder:

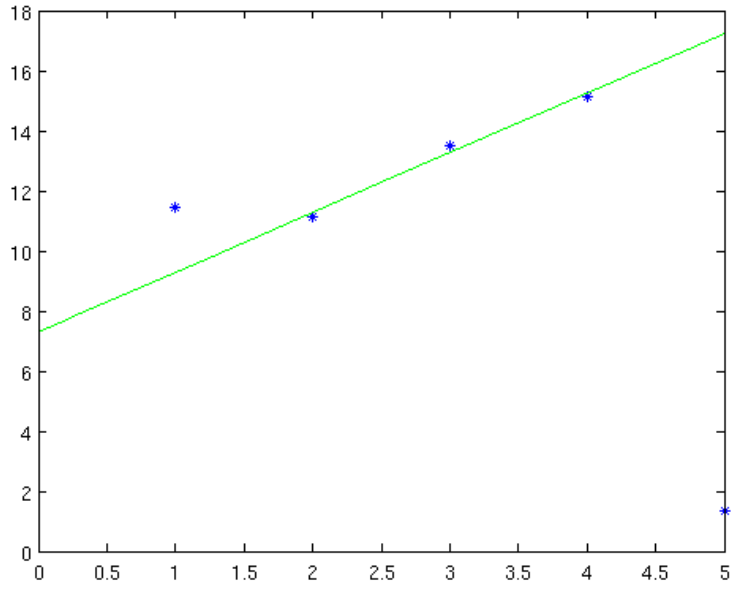


Illustration 3:I denne situation, er der foretaget robust udjævning, på det samme datasæt, som ovenover. Her findes der frem til den rigtige linje.

I denne situation er der foretaget robust udjævning, af det samme data, som i foregående figur. Det er her lykkedes den robuste udjævningsmetode, at finde og fjerne den grove fejl, så den ikke længere får indflydelse på beregningerne.

Undertegnede har gennem undervisningen på Landinspektørstudiets Measurement Science master program, stiftet bekendtskab med nogle af disse robuste metoder. Andre metoder er lært gennem selvstudie og andre aktiviteteter.

Det er interessant, at der på trods

af fejl i dataene, stadig kan opnås et godt eller brugbart resultat i udjævningen. Derfor kunne det være interessant at arbejde, med robust udjævning i specialet, med henblik på at opnå et dybere indblik i hvordan de virker og hvor godt de virker. Projektet er motiveret af en personlig interesse for robust udjævning, dette bunder i en fascination, af at det er muligt at opnå gode resultater, med dårlig data. Fascinationen ligger i at det er muligt at få en computer til automatisk at finde den delmængde af dataen der er god og beregne på det. For undertegnede virker det som en blanding af AI (Artificial Intelligence = kunstig intelligens) og mønstergenkendelse. Disse to beskrivelser er valgt, da det for undertegnede går ud på at sætte en computer i stand til at finde et mønster, i data, hvor det ikke er alle data der repræsenterer mønsteret. Herefter skal computeren foretage et fornuftigt valg, for hvilke data som repræsenterer mønsteret, og hvilke data der ikke gør.

Som nævnt vil projektet fokusere på hvor godt metoderne virker. For at finde ud af dette, bliver det undersøgt hvilke metoder der er, for at danne et overblik over emnet.

Det næste trin i projektet er, at lave en foranalyse, som præsenterer nogle forskellige metoder, til at lave robust udjævning. Denne foranalyse forklarer lidt om metodernes styrker og svagheder, samt et kort indblik i hvordan de virker. Når dette er gjort, klarlægges problemformuleringen, for projektet. Projektet, går kort fortalt ud på at sammenligne nogle robuste udjævningsmetoder. I afsnittet med problemformuleringen indgår hvilke undersøgelser der skal laves, for at besvare problemformuleringen. Dertil indgår det også hvilke robuste udjævningsmetoder, der arbejdes videre med. Når problemformuleringen er klarlagt, beskrives de robuste udjævningsmetoder, som skal undersøges, i høj detaljeringsgrad. Når dette er gjort vil der bliver lavet forsøg, eller eksperimenter, med de udvalgte algoritmer. Disse eksperimenter bliver tilrettelagt, med henblik på finde ud af hvilken, af de robuste udjævningsmetoder der er bedst, ud fra det eller de kriterier, som problemformuleringen opstiller. Resultatet af disse undersøgelser, og hvad det måtte betyde, bliver behandlet i den afsluttende konklusion.

2 Foranalyse

Fra eksemplerne i indledningen kan det ses, at det er et problem for udjævningen, hvis der er grove fejl i datasættet. Derfor blev begrebet 'robust udjævning' introduceret. Her præsenteres nogle robuste udjævningsmetoder, med deres fordele og ulemper. Der startes med en kort introduktion, med mindste kvadraters princip.

2.1 Mindste kvadraters princip

Udjævning efter mindste kvadraters princip er den måde overbestemmelser bliver håndteret på, i landmålingen. Metoden går ud på at minimere summen af kvadrerede residualer. Indenfor landinspektørfaget regnes mindste kvadraters princip for at være den rigtige løsning på et udjævningsproblem. Metoden har følgende grundlæggende formel indenfor lineær algebra:

$\hat{x} = (A^T WA)^{-1} A^T Wb$, hvor A er designmatricen, W er matricen med vægtene til de forskellige observationer, b er observations vektoren og $\hat{x}$ er løsningen. Residualerne fås af følgende formel:

$r = A \cdot \hat{x} - b$, hvor A er designmatricen, $\hat{x}$ er løsningen på ligningssystemet, b er observationsvektoren og r er residualvektoren. Som det er demonstreret i indledningen er denne måde at udjævne på ikke robust, hvis der indgår grove fejl i datasættet. Det vil sige at der er fejl i observationsvektoren, som ikke skyldes normalfordelt støj, eller stor støj, som ikke er vægtet korrekt. For at håndtere dette er der en række robuste udjævningsmetoder til rådighed. De fleste af disse metoder benytter sig på en eller anden måde af mindste kvadraters princip. Et mindste kvadraters estimat, kendes også under navnet L_2-estimat. En god egenskab ved MKP er at man kan kvalitetsvurdere udjævningen ud fra følgende formel: $\frac{r^T r}{m-n} = {\sigma_0}^2$, som hedder spredningen på vægtenheden. Denne størrelse fortæller noget om hvor godt de ubekendte er bestemt. Denne vurdering er dog, ligesom MKP selv, ikke robust. Dette ligger i at det er de kvadrerede residualer der indgår, i formlen. Jo mindre spredningen på vægtenheden er, jo bedre er de ubekendte bestemt.

2.2 Data snooping

Data Snooping er en principielt simpel metode. Kort beskrevet går det ud på at de normaliserede residualer udregnes. De normaliserede residualer udregnes ved hjælp af kovarians matricen. Denne findes på følgende måde:

$$Qr = W^{-1} - A \cdot (A^T \cdot W \cdot A)^{-1} \cdot A^T$$

Qr matricen er en kvadratisk matrice, i hvilken diagonal elementerne skal bruges til at udregne de normaliserede residualer. Dette foregår på følgende måde:

$w_i = \frac{r_i}{\sqrt{{qr_i}^2}}$ hvor w_i er det i'te normaliserede residual, r_i er det i'te residual og qr_i er den i'te indgang i Qr matricens diagonal. Når de normaliserede residualer er regnet, testes hvert residual for sig. Typisk

vil normaliserede residualer på mere end ca. ± 3 gange spredningen blive sorteret fra, da dette stemmer overens med forståelsen af et en grov fejl ligger over 3 gange spredningen. Dette medfører en risiko for to uheldige situationer. Der er en risiko for at frasortere en observation med et normaliseret residual på ca. 3 gange spredningen, som ikke er en grov fejl. Samtidig er der en risiko for at acceptere en observation med en grov fejl, med et residual på mindre end 3 gange spredningen. Denne måde at foretage robust udjævning på medfører en række problemer. For hver grov fejl der er i datasættet skal hele udjævningen foretages om. Dette kan beregningsmæssigt blive en omkostningsfuld affære, hvis det fx er laserscanningsdata der udjævnes på. Eller et andet system, med mange observationer og forhøjet risiko for grove fejl. Hertil kommer at metoden i bund og grund er en MKP algoritme, med en ekstra løkke, til at frasortere formodede grove fejl. Problemet ligger i at en grov fejl vil 'trække' løsningen hen mod sig. Se evt. figur 2 i indledningen, hvor linjen er 'trukket' ned mod den grove fejl. Dette medfører en risiko for at en grov fejl kan 'trække' så meget i løsningen, at observationer der ikke er en grov fejl, kommer til at ligne grove fejl, pga. residualets størrelse.

2.3 IRLS

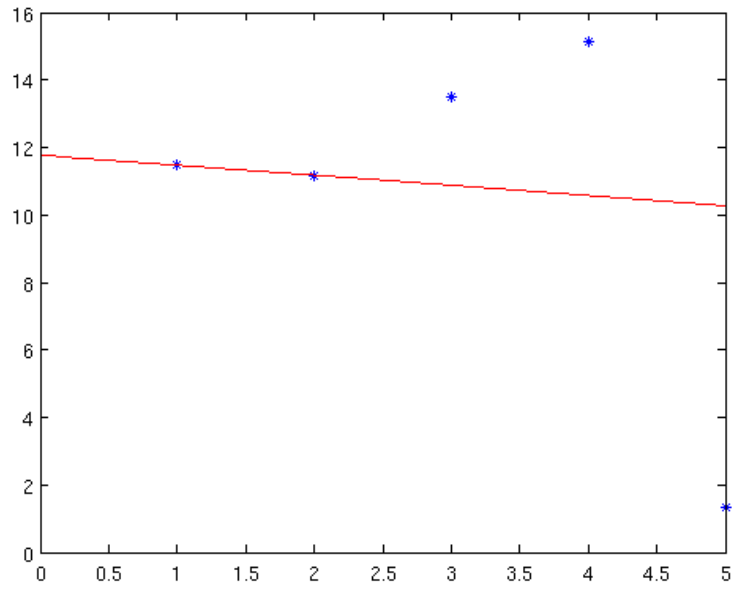


Illustration 4: I denne udjævning er der brugt IRLS, på et datasæt med en grov fejl. På grund af at L2-estimatet er gået galt, finder IRLS ikke frem til den rigtige løsning.

IRLS, står for 'Iteratively Reweighted Least Squares'. IRLS er mere en teknik end en udjævningsmetode. Teknikken går på at der for hver iteration vægtes på baggrund af residualerne fra forrige iteration. Det vil sige at der i hver iteration udregnes et nyt estimat på baggrund af forrige estimat. Fejlene i estimaterne bruges til at vægte observationerne i næste estimat. Observationer som medfører store residualer nedvægtes, mens observationer der medfører små residualer får stor vægt. På denne måde holder IRLS algoritmer fast i de gode observationer, mens dårlige observationer udvægtes. Selvom IRLS regnes som en robust udjævningsmetode, har den en svaghed. Den starter med et et alm. L_2-estimat. Hvis en grov fejl er så stor at dette estimat er helt forkert, vil IRLS aldrig kunne få rettet op på det.

Kort fortalt er IRLS en gentagelse af følgende formel: $\hat{x} = (A^T WA)^{-1} A^T Wb$, hvor der for hver iteration udregnes en ny vægtmatrix, W.

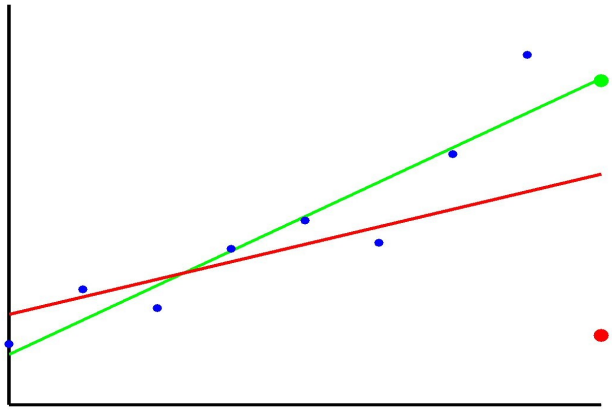


Illustration 5: Illustrationen viser hvad der sker nå der udjævnes med L1-normen, uden fejl i x-aksens retning (grøn linie) og med fejl i x-aksens retning (rød linie).

2.3.1 L_1-normen

Én af de robuste vægte som er baseret på MKP er L_1-normen. Her anvendes den reciprokke af residualernes absolutte værdi, som vægte i W-matricen: $W_i = \frac{1}{|r_i|}$, hvor r_i indikerer det i'te residual. Et problem med denne måde at vægte på er at den ret hurtigt får vægtet observationerne ud, som ikke ligger meget tæt på middelværdien. Risikoen her er at man risikerer at nedvægte gode observationer alt for meget[1].

De næste to vægtfunktioner der præsenteres er Huber og derefter Bisquare. Disse to vægtfunktioner tilhører den kategori af robuste metoder, som kaldes m-estimatorer. M står for 'Maximum-likelihood'[2]. Det er ingen forskel mellem at skrive M-estimator eller m-estimator. Et m-estimat er et mindste kvadraters estimat, hvori der i beregningerne har indgået en m-estimator, fx Huber.

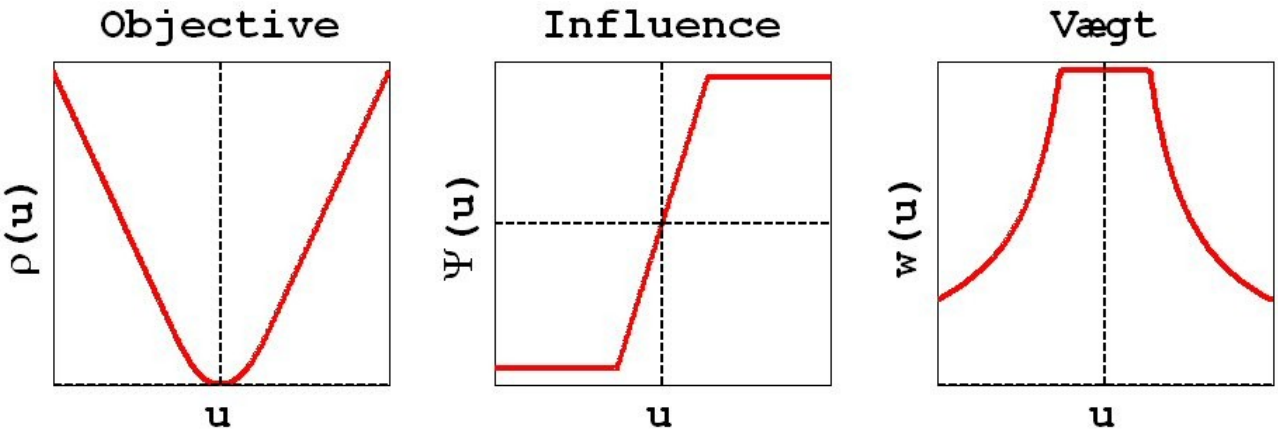


Illustration 6: Illustrationen viser nogle egenskaber ved Huber estimatoren. Især sidste figur 'vægt' er vigtig.

2.3.2 Huber

Huber er en af de tidligste robuste estimatorer[3]. Huber-estimatoren er en kombination af L_2-normen og L_1-normen. Residualer som ligger under en given tærskelværdi, vægtes efter L_2-normen, mens residualer der ligger over denne tærskel værdi vægtes efter L_1-normen. I figuren til højre, nedenunder

1 [P. Cederholm, Modelling Based on Coordinates, kursusgang 4, slide 4, 2009]
2 [2002 John Fox, Robust Regression – Appendix to An R and S-PLUS Companion to Apllied Regression, s. 1]
3 [P. Cederholm, Modelling Based on Coordinates, kursusgang 4, slide 9, 2009]

ses hvordan der vægtes. Længden af den vandrette linie i toppen er tærskel værdien. Uden for denne værdi vægtes efter L_1-normen. Denne tærskel værdi kaldes 'tune' værdi, t. Til dette bruges de normaliserede residualer. Hvis det normaliserede residual er mindre end tune-værdien vægtes der efter L_2-normen, mens der nedvægtes efter L_1-normen når det normaliserede residual er større end tune-værdien. En væsentlig forskel mellem Huber og L_1-normen er at Huber-estimatoren ikke nedvægter på residualer nær u (middelværdien), som L_1-normen gør. Tune-værdien afgør hvor langt residualet skal være fra u for at Huber-estimatoren begynder at nedvægte den pågældende observation. 'u' henviser til 'u', i figuren under overskriften 'vægt'.

Første graf (fra venstre) viser hvad det er der skal minimeres og hvordan. Den midterste graf viser at der er tale om en 'bounded influence function'. Det betyder, at selvom residualerne vokser og vokser får de ikke større indflydelse på løsningen af den grund. Den sidste graf, til højre, viser vægt funktionen. Her kan man se hvilke vægte residualerne for tildelt vægte alt efter deres størrelse. Et muligt problem med Huber kan være at når nedvægtningen begynder, sker dette for hurtigt. Samtidig kan 'knækkene' på vægt kurven, medføre nogle logiske problemer. Hvis 'knækket eksempelvis ligger på ±2. Hvis et residual ligger på 1.9, bliver det ikke vægtet ned, mens et residual på 2.2 risikerer en kraftig nedvægtning. Dette uden at der er den store forskel på residualernes størrelse.

2.3.3 Bisquare

Bisquare er en anden m-estimator, som også går under navnet Tukey og Biweight eller en kombination heraf[4]. Ligesom andre m-estimatorer søger den at adressere det problem der er med at genvægte efter L_1-normen, netop at genvægtningen er meget kontant[5]. Hvor Huber vægter alle residualer der ligger inden for tuningskonstanten, med 1, har Bisquare en mere 'flydende' vægtning.

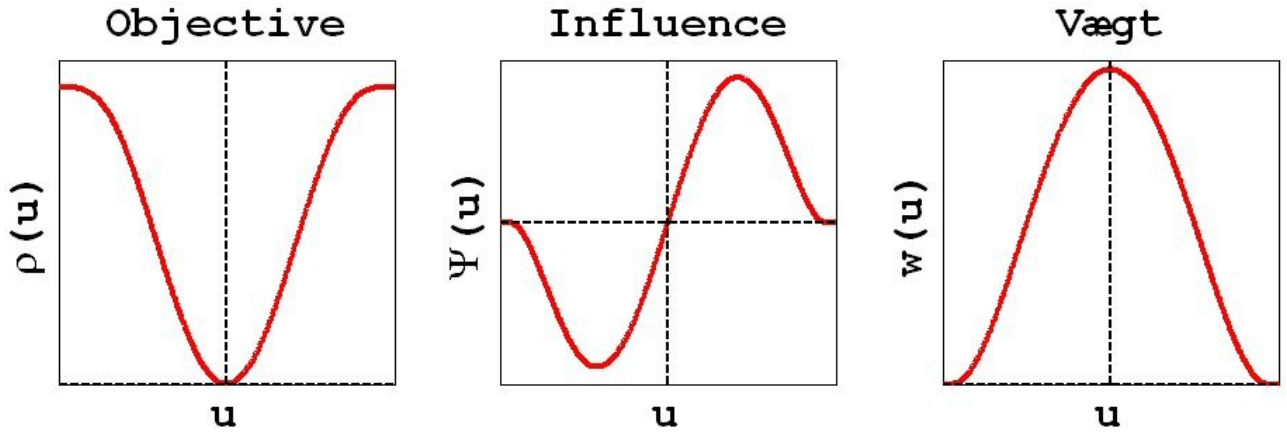


Illustration 7: Illustrationen viser nogle egenskaber ved Bisquare estimatoren, især sidste figur 'vægt' er vigtig.

Ligeså snart residualerne bevæger sig væk fra middelværdien, begynder nedvægtningen, men dog ikke så kraftigt som ved L_1-normen, samtidig med at man opnår mindre residualer end med Huber.

4 [2002 John Fox, Robust Regression – Appendix to An R and S-PLUS Companion to Apllied Regression, s. 2]
5 [P. Cederholm, Modelling Based on Coordinates, kursusgang 4, slide 4, 2009]

2.4 RANSAC

Som et alternativ til de overnstående metoder, som er baseret på MKP, er der nogle værktøjer, som fungerer på helt andre måder. RANSAC (RANdom SAmple and Consensus) er en robust metode til bestemmelse af parametre i en given model. Denne måde er baseret på hvad man kan kalde 'trial and error', heri ligger at RANSAC er en iterativ metode. RANSAC kan fx bruges til at estimere en ret linje i 2d, ud fra et datasæt. I dette eksempel estimeres linjen ud fra et datasæt med 20 punkter. I første iteration udtrækkes 2 tilfældige punkter og der opstilles en ligning for linjen, med de to punkter, en såkaldt 'model' i RANSAC terminologi. Herefter opstilles der et mål for hvor godt resten af punkterne passer med denne linje. Denne overensstemmelse kaldes en 'afstandsfunktion' i RANSAC. I dette eksempel opstiller RANSAC algoritmen et udtryk for hvor langt der er fra de resterende 18 punkter og ind til linjen. Herefter gemmes både model og resultatet fra 'afstandsfunktionen'. I næste iteration udvælges to nye tilfældige punkter og der opstilles en ny model. Hvis 'afstandsfunktionen' kommer frem til at den nye løsning passer bedre end den forrige, forkastes den forrige og den nye gemmes. Oftest, som en del af successkriteriet, skal en hvis procentdel af de resterende punkter passe med linjen, inden for en given tærskel værdi, før RANSAC anser det som en success. Med det menes at afstandsfunktionens resultat skal ligge under en hvis tærskelværdi, samtidig med at en hvis procentdel af de testede punkter stemmer overens med denne model. Denne måde at prøve sig frem på, går RANSAC i stand til at undgå det problem, som er illustreret under IRLS, hvor et dårligt udgangspunkt, ødelægger resultatet. Et dårligt udgangspunkter for RANSAC ødelægger ikke de følgende beregninger, da disse ikke har noget med hinanden at gøre.

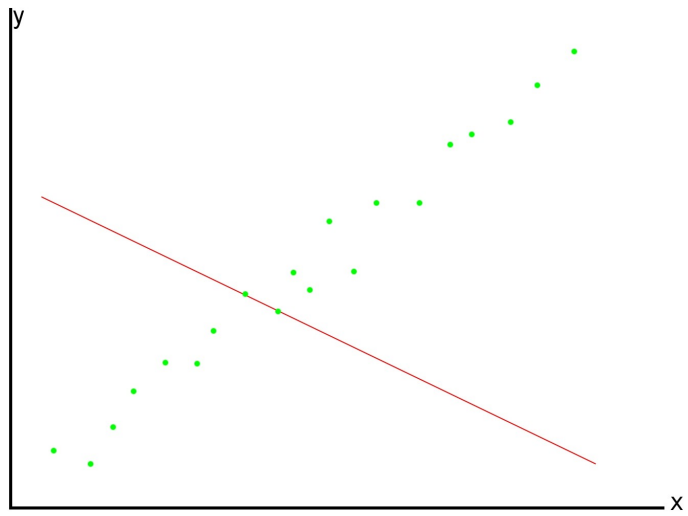


Illustration 8: Billedet viser hvordan linien fx kan ligge, efter RANSACs første iteration. Afstandsfunktionen måler hvor langt, de andre punkter ligger fra denne linje.

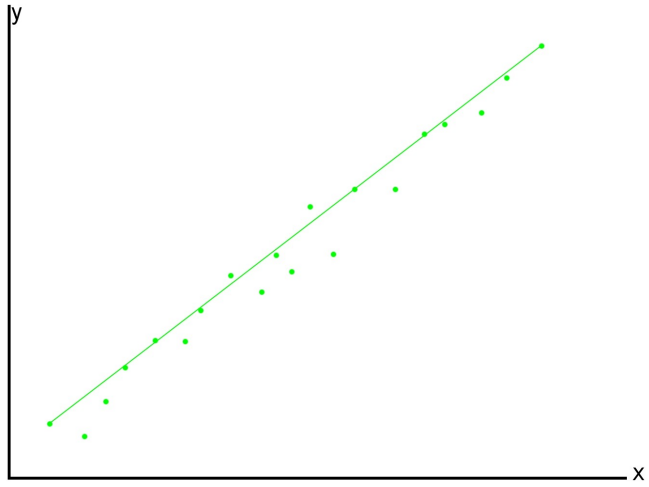


Illustration 9: Billedet viser den linje RANSAC er nået frem til, efter at have testet al data, for at finde den bedste løsning.

I dette eksempel kunne et succeskriterie være at

mindst 16 af de 20 punkter (80%) skal passe med linjen inden for en tærskel værdi af fx 1cm. Ud fra alle løsninger som de 20 punkter præsenterer, vil RANSAC finde den første, eller den bedste der opfylder dette kriterie, hvis denne løsning eksisterer. En vigtig detalje ved RANSAC er at den kan implementeres på to grundlæggende forskellige måder, hvad angår succeskriteriet. Den kan programmeres til, systematisk at gå alle data igennem, og finde den bedste løsning, helt uden et succeskriterie. RANSAC vil i dette tilfælde returnere den bedste model (linjens ligning), samt et udtryk for den fejl denne ligning medfører. Efterhånden som mængden af overbestemmelserne stiger, jo flere iterationer skal RANSAC igennem. Det betyder at, det måske ikke kan betale sig at finde den bedste løsning i sine data, da antallet af iterationer kan medføre at det tager for lang tid at nå resultatet. Et alternativ hertil er at man, som i ovennævnte eksempel, opstiller et succeskriterie, fx 80% af punkterne skal passe bedre end 1cm. Her vil RANSAC afbryde, ligeså snart dette kriterie er opfyldt. Dette medfører umiddelbart to risici:

1. Det kan være at der ikke eksisterer en løsning i det pågældende datasæt, som opfylder dette kriterie.
2. Det kan være at der eksisterer en bedre løsning i datasættet.

2.5 Andre estimatorere

I forbindelse med robust udjævning er det værd at nævne, at der eksisterer andre robuste udjævningsmetoder, end dem der er beskrevet ovenover. Her i blandt r- og s-estimatorer, det er dog udenfor dette projekts ressourcer at introducere disse. De skal dog nævnes.

2.6 Robust kvalitetsvurdering

Som nævnt under Mindste kvadraters princip, er der en måde at vurdere en alm. udjævning på, som ikke er robust. På grund af dette er det nødvendigt at finde et udtryk for kvaliteten af udjævningen, som gælder for robust udjævning. Disse eksisterer der mange af[6]. I forbindelse med studiet er der stiftet bekendtskab med en måde at gøre dette på, hvorfor denne metode bruges:

$s = \frac{med\left(\left|r - med\left(r\right)\right|\right)}{0.6745}$, hvor skalaren er valgt så $s \approx \sigma_0$ når antallet af observationer er stort.

2.7 Vurderingspunkter

Som det kan ses af præsentationen, er der adskillige metoder til at foretage robust udjævning. Derfor er det naturligt at spørge sig hvilken metode man skal bruge i en given situation. I betragtning af at nogle af metoderne virker på vidt forskellige måder kunne man også med rimelighed forestille sig at nogen metoder måske er bedre end andre. For at komme videre er det relevant at spørge om hvad der gør en robust udjævningsmetode bedre end en anden. Herunder er et udvalg af forslag til hvilke kriterier udjævningsmetoderne kunne vurderes på, ikke i nogen bestemt orden:

6 [P. Cederholm 2009, 'Modelling Based on Coordinates', kursusgang 4, slide 7]

- Beregningstid
- Robusthed
- Særlige forhold
- Implementeringslethed/kompleksitet

Beregningstid

Dette kriterium vurderer hvor hurtigt en robust udjævningsopgave kan udføres. Det drejer sig om hvor meget tid der skal bruges på at opnå et brugbart resultat. Nogen metoder kan justeres til at være meget robuste, idet de kan finde et brugbart resultat, selvom der er mange og store grove fejl i datasættet, disse udjævningsmetoder kræver dog, som regel meget beregningstid[7]. Et problem med beregningstid er at alt efter hvilket programmeringssprog de skrives på, kan de køre hurtigere eller langsommere end i andre programmeringssprog. Fx vil en Fortran-kode køre hurtigere end en matlab-kode[8].

Robusthed

Dette omfatter udjævningsmetodens evne til at finde en brugbar løsning i datasættet, selvom der er grove fejl i. En udjævningsmetode, som kan håndtere 20% grove fejl i et datasæt kan kaldes mere robust end en udjævningsmetode, som kan håndtere 10% grove fejl i datasættet. Her måles hvor mange procent dårlig data der skal til, før udjævningsmetoden ikke længere fungerer. Dårlig geometri kan også få en robust udjævningsmetode til at stoppe med at virke. Spørgsmålet er her om det er relevant at undersøge dette aspekt også. Det virker umiddelbart ikke sådan, da geometrien ikke er et problem der er specielt for robust udjævning, med det menes at 'geometriproblemet' har ikke noget med robust udjævning at gøre, men noget med udjævning i al almindelighed at gøre.

Særlige forhold

Dette kriterium er lidt sværere at tilgå end de andre. Nogen udjævningsmetoder kan fx være meget sårbare i visse tilfælde, som der skal tages højde for. Det kan bl.a. være at hvis man vil approksimere en 2d linie ud fra et 2d punktdatasæt, ville det være en god idé at vælge en udjævningsmetode, som kunne håndtere fejl i begge aksens retninger. Det besværlige ved dette kriterium er at man ikke umiddelbart kan sammenligne forskellige metoder, samtidig med at der skal tages stilling til dette kriterium i den enkelte situation.

Implementeringslethed/kompleksitet

Dette dækker over hvor let en udjævningsmetode er at implementere i en algoritme og hvor kompliceret den er. En udjævningsmetode skal sandsynligvis kun implementeres én gang, men det kan være fordelagtigt at vælge en metode der er nem at implementere, da dette muligvis kan munde ud i

7 [J. Heikkilä, Robust Regression, Graduate course on Advanced statistical signal processing, s. 23]
8 [Jens Juhl, 2010]

færre programfejl. Med kompleksitet menes der hvor kompleks en udjævningsmetode er. Nogle metoder mere komplekse end andre. Det kan, ligesom ved implementeringslethed, vise sig at være fordelagtigt at vælge en simplere løsning, frem for en kompleks. Den simplere løsning er muligvis nemmere at forstå og det er derfor nemmere at opstille tærskelværdier til den, hvis sådanne er indblandet, o.lign. Brugeren/programmøren har muligvis mere føling med en simplere udjævningsmetode.

3 Problemformulering

I betragtning af at der eksisterer mange robuste udjævningsmetoder, er det relevant at spørge om der er en der er bedre end de andre. Dette spørgsmål former kernen af projektet. Der arbejdes på at udforme en komparativ analyse, som skal belyse om en af de præsenterede robuste udjævningsmetoder er bedre end de andre. For at undersøge dette skal der laves en række eksperimenter, hvor de robuste udjævningsmetoder, bliver udsat for forskellige situationer. På baggrund af disse forsøg kan der muligvis tegne sig et billede, af om en af metoderne er bedre, end de andre. Hertil er det vigtigt at gøre sig klart, hvordan udjævningsmetoderne skal vurderes. Dette er et helt centralt spørgsmål der skal besvares før projektet kan komme videre. I foranalysen er der præsenteret en række egenskaber, som metoderne kunne vurderes på. Dette projekt fordrer mange eksperimenter, og spørgsmålet er dertil i sig selv meget stort. På grund af disse forhold er det nødvendigt at foretage nogle afgrænsninger af projektet, for at det kommer ned i en størrelsesorden som er passende for et afgangsprojekt. En af de første afgrænsninger der foretages, er i hvordan udjævningsmetoderne vurderes.

3.1 Hvilke kriterier indgår i projektet

De 3 første kriterier, der bliver nævnt i foranalysen, er af tekniske karakter, hvor det sidste, er af menneskelig karakter. Det vil sige at ved det sidste kriterie vurderes udjævningsmetoderne ud fra hvor let et menneske måtte have ved at benytte sig af dem. Som det blev beskrevet i starten af dette afsnit, kan det være passende at undersøge om en udjævningsmetode er bedre end en anden ud fra et eller flere af de ovenstående kriterier. I betragtning af at beregningstiden for en robustudjævningsmetode, kan afhænge af hvilket programmeringssprog den er skrevet i, vurderes det at være besværligt, næsten umuligt at lave en realistisk sammenligning på baggrund af beregningstiden. Selvom alle algoritmerne i dette projekt bliver skrevet i samme sprog, er det ikke sikkert at rangordnen imellem dem bibeholdes når de implementeres i et andet sprog. Med det skal forstås at i fald IRLS beregner ca. dobbelt så hurtigt som RANSAC i matlab-kode er det ikke sikkert IRLS også vil være dobbelt så hurtig i C-kode. Det vil derfor være svært, måske endda umuligt at drage almene konklusioner på beregningstiden. På grund af dette bliver algoritmerne ikke sammenlignet på baggrund af den påkrævede beregningstid.

I og med at kompleksitet og implementeringslethed, kan være subjektive, vurderes også disse kriterier til at være uegnede for dette projekt. En programmør/bruger kan fx være meget bekendt med s-estimatorer og har slet ikke stødt på IRLS. Selvom IRLS måske, teknisk set, kan være nemmere at implementerer og mindre kompleks, kan denne udjævningsmetode være mere besværlig for programmøren/brugeren. Samtidig er det til dels en subjektiv vurdering om en metode er nem eller

svær at implementere.

Tilbage er der robusthed og særlige forhold, at forholde sig til. Fokusset for sammenligningen af de robuste udjævningsmetoder lægges i dette projekt på kriteriet 'robusthed', men kriteriet 'særlige vilkår' vil sandsynligvis også blive berørt, da det kan influerer implementeringen af en robust udjævningsmetode i en given situation.

3.2 Udjævningsopgaver

For at sammenligne udjævningsmetoderne skal der være et sammenligningsgrundlag. Med det menes der at udjævningsmetoderne skal beregne forskellige opgaver. Projektet skal derfor stille en række udjævningsopgaver, som skal løses, med nogle udvalgte udjævningsmetoder. De opgaver der kommer til overvejelse er opgaver, som genererer meget måledata, med forhøjet risiko for grove fejl, vil blive overvejet og inddraget. Eller opgaver hvor der traditionelt anvendes robust udjævning. Følgende opgaver er derfor relevante at overveje:

- 2d linje ud fra et 2d punktdatasæt
- 2d transformation (2 translationer, 1 rotation, 1 skala, eller kombinationer af disse)
- Approksimering af et plan i 3d ud fra laserscanningsdata
- Polygonudjævning
- Afstand

Hvis alle disse opgaver skal behandles i projektet, kan man hurtigt risikere at skulle lave alt for mange eksperiementer, end der er tid til i projektet. Derfor er det nødvendigt at vælge en eller to opgaver fra. For at finde ud af hvilke opgaver der skal arbejdes videre med, beskrives de kort.

2d linje ud fra et 2d punktdatasæt

Denne opgave går ud på at der skal findes en linje, ud fra punkter i 2d. Det kan fx være en skelgrænse, som er indmålt.

2d transformation (2 translationer, 1 rotation, 1 skala, eller kombinationer heraf

I denne opgave er der indmålt en række punkter i 2d, som er defineret i to forskellige koordinatsystemer. Det er målet for de robuste udjævningsmetoder at finde frem til transformationsparametrene, mellem de to koordinatsystemer.

Approksimation af et plan i 3d ud fra lasercanningsdata

Denne opgave går ud på at der ud fra en masse punkter i 3d skal laves et plan i 3d. I sådanne opgaver er der oftest mange data og mange grove fejl. Ikke nødvendigvis fordi der lavet fejl i opmåling, men på

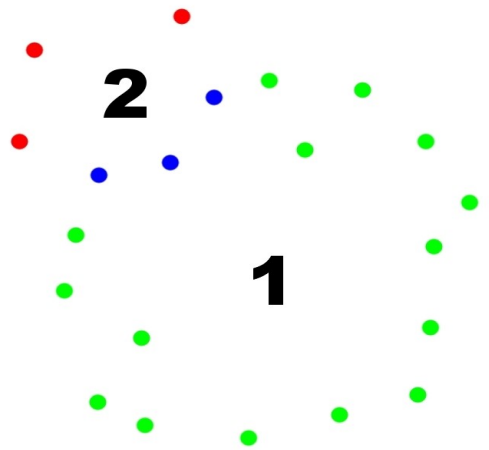


Illustration 10:Figuren viser en polygonudjævning, hvor første opstillings punkter er angivet med grøn, anden opstilling er angivet med rød, mens fællespunkterne er blå.

grund af opgavens natur. Hvis det plan der søges er en væg, vil der være punkter, som ikke ligger på væggen. Dette kan være en opslagstavle, et vindue eller en dør. Der kan være objekter i datasættet, som ikke beskriver det der søges. Derfor skal der en robust udjævningsmetode til, for at finde frem til det rigtige plan.

Polygonudjævning

Denne opgave går ud på at sammenknytte flere forskellige opstillinger, etableret med en totalstation. Et problem med denne opgave er at det kun er få punkter, som binder de to opstillinger sammen. En algoritme som RANSAC, ville ikke kunne håndtere denne opgave. Dette ligger i at RANSAC ikke kan udtrække et tilfældigt antal observationer og løse problemet på baggrund af dette. Problemet er for kompliceret til RANSAC. Dette problem er bedst løst med en IRLS algoritme, men dette er heller ikke problemfrit.

I figuren (Illustration 10) er punkter fra den ene opstilling angive med grøn, mens den anden opstillings punkter er angive med rød. Fællespunkterne er blå. I denne situation kan RANSAC algoritmen ikke udtrække tilfældigt og udregne koordinater til punkterne. For at løse det er RANSAC nødt til at vælge blandt de blå punkter og det der ellers skal bruges. For IRLS algoritmer er problemet af en anden karakter. Den anden opstilling (angivet med rød) har færre overbestemmelser end første opstilling (angivet med grønt). Dette medfører at anden opstilling kan medføre større residualer end første. Det betyder at der ikke kan sættes nogen global tærskel værdi for IRLS i denne situation. Udjævningen kan tåle at der bliver udvægtet flere observationer fra første opstilling end fra anden opstilling, derfor skal de enkelte opstillinger have deres egen tærskelværdi, baseret på hvor mange punkter der indgår i opstillingen.

Afstand

Denne opgave går ud på at en afstand er målt flere gange og der skal dannes et middeltal. Samme form for udjævning bruges når der skal tages et middeltal af koordinater fra en GPS. Opgaven kunne lige såvel hedde udjævning af middeltal, men kaldes udjævning af afstand i projektet.

Opgaver i projektet

Som nævnt er der for mange opgaver til at projektet kan behandle dem alle. Derfor vælges nogle fra. Umiddelbart vælges polygonudjævningen fra, da det ikke umiddelbart indenfor projektets tids horisont

er muligt at lave en RANSAC algoritme som kan håndtere dette problem, hvis dette er muligt. Samtidig er der ikke muligt i projektets tidshorisont at få IRLS til at løse problemet fornuftigt, med henvisning til de forskellige tærskelværdier. Oven i dette fravælges det at arbejde videre med udjævning af et plan i 3d. Dette er primært på grund af mængden af data, som laserscanneren vil genererer. Her vil beregningstiden sandsynligvis være meget stor og besværliggøre projektet unødigt, i forhold til hvilket udbytte der vil være, i dette eksperiment. Tilbage er der approksimering af 2d linje, 2d transformation og udjævning på en afstand. Disse tre opgaver arbejdes der videre med.

3.3 Udjævningsmetoder

I foranalysen er der præsenteret et ikke-udtømmende udvalg af robuste udjævningsmetoder. Derfor kan det være relevant at vælge nogle af udjævningsmetoderne fra. Umiddlebart vælges r- og s-estimatorer fra, da der ikke er blevet undervist i dem og undertegnede ikke kender til dem udover deres navn. Derfor undersøges de følgende udjævningsmetoder:

- Mindste kvadraters princip + data snooping
- IRLS med Huber
- RANSAC
- RANSAC + IRLS + MKP

Udover at metoderne testes hver for sig, laves der også et forsøg med at kombinere algoritmerne. Argumentet for denne kombination, er at alle algoritmerne har deres styrker og svagheder, men ved at kombinere algoritmerne korrekt, kan det være at deres individuelle svagheder imødegås. Det vil sige at deres svagheder dækkes af en af de andre algoritmers styrker.

4 Teori

I dette afsnit bliver de udvalgte udjævningsmetoder beskrevet i detaljer. Der bliver lagt et fokus på beskrivelser af IRLS og RANSAC. Hensigten er at beskrive de overvejelser der er gjort i forbindelse med implementeringen.

4.1 Beskrivelser af de udvalgte udjævningsmetoder

Herunder følger en beskrivelse af de overvejelser der følger med de enkelte metoder. Som nævnt er IRLS en af de metoder der arbejdes med. Her er det relevant at beskrive hvilken robust funktion der indgår i IRLS, om det er en L_1-estimator eller en m-estimator, og i så fald hvilken m-estimator. Der lægges i dette afsnit fokus på RANSAC og IRLS, da disse er særligt vigtige for projektet.

4.2 IRLS

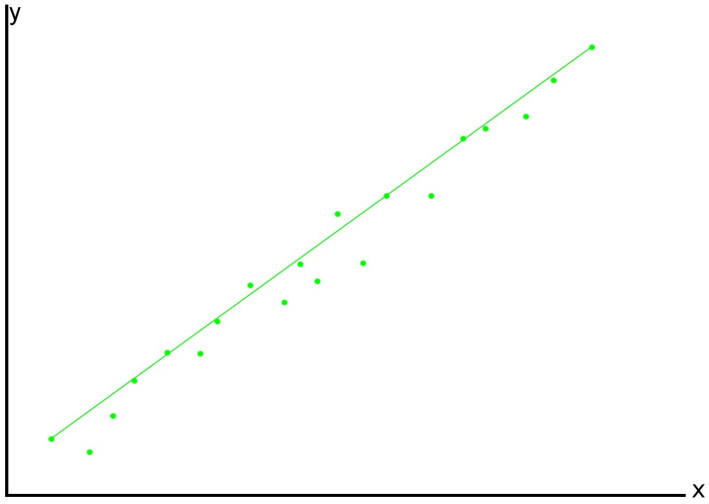
I forbindelse med arbejde med IRLS er der to vigtige overvejelser at gøre sig. Hvilken robust estimator

skal der bruges, og hvad skal stop kriteriet være. Det kunne i sig selv være interessant at sammeligne flere forskellige IRLS algoritmer, med forskellige robuste estimatorer indbygget, men i forbindelse med dette projekt arbejdes der kun med én IRLS algoritme. Der vælges at arbejde videre med Huber, da det viser sig at denne måde at vægte på løser nogle af de problemer der umiddelbart er ved at anvende L_1-normen.

Den næste overvejelse er hvad stop kriteriet skal være. Som nævnt skal der være en sammenhæng mellem opgavens natur/krav og stop kriteriet. Som det er foreslået i foranalysen, kunne et stop kriterie være når to på hinanden følgende løsninger har en forskel, som er mindre end en given tærskel værdi. estimatet fra iteration i+1 trækkes fra iteration i og det afgøres om resultatet ligger under en fastsat tærskel værdi. Denne form for stop kriterie arbejdes der videre med i dette projekt. Her næst skal det overvejes om der kan sættes et stop kriterie for alle opgaver, eller om disse skal sættes individuelt. Ved fastsættelsen af dette kriterie er der to fælder, der skal undgås. Den ene er at tærskelværdien sættes for højt. Faren heri er at algoritmen stopper for tidligt og der kunne måske være opnået et bedre resultat. En anden fare er at sætte tærskelværdien for lavt. Her kan der muligvis opstå en situation, hvor dataene ikke har den 'opløsning' som kræves af stopkriteriet. Fx ved afstandsmåling, kunne man forestille sig situationen således: Afstandsmåleren har en spredning på ca. 10 cm, skal stopkriteriet så sættes til 0.1 mm? I denne situation er det ikke sikkert at dataene har den detaljeringsgrad som skal til.

4.3 RANSAC

RANSAC (RANdom SAmple and Consensus) er en robust metode til bestemmelse af parametre i en given model. Det er en metode som undertegnede/gruppen har stiftet bekendtskab med i forbindelse med et sommerferiejob hos Scankort. Metoden bruges som regel til orientere et billedeplan i et 3d rum. Gruppen/undertegnede finder metoden interessant som koncept. Gruppen er ikke bekendt med at der er blevet undervist i denne metode. Konceptet går ud på at man har et datasæt, som skal tilpasses en matematisk model. RANSAC gennemløber en masse iterationer hvor den tager tilfældige datapunkter ud og opstiller modellen efter det. Derefter vil RANSAC måle "afstanden" fra den opstillede model til de resterende datapunkter. På denne måde "måles" hvor god modellen er.



I eksemplet på billedet vil RANSAC måle afstanden fra punkterne og ind til linjen. Efter første iteration gemmes modellen og de punkter den er dannet af og en ny iteration startes. Hvis den nye iteration giver et bedre resultat ie. "afstanden" mellem modellen og det resterende datasæt er mindre, gemmes den nye løsning og den gamle forkastes. På denne måde fortsætter RANSAC indtil den har opnået et givent successkriterie, eller fundet det bedste

resultat. Net resultatet er, at i fald at beregningerne er gået godt, vil man have fået model, som er fri for grove fejl. En stor fordel ved RANSAC er at den ulig Mindste kvadraters metode, kan håndtere en meget stor mængde grovefejl > 50%[9].

Der er nogle problemstillinger, som der skal tages højde for i forbindelse med anvendelse af RANSAC algoritmer. Disse er stopkriteriet og beregningstiden. RANSAC er en iterativ process, den adskiller sig dog, fra andre iterative metoder, ved at RANSAC skal igennem ALLE observationer evt. flere gange. Dette udspringer af at RANSAC for hver iteration udtager et tilfældigt antal punkter og regner modellen. Dette resulterer i at RANSAC kan gennemløbe væsentlig flere iterationer end IRLS. Ved store datasæt kan beregningstiden være et problem der skal tages i forbehold. Et væsentlig større problem med implementeringen af RANSAC er at få etableret nogle fornuftige stop kriterier.

I det følgende er der en let forståelig, konceptuel gennemgang af RANSAC i pseudo-kode, farverne er til for at gøre det nemmere at referere til pseudo-koden, når den gennemgås:

```
input:
   data - a set of observations
   model - a model that can be fitted to data
   n - the minimum number of data required to fit the model
   k - the maximum number of iterations allowed in the algorithm
   t - a threshold value for determining when a datum fits a model
   d - the number of close data values required to assert that a model fits well to
data
output:
   best_model - model parameters which best fit the data (or nil if no good model
is found)
   best_consensus_set - data point from which this model has been estimated
   best_error - the error of this model relative to the data

iterations := 0
best_model := nil
best_consensus_set := nil
best_error := infinity
while iterations < k
   maybe_inliers := n randomly selected values from data
   maybe_model := model parameters fitted to maybe_inliers
   consensus_set := maybe_inliers

   for every point in data not in maybe_inliers
       if point fits maybe_model with an error smaller than t
           add point to consensus_set

   if the number of elements in consensus_set is > d
       (this implies that we may have found a good model,
       now test how good it is)
       better_model := model parameters fitted to all points in consensus_set
       this_error := a measure of how well better_model fits these points
```

9[M. Zuliani 2009 "RANSAC4Dummies.pdf" s. 14]

```
        if this_error < best_error
            (we have found a model which is better than any of the previous ones,
           keep it until a better one is found)
            best_model := better_model
            best_consensus_set := consensus_set
            best_error := this_error

    increment iterations

return best_model, best_consensus_set, best_error
```

Eksemplet ovenover er taget fra wikipedia.org. Dette eksempel er valgt da det giver en god introduktion til RANSAC og forklarer hvad RANSAC går ud på. Den fulde artikel på wikipedia.org kan ses her:

http://en.wikipedia.org/wiki/Ransac

Ud fra faglig viden og vurdering af referencerne, vurderes artiklen at være af tilstrækkelig høj faglig kvalitet og derved egnet til universitetsbrug.

Her følger en gennemgang af RANSAC pseudo-koden ud fra farve koderne. Den første sorte tekst i pseudo-koden er input og output. I projekt kontekst er 'data' mængden af observationer der skal udjævnes på. 'model' dækker over den opgave der skal løses. 'Model' kan derfor være en 2d transformation, eller linjens ligning. 'n' er det antal punkter der minimum skal til for at opstille modellen. Dette er nødvendigt at vide, da RANSAC i hver iteration udtrækker n punkter fra 'data', for at opstille 'model'. 'k' er det maksimale antal iterationer RANSAC skal igennem. Denne værdi kan beregnes på flere måder. Dette beskrives nærmere under 'stopkriterie' i afsnittet 'RANSAC overvejelser'. 't' er den værdi der angiver hvor 'tæt' en observation skal ligge på modellen, for at blive vurderet til at være fri for grove fejl. Når RANSAC har fundet en model testes denne på den resterende data, dvs al den data som ikke indgik i opstilling af modellen. Residualet er en oplagt værdi at sammenligne på. 'd' er det antal punkter der skal til for at vurdere om den fundne løsning er god nok. Hvis en model er opstillet på 3 punkter og passer på 1 punkt ud af 20 resterende punkter, kan man måske finde en løsning som dækker bedre. 'best_model' er i projekt sammenhæng den bedste model som RANSAC har fundet, den der giver de mindste residualer og opfylde kravet fra 'd'. 'best_consensus_set' er mængden af de punkter som 'best_model' passer til indenfor 't'. 'best_error' er størrelsen af den fejl, som 'best_model' genererer.

Rød tekst

Denne del er opstarten af RANSAC. Så længe 'iteration' er mindre end 'k' kører RANSAC. 'best_model' og 'best_consensus_set' sættest til 0 i første iteration og 'best_error' sættes til uendelig. Dette tvinger RANSAC til at gennemføre flere iterationer. Dette medfører at, skulle RANSAC, tilfældigvis i første iteration finde en løsning med en 'best_error' = 0 og 'consensus_set' > d, så arbejder den videre for at se om en bedre løsning skulle være til stede. Herefter opstilles modelparametrene. De n punkter lægges over i maybe_inliers, som også danner grundlag for consensus_set.

Blå tekst

Her testes om resten af punkterne passer på den opstillede model. Hvis de gør det, med en fejl mindre eller lig med t, lagres de i consensus_set. Hvis antallet af punkter i consensus_set er større end d (dette indikerer at der muligvis er fundet en god model), opstilles der et fejludtryk for hele datasættet.

Grøn tekst

Her testes om den nuværende fejl er mindre end den forrige. Hvis dette er tilfældet opdateres variablene. Det vil sige best_model = better_model, best_consensus_set = consensus_set og best_error = this_error. Dette betyder at det kun er når en bedre model er fundet, at variablene opdateres.

Mørk lilla

Her opdateres antallet af iterationer, så RANSAC kan kører igen, eller stoppe, hvis stopkriteriet er nået.

Herefter retuneres output.

5 Data udvælgelse

I dette afsnit beskrives hvilke overvejelser der ligger til grund for de datasæt der udjævnes på, i de eksperimenter der foretages. Datasættene skal udsætte de robuste udjævningsmetoder for forskellige situationer. Her er det værd at overveje hvad der, i projekt kontekst, udgør forskellige situationer. Hvis der fx udjævnes på en ret linje, er det så to forskellige situationer hvis den ene linje hedder $y = 2x + 3$ og den anden hedder $y = 0.5x - 4$, begge situationer indeholder 10 punkter og 2 grove fejl? I denne situation udjævnes der på to forskellige linjer, men da det er robuste udjævningsmetoder der undersøges, er det antallet og størrelsesordnen af grove fejl der er interessant for projektet. Det vil sige at situationerne i førnævnte eksempel er den samme. To forskellige situationer, for dette projekt, er $y = 2x + 3$ med fx 5 punkter og 2 grove fejl og $y = 2x + 3$ med fx 20 punkter og 3 grove fejl. I disse to situationer udjævnes der på den samme linje, men i den ene situation er der 5 punkter at udjævne på, og 2 grove fejl, mens der i den anden situation er 20 punkter at udjævne over og 3 grove fejl. Pointen er her at formlen for den rette linje er ligegyldig. Det som er interessant for projektet er antallet af punkter der skal beskrive linjen, samt antallet af grove fejl, herunder også størrelsesordnen af de grove fejl. Derfor skal testsene justere på antallet af observationer, samt antallet og størrelses ordnen af de grove fejl. Dette er en af de grundlæggende overvejelser der ligger til grund for alle datasæt til eksperimenterne.

Herunder kommer der overvejelser der er specifikke for de enkelte undersøgelser.

5.1 Data til udjævning af afstand

Analogt til de generelle overvejelser er det ligegyldigt hvad 'afstanden' er i disse eksperimenter. I betragtning af at der udjævnes på et tal, den størrelse der søges, kan dette tal fx være 0. Hvad tallet er har ingen betydning for udjævningen, men for forståelsens skyld vælges der et tal > 0, fx 50. Med dette menes at eksperimenterne skal efterligne en situation hvor der er målt en afstand på 50 m. Dette er gjort for at lette den menneskelige forståelse af situationen. Scripts til generering af afstandsdata kan ses i bilag [Bilag 1]

5.2 Data til udjævning af ret linje i 2d

I forbindelse med datasættene der beskriver linjer, er der andre forhold der skal tages i betragtning. En af de vigtigste overvejelser er om der er fejl på x-koordinaterne, i systemet. Grunden til dette er, at det er med til at bestemme hvilken form for mindste kvadraters metode der skal anvendes. Den almindelige mindste kvadraters metode, kan ikke håndtere fejl på x-aksen. Dette kommer af at x-koordinaterne ikke opfattes som observationer, i ligningssystemet. Hvis der er fejl på x-aksen skal den generelle mindste kvadraters metode anvendes. I forbindelse med projektet, har det ingen betydning om der anvendes almindelig mindste kvadraters metode, eller den generelle. Det forventes ikke at påvirke resultatet af projektet, som er at finde ud af, om én af de udvalgte robuste udjævningsmetoder er bedre end de andre. Af denne grund vælges der at arbejde videre med almindelig mindste kvadraters metode. Dette udelukker derfor at der kan forekomme fejl på x-aksen. Alternativ kan eksperimentet beskrives, som et system, der gennem definition udelukker fejl på x-aksen. Fx at der måles med bestemte intervaller, på x-aksen, hvormed målestøjen forekommer på y-aksen.

Oven i dette skal punkternes fordeling over x-aksen, overvejes. Det vil sige om, der er 'klumper' af observationer, eller om de ligger med konstante intervaller på x-aksen. I betragtning af at en af fortolkningerne af eksperimentet, er et forsøg hvor der måles med bestemte intervaller, forekommer det mest logisk at, observationerne er fordelt med konstante intervaller over x-aksen.

Det er blevet nævnt at linjens orientering ikke er vigtig for projektet. Dette er også rigtigt, dog er der én situation der skal undgås: Linjen må ikke ligge lodret, dvs. parallel med y-aksen. Det vurderes dog ikke til at være noget problem at undgå dette. Den linje der udjævnes på er defineret af formlen $y = 1x + 3$, scripts til generering af linjedata kan ses i bilag [Bilag 2].

5.3 Data til udjævning af 2d transformation

I dette eksperiment er der nogle overvejelser, der minder om dem fra linje-eksperimentet. Om punkterne skal ligge mere eller mindre tilfældigt, eller om de skal konstrueres, med henblik på at opnå en bestemt geometri. En fordel ved konstruerede punkter, er at det er muligt at styre geometrien, i de punkter som der transformeres over. Det er herved også muligt at påvirke geometrien bevidst, når de grove fejl introduceres. I kontrast til dette kan, der etableres en mængde tilfældige punkter, med en tilfældig geometri. Herved kan man ikke bevidst, eller i samme grad, påvirke geometrien, når de grove fejl introduceres.

Umiddelbart er det ikke ønskeligt at undersøge, hvordan udjævningsmetoderne håndterer forskellige former for geometri. Men dette ønske er ikke i konflikt med valget af konstruerede data, kontra tilfældige data. Dette forhold skyldes at det netop er antallet af punkter, som transformationen foregår over, samt antallet af grove fejl, som er interessant, ikke geometrien. Et andet ønske kunne være at eksperimentet, skal ligne en virkelig situation. Men heller ikke dette ønske synes at kunne afgøre sagen. Når der foretages transformation, over en mængde punkter, tænkes der over geometrien i nettet, men det står ikke altid landinspektøren frit for, at etablere punkter hvor end det er ønskeligt. Dertil kommer at man kan befinde sig i en situation hvor der skal foretages en transformation, på baggrund af et udvalg af tilgængelige fællespunkter. Med dette menes, at det ikke er sikkert at de punkter der transformeres over, var designet til at der skulle foretages en transformation over dem. Det kan være at nogle af punkterne er detailpunkter, fra en teknisk opmåling af en baggård. I denne situation, kan geometrien være mere eller mindre tilfældigt eller uhensigstsmæssig, da der transformeres over tilgængelige punkter, frem for ønskelige punkter.

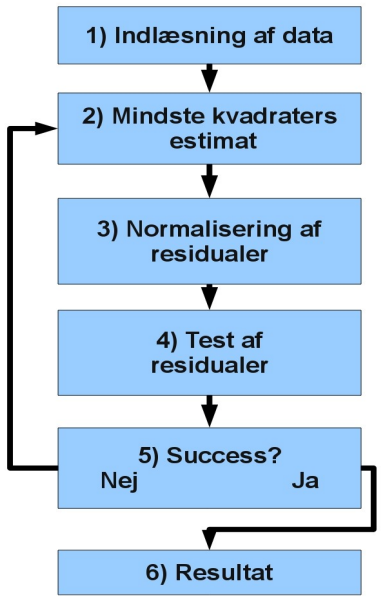
Da det ikke umiddelbart virker muligt at producere et godt fagligt argument, for konstruerede punkter, eller tilfældige punkter, foretages valget i stedet for, på grundlag af hvad der virker nemmest. Umiddelbart findes det nemmest at generer en mængde tilfældige punkter, hvilket betyder at geometrien i punkskyen er mere eller mindre tilfældig. Scripts til generering af data kan ses i bilag [Bilag 3]

6 Implementering

Dette afsnit beskriver i detaljer, hvordan de forskellige algoritmer og metoder til robust udjævning, er omsat til funktionelle programmer.

6.1 Mindste Kvadraters Metode + data snooping

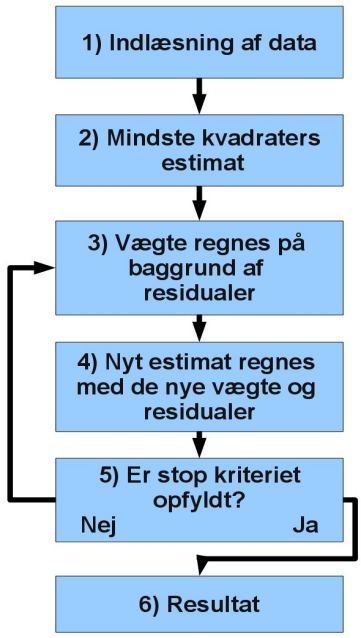
Mindste kvadraters metode og data snooping er implementeret i matlab efter følgende figur:



1. Først indlæses der data . Det vil sige mængden af observationer indlæses som et data array i matlab, som der kan arbejdes med.
2. Hernæst opstilles der et alm. mindste kvadraters estimat På baggrund af dette estimat opstilles der en residual vektor, som indeholder residualer til alle observationerne.
3. Herefter normaliseres residualerne, med henblik på at teste dem.
4. I testen , gennemgås residualerne hver for sig, for at se om de ligger over en given tærskel værdi. Denne tærskel værdi sættes som regel ved 3. Dette er i overensstemmelse med forståelsen for at en grov fejl ligger uden for 3 gange spredningen. I dette projekt er denne tærskel værdi sat til 2.8.
5. Så fremt der findes et residual der er større end tærskel værdien, fjernes den tilsvarende observation, og der laves et nyt estimat 2). Herefter kører løkken igen, indtil der ikke er flere residualer der overstiger tærskelværdien 2.8 og resultatet returneres. MKP og data snooping kan ses i matlab kode i bilag [Bilag 4].

6.2 IRLS

I projektet er IRLS implementeret på følgende måde:

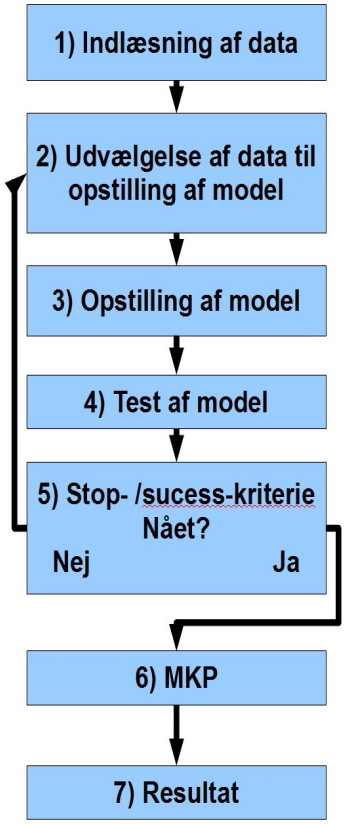


1. Der indlæses data, i et array, så der kan arbejdes med det.
2. Der laves et mindste kvadraters estimat. På baggrund af dette opstilles en residual vektor, som danner grundlag for vægtmatricen i næste beregning.
3. På baggrund af de nye vægte, regnes et nyt estimat.
4. På baggrund af dette estimat regnes der nye residualer.
5. Så fremt at stop kriteriet endnu ikke er nået, beregnes nogle nye vægte, på baggrund af estimatet, og løkken kører igen.
6. Når stop kriteriet opnås, returneres resultatet.

I den IRLS algoritme der er lavet i dette projekt, regnes vægtene på baggrund af Huber-vægtfunktionen. Stopkriteriet i de IRLS algoritmer der er anvendt, i dette projekt, er det samme, for alle algoritmer. Stopkriteriet er fastsat, som en ændring af løsningsvektoren. Når ændringen, af løsningsvektoren, imellem to iterationer, falder under en given tærskelværdi afbryder algoritmen og resultatet returneres. Denne tærskelværdi er i projektet sat til 0.0004, det vil sige at når løsningsvektoren, mellem to iterationer, ændrer sig mindre end denne værdi, afsluttes IRLS. Et sådant stopkriterie kan medføre den fare, at i fald løsningen ikke konvergerer, vil IRLS lave en uendelig løkke. Dette problem kan løses ved at sætte et max antal iterationer, for algoritmen. I projektet er der sat et max på 1000 iterationer. Scriptet gør desuden brug af Peter Cederholm's script 'robvegt.m' til at opstille vægtmatricerne. Implementeringen af IRLS kan ses i bilag [Bilag 5].

6.3 RANSAC

Den måde RANSAC er implementeret på, i projektet adskiller sig på nogle punkter fra den pseudokode der er præsenteret i teori kapitlet. Denne forskel ligger i hvordan der udtrækkes data til opstilling af modellen.



1. Først indlæses dataen.
2. Herefter udvælges der systematisk data, til opstilling af modellen. Dataen bliver udvalgt så alle kombinationer af observationer afprøves. Dette betyder at RANSAC algoritmen tester alle observationer.
3. Når de rigtige observationer er udtrukket, opstilles modellen på baggrund af de udtrukkede data.
4. Herefter testes modellen på de resterende data. Både modellen og resultatet af testen gemmes og anden iteration startes. I slutningen af hver iteration testes modellerne mod hinanden. Den model, som medfører den mindste residual vektor, gemmes og en ny iteration påbegyndes. Skulle det ske at der findes en bedre løsning, gemmes denne og den forrige forkastes. På denne måde sikrer man at det hele tiden er den bedste løsning der gemmes.
5. Når RANSAC når sit stopkriterie afbrydes algoritmen.
6. På dette tidspunkt har RANSAC opbygget et konsensus sæt, som der lavet et alm. mindste kvadraters estimat på.

Ad 2. Den måde som dataene udtrækkes på, er i overensstemmelse med binomialkoefficienten. Dette betyder at dataen bliver udtrukket, så samtlige forskellige kombinationer afprøves.

I forbindelse med udjævning af data fra afstandsmåling, er der et special tilfælde i forhold til RANSAC. Dette går ud på at, da RANSAC gennemprøver samtlige kombinationer, vil det i realiteten svare til en median. Den median der menes her, er at alle observationer ordnes i rækkefølge, fra den mindste til den største, og den observation der ligger i midten, vælges. Dette er den observation, som genererer den mindste residual vektor i forhold til de andre observationer. På baggrund af dette, er RANSAC i tilfældet hvor der udjævnes på afstande, reduceret til en median.

RANSAC bruger en tærskelværdi til at vurdere om en observation er behæftet med en grov fejl. Da spredningerne for dataene er sat til 4 mm, eller 0.004 enheder, er RANSACs tærskelværdi sat til

15 mm eller 0.015 enheder. Denne værdi er valgt da den ligger lige over 3 gange spredningen. Grunden

til at den er lidt større end 3 gange spredningen, er at RANSAC ikke udjævner, men finder den bedste løsning i datasættet. I forbindelse med afstandsmålingen, kan det illustreres ved at RANSAC finder medianen. Det betyder at løsningen er flyttet i forhold til middeltallet. For at løse dette problem, fastsættes RANSACs tærskelværdi til at være lidt større end 3 gange spredningen.

Som det er nævnt i foranalysen, opbygger RANSAC et konsensussæt. I implementeringen af RANSAC, er der lagt endnu et success kriterie på, i forhold til hvornår RANSAC opdaterer modellen. Dette kriterie er størrelsen af konsensussættet. Dette betyder at det ikke længere er nok at den nye model medfører en mindre fejl, men den skal også have et konsensussæt, som er ligeså stort, eller større end det forrige.

De matematiske modeller i RANSAC bliver opstillet ved hjælp af mindste kvadraters princip. For hver iteration bliver der indlæst lige det nødvendige antal datapunkter, som er nødvendigt for at opstille modellen. Der er herved ikke tale om udjævning efter mindste kvadraters princip, men matematikken bliver anvendt til at opstille modellen. I nogle af opgaverne vil det være bedre at anvende anden matematik i opstillingen af modellen, end den fra MKP, da programmerne vil køre hurtigere. Men da det tager få sekunder at skrive formlerne fra MKP, vælges disse. Det skal hertil også overvejes hvor meget tid der ville være gået med at sætte sig ind i denne matematik.

Afstandsfunktionen er en funktion, som beskriver hvor god en kongruens der er mellem modellen og det resterende datasæt. Afstandsfunktionen tester på alle de punkter der ikke indgår i modellen. For dette projekt kan afstandsfunktionen, sammenlignes med residualerne. Residualerne beskriver den fejl der begås ved at benytte den fundne model. Dette gør at residualerne er en god måde at beskrive kongruensen på, mellem den fundne model og det resterende datasæt. Et væsentligt spørgsmål er om RANSAC skal måle på de absolutte residualer eller kvadrerede residualer. Det giver ikke nødvendigvis det samme resultat. Det er i projektet valgt at RANSAC vurderer løsningen på de absolutte residualer.

Stop kriteriet er måske det element i RANSAC, som er mest besværlig at håndtere[10]. Stopkriteriet kan være opstillet på flere forskellige måder.

Dels kan man anvende fejlteori, til at udregne hvor mange iterationer RANSAC skal igennem for at have opstillet en model, der med en vis statistisk sandsynlighed overholder en hvis tærskelværdi. Den statistik der anvendes her bygger på sandsynligheden får at udtrække fællespunkter der er behæftet med grove fejl og den omvendte situation. Udfra dette kan man opstille et udtryk for sandsynligheden for at en model overholder et givent successkriterie. Hvis q er chancen for at vælge et godt udtræk, så er risikoen for at vælge et dårligt udtræk (mindst én grov fejl) $1-q$. Hvis der er opstillet h forskellige modeller er risikoen for at de alle sammen er fejl behæftede $(1-q)^h$. Denne størrelse går mod nul når $h \to \infty$. Her vendes problemet på hovedet, der vælges en statistisk sikkerhed for at beregningerne er gået godt. Det kan fx være 0.95, svarende til 95%. Dette tal kaldes 'alarmrate' og

10[M. Zuliani 2009 "RANSAC4Dummies.pdf" s. 19]

repræsenteres med ε RANSAC terminologi. Herefter er antallet af iterationer som RANSAC skal igennem givet ved:

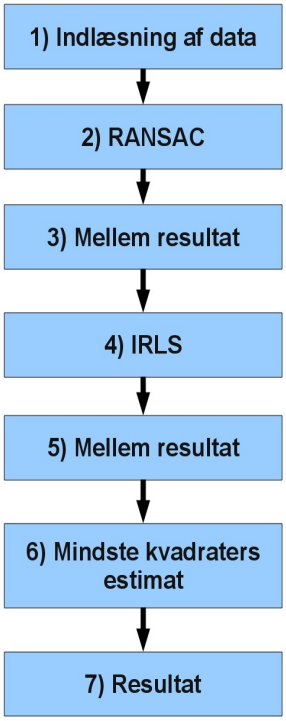
$$\frac{\log(\varepsilon)}{\log(1-q)}$$

En anden måde at lave stopkriteriet på, ligger i at RANSAC algoritmen bliver skrevet så der ikke længere udtrækkes tilfældige data, men dataene til modellen 'håndplukkes'. Tanken er at det skal gøres på en måde så alle mulige forskellige kombinationer af datapunkter afprøves. Det vil sige en binomialkoefficient.

En tredje måde at lave et stopkriterie på er at lade RANSAC gemme de gode løsninger. Når et antal gode løsninger passer med hinanden, indenfor en vis margin, stoppes algoritmen. Det kan fx være at RANSAC arbejder sig systematisk gennem dataen, hvis 3 løsninger passer indenfor 20%, så stopper RANSAC og returnerer den bedste af de 3 løsningner. Implementeringen af RANSAC i matlab kan ses i bilag [Bilag 6].

6.4 RANSAC + IRLS + MKP

Implementeringen af den algoritme der kombinerer RANSAC, IRLS og MKP, adskiller sige ikke meget fra den måde de enkelte udjævningsmetoder, er implementeret på. De er implementeret ved at blive sat i forlængelse af hinanden. Denne metode er kaldet RIL (Ransac Irls Least-squares):



1. Først indlæses der data i et array.
2. Hernæst kører der en RANSAC algoritme, som den er beskrevet ovenover.
3. Mellemresultatet, det vil sige resultatet fra RANSAC sendes videre til
4. IRLS algoritmen. Denne IRLS er den samme som beskrevet tidligere.
5. Resultatet fra denne IRLS udjævning, sendes videre til
6. et mindste kvadraters estimat, som denne er beskrevet tidligere, dog uden data snooping delen.
7. Til sidste returneres resultatet

Efter RANSAC algoritmen, er de største grove fejl sorteret fra. Dette sikrer en godt fundament for IRLS at arbejde videre med. Normalt Starter IRLS med et mindste kvadraters estimat, for at

for opstillet nogle residualer, der kan laves vægte ud fra. I denne algoritme, startes der med de residualer, som den bedste løsning fra RANSAC har medført. Der startes altså ikke længere med et mindste kvadraters estimat. IRLS regner videre på det resultat som RANSAC er noget frem til. Efter at IRLS har kørt er de sidste grove fejl der måtte være i datasættet, sorteret fra. Til at sorterer observationer fra, benyttes vægtmatricen. Hvis en observation har fået en vægt, som er under en given tærskel værdi, vurderes den som værende en grov fejl. Det kan umiddelbart være et problem at opstille denne tærskel værdi, men her kan et plot af vægtfunktionen være til hjælp. Umiddelbart kan der dannes sammenhæng mellem tærskelværdien for data snooping og den tærskelværdi der skal bruges i RIL. På x-aksen i plottet ses de normaliserede residualer. Da data snooping sorterer residualer fra, som er større end 2.8, vurderes det til at være fornuftigt, hvis IRLS i RIL algoritmen, gør det samme.

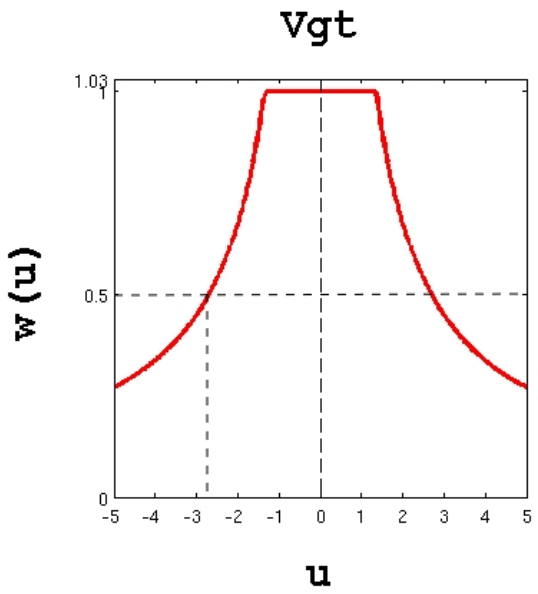


Illustration 11:Plot af Huber vægt, med hjælpelinjer til at fastsætte tærskel værdien, for hvornår en observation sorteres fra.

Det betyder at der nu er banet vej, for almindeligt mindste kvadraters estimat. Igen her gøres der brug af Peter Cederholm's script 'robvegt.m' til at opstille vægtene til IRLS-delen af programmet. Implementeringen af RIL kan ses i bilag [Bilag 7].

7 Forsøg

I afsnittet om problemformuleringen blev det nævnt at det er nødvendigt at etablere et sammenligningsgrundlag. En del af dette er de opgaver der skal løses, men måske endnu vigtigere at have et 'rigtigt' resultat at sammenligne med; en kvalitetsvurdering. Til at kvalitets vurdere udjævningsmetoderne anvendes MKP (Mindste Kvadraters Princip). For hver test laves der et MKP estimat, på data uden grove fejl. Dette estimat opfattes herefter i testen, som værende det rigtige svar. Uden fejl i datasættet skal de robuste metoder ligge meget tæt på dette estimat. Dette er et kriterium for at metoden kan tages i overvejelse. Heraf følger et nyt problem. De robuste metoder forventes ikke at producere et resultat, som er præcis det samme som MKP. Det forudsættes stadig at de arbejder på data uden grove fejl. Hvor langt kan resultatet være fra MKP, for stadig at blive vurderet som værende tæt nok på? Dette er noget der vil blive taget stilling til i de enkelte tests, da det ikke forventes at der kan opstilles et overordnet succeskriterium på dette punkt.

Det næste at finde ud af er hvad der skal sammenlignes. Indtil videre er det klargjort at de robuste udjævningsmetoder skal sammenlignes med MKP. Skal der sammenlignes på residualernes størrelse eller et andet kvalitets udtryk? Umiddelbart virker residualerne som en god størrelse at sammenligne på, ud fra følgende kriterier:

- Residualerne er i samme enhed, for samme opgave.
- Giver et absolut indtryk af kvaliteten.

Med første punkt menes at de residualer der kommer ud af to forskellige robuste udjævningsmetoder, må have samme enhed, under forudsætning af at de to udjævningsmetoder har løst den samme opgave, på baggrund af den samme data. Dette gør at residualerne fra de to udjævningsmetoder, umiddelbart kan sammenlignes.

Med andet punkt menes at man får at absolut indtryk af hvordan de robuste udjævningsmetoder klarer en given opgave. Dette kommer af at der er enheder på residualerne. Dette kan måske give et mere håndgribeligt indtryk udjævningsmetoderne, frem for at få at vide at den ene er 1½ gang bedre end den anden, hvis det nu måtte være tilfældet.

Dog kan det være svært at sammenligne et antal residualvektorer, for at se hvad for en der er bedst. Residualerne gør det nemmere at se størrelsesordnen af den fejl, der begås, ved at anvende de fundne ubekendte i udjævningen, men samtidig ofres overskueligheden, hvilket er vigtigt for dette projekt.

Som alternativ til at bruge residualerne til kvalitetsvurderingen, kan man bruge spredningen på vægtenheden, som nævnt i teori afsnittet. Denne størrelse vil kunne fortælle noget om hvor godt udjævningen er foregået, som er projektets formål. Hertil kommer at denne størrelse er nemmere, at overskue end en residualvektor. På baggrund af dette vurderes det at være bedst at sammenligne metoderne på baggrund af spredningen på vægtenehden, på trods af residualernes gode egenskaber, og

fordi det er nemmere at skabe et overblik ud fra spredningen på vægtenheden.

7.1 Forsøgsparametre

Vurderingen af forsøgsresultaterne sker på baggrund af et sammen hold af σ_0 og antallet af fundne grove fejl. Alle programmerne, undtagen IRLS, benytter sig af den σ_0 der er beskrevet i teori afsnittet. Det er vigtigt at notere sig at denne størrelse regnes efter at evt. fundne grove fejl er fjernet fra datasættet. Det vil sige, i et forsøg hvor der indgår 5 observationer, inklusive én grov fejl, regnes σ_0 på baggrund af 4 observationer, i fald den grove fejl bliver fundet. Dette forhold har en stor betydning for forsøgene, med små antal observationer. Hertil kommer at IRLS ikke sorterer observationer fra, men udvægter dem. Som nævnt i implementerings afsnittet er tærskel værdien for RILs IRLS-algoritme sat til 0.5 da denne værdi skaber kongruens mellem data-snooping og IRLS. På grund af dette vurderes alle vægte i IRLS, som er mindre end 0.5 til at være en formodet grov fejl, som er fjernet.

I hvert forsøg, laves der 3 eksperimenter. I første eksperiment er der 5 observationer og en grov fejl, herefter 10 observationer, 3 grove fejl, og til sidst 20 observationer og 10 grove fejl:

Antal obs.	Antal grove fejl	Procent grove fejl
5	1	20,00%
10	3	30,00%
20	10	50,00%

7.2 Afstand

Herunder udjævnes der på en afstand. Situation er at siddestille med at udjævne på et middeltal. For at lette forståelsen for hvad der sker er afstanden sat til 50 m, hvorefter der er genereret 20 observationer, med en spredning på 4 mm. I første omgang introduceres der ikke grove fejl, da det som nævnt er vigtigt at udjævningsmetoderne producerer et resultat som ligger tæt på MKP, under dette forhold. Herunder præsenteres resultaterne af denne test:

	MKP	MKP+data	IRLS	RANSAC	RIL
$\hat{x}$	49.9995	49.9995	49.9998	49.9995	49.9995
σ_0	0.0045	0.0045	0.0046	0.0045	0.0045

Ud fra denne test, ser det ud til at alle programmerne når frem til de rigtige resultat, når der ikke er nogen fejl. Det kan umiddelbart godt undre at tallene er angivet med 4 decimaler, da spredningen er på 4 mm. Fjerde decimal angiver 1/10 mm. Dog kan det også ses af resultaterne i skemaet, at man er nødt til at angive tallene med min. 4 decimaler, for at se forskellene. Umiddelbart undrer det ikke at IRLS er den der skiller sig ud, da den har vægtet observationerne anderledes, må den komme til at andet

resultat. Da der ikke er nogle grove fejl i dataene, har de andre programmer, foretaget en alm. udjævning, efter mindste kvadraters princip og når derfor samme resultat som MKP.

Her gentages forsøget, dog med én grov fejl i dataene og 5 observationer. En af afstandene er blevet for kort. Her kan man forestille sig at der er kommet et objekt i vejen, for afstandsmålingen, uden at det er opdaget, som følge heraf er én af afstandene blevet 10 m for kort:

	MKP	MKP+data	IRLS	RANSAC	RIL
$\hat{x}$	47.9959	49.994	49.9941	49.994	49.996
σ_0	4.4679	0.0039	0.0021	0.0039	0.0007
Fundne fejl		1	2	1	2

Som det kan ses, er MKP estimatet fejlbehæftet, som forventet, og beskrevet i indledningen. Alle de robuste udjævningsmetoder regner sig frem til et resultat, som ligger næsten indenfor 1 gange spredningen. Det virker dog bemærkelsesværdigt at IRLS og RIL har nogle meget små værdier for spredningen på vægtenheden. Faktisk er værdien, for RIL, som den står i matlab 7.3711e-4, hvilket svarer til 0.0007311. Dette formodes at komme af at de har fra sorteret en observation for meget.

I dette forsøg, er der bygget videre på ovenstående. Der er her 10 observationer og 3 grove fejl. Den ene af de grove fejl, er præcis den samme som ovenstående. Derudover er der kommet 2 afstande ind som er for store, henholdsvis 3 og 12 m.

	MKP	MKP+data	IRLS	RANSAC	RIL
$\hat{x}$	50.4982	49.9975	49.9999	49.9975	49.9979
σ_0	5.2752	0.0063	0.0128	0.0063	0.0077
Fundne fejl		3	3	3	5

Igen her ses det at MKP estimatet er fejlbehæftet. På grund af at der denne gang er 2 afstande der er for store, ser det ud som om at de robuste udjævningsmetoder er kommet tættere på det rigtige resultat, end forrige forsøg, men det er σ_0 som er den interessante størrelse.

I sidste forsøg er der 20 observationer og 10 grove fejl. Der er de samme 3 grove fejl, som i sidste forsøg, samt 7 nye grove fejl. De grove fejl, kan ses i datasættet, som er vedlagt i bilag.

	MKP	MKP+data	IRLS	RANSAC	RIL
$\hat{x}$	50.3395	49.7494	50.0005	50.0008	50.0012
σ_0	4.3940	0.5915	0.081	0.0025	0.0029
Fundne fejl		6	9	10	13

Her begynder algoritmerne at sprede sig i forhold til hinanden, når der kigges på σ_0 og antallet af fundne grove fejl. Det er bemærkelsesværdigt at IRLS kommer så tæt på det rigtige resultat, selvom om

den kun har fundet 9 ud af 10 fejl, samtidig med at σ_0 fra IRLS er bemærkelsesværdigt høj, i forhold til RANSAC og RIL, som også kom tæt på det rigtige resultat.

7.3 Ret linje

I dette forsøg søges der en ret linje ud fra punkt datasæt, i 2d. Først testes alle algoritmerne for at se om de producerer resultater der ligger tæt på MKP. Dette gøres ud fra et sæt på 20 observationer, uden grove fejl. I skemaet herunder angives de parametre som algoritmerne finder frem til. Disse følger linjens formel: $y = ax + b$. I rækken 'ax' angives hvilken hældnings koefficient som der findes, mens rækken 'b' angiver hvilken konstant der er fundet. Linjen der søges har formlen $y = 1x + 3$, med en spredning på 0.004 enheder, i y-aksens retning.

	MKP	MKP+data	IRLS	RANSAC	RIL
ax	1.0002	1.0002	1.0002	1.0002	1.0002
b	3.0006	3.0006	3.0007	3.0006	3.0006
σ_0	0.006	0.006	0.005	0.006	0.006

Umiddelbart er der ikke nogen af algoritmerne som regner forkert. Kun IRLS kommer frem til at resultat, som adskiller sig fra MKP

I dette forsøg er der 5 observationer med én grov fejl i blandt. y-koordinaten på sidste observation er blevet 7 enheder for lille.

	MKP	MKP+data	IRLS	RANSAC	RIL
ax	0.6605	1.0002	0.6605	1.0002	1.0002
b	4.8677	3.0024	4.8677	3.0024	3.0024
σ_0	2.8623	0.0052	2.5313	0.0052	0.0052
Fundne fejl		1	0	1	1

I dette forsøg klarer alle de robuste algoritmer sig godt, pånær IRLS. Problemet er skitseret i afsnit 2.3 IRLS, i foranalysen. På grund af et dårligt udgangspunkt, får IRLS ikke løsningen til at konvergere. Umiddelbart er det paradoksalt at Data Snooping for løst problemet, mens IRLS ikke gør.

I dette forsøg er der 10 observationer og 3 grove fejl.

	MKP	MKP+data	IRLS	RANSAC	RIL
ax	0.7336	0.9002	0.8	1.0005	1.0005
b	5.0651	4.0101	4.6447	2.9939	2.9939
σ_0	2.0879	1.222	1.788	0.0041	0.0041
Fundne fejl		1	1	3	3

I dette forsøg er det kun RANSAC og RIL der kommer frem til det rigtige resultat.

I sidste forsøg er der 20 observationer og 10 grove fejl.

	MKP	MKP+data	IRLS	RANSAC	RIL
ax	0.9636	0.9448	0.969	1.0003	1.0003
b	2.6898	3.25	3.0901	2.9985	2.9985
σ_0	3.017	0.8121	0.5852	0.0064	0.0064
Fundne fejl		2	3	10	10

Ligesom i foregående forsøg er det kun RANSAC og RIL der finder frem til de rigtige resultater.

7.4 Helmert

I disse forsøg laves der robust udjævning på en Helmert transformation, der involverer 1 skala, 1 rotation og 2 flytninger. De transformationsparametre der søges er:

Skala	Rotation (grader)	Flytning x	Flytning y
0.9	15	2	4

I første omgang testes der på 20 punkter uden grove fejl, for at se om de robuste udjævningsmetoder finder et resultat der ligger tæt på MKP estimatet.

	MKP	MKP+data	IRLS	RANSAC	RIL
Skala	0.8999	0.8999	0.8999	0.8999	0.8999
Rotation	15.005	15.005	15.005	15.0048	15.005
Flytning x	2.006	2.006	2.0059	2.0060	2.006
Flytning y	4.0033	4.0033	4.0032	4.0033	4.0033
σ_0	0.0042	0.0042	0.0040	0.0042	0.0042

Umiddelbart ser det ud til at alle algoritmerne finder frem til de rigtige parametre.

Her gentages forsøget, med 5 punkter og 1 fejl. I punkt nr 1 er x-koordinaten i til-systemet blevet flyttet 1 m.

	MKP	MKP+data	IRLS	RANSAC	RIL
Skala	0.8997	0.8997	0.8999	0.8999	0.8999
Rotation	14.2067	14.2067	14.9949	14.9959	14.9959
Flytning x	1.2965	1.2965	1.9936	1.9945	1.9945
Flytning y	4.2996	4.2996	4.0079	4.0052	4.0052
σ_0	0.2734	0.2734	0.0010	0.0028	0.0028
Fundne fejl		0	0	1	1

Alle de robuste algoritmer, pånær data snooping, finder frem til en god løsning. IRLS-algoritmen, formår ikke at finde fejlen, men finder alligevel en rimelig god løsning.

I det næste gentages forsøget, denne gang med 10 observationer og 3 grove fejl. X-koordinaten på første punkt er stadig flyttet med 1 m, ligesom i foregående forsøg. Hertil kommer at der er byttet om på punkt 2 og 3 i Til-systemet. Forstået på den måde at punkt 2 i fra-systemet er matchet til punkt 3 i til-systemet og omvendt.

	MKP	MKP+data	IRLS	RANSAC	RIL
Skala	0.7215	0.9000	0.8996	0.9000	0.8999
Rotation	14.6545	15.0165	15.0218	15.0165	15.0077
Flytning x	5.2843	2.0074	2.0166	2.0074	2.0056
Flytning y	9.7023	3.9977	4.0072	3.9977	4.0039
σ_0	4.4003	0.0041	0.0060	0.0041	0.0026
Fundne fejl		3	3	3	4

I sidste forsøg er der 20 observationer og 10 grove fejl. Der indgår de samme tre grove fejl, som i foregående forsøg. Hertil er der byttet rundt på nogle flere koordinater og andre har fået flyttet lidt på deres koordinater.

	MKP	MKP+Data	IRLS	RANSAC	RIL
Skala	0.8097	0.8997	0.8998	0.8997	0.8994
Rotation	12.3879	14.9974	14.9892	14.9953	14.9896
Flytning x	2.8079	2.0039	2.0000	2.0053	2.0049
Flytning y	8.0929	4.0098	4.0098	4.0114	4.0224
σ_0	3.8357	0.0046	0.0078	0.0062	0.0019
Fundne fejl		10	9	9	13

8 Resultater

I dette afsnit beskrives resultaterne mere dybtgående. Resultaterne bliver udsat for forskellige analyser, for at se om der skulle være tendenser eller mønstre. Derudover vil evt. bemærkelsesværdige resultater blive diskuteret.

8.1 Succesrate

I dette afsnit beskrives de enkelte algoritmers succesrate. Denne succesrate er defineret ud fra om algoritmer har fundet og fjernet fejlene, alle fejlene og ikke mere. Samt om deres resultat ligger tæt på det rigtige svar og om σ_0 er tilstrækkelig lille. Denne vurdering medfører to problemer: Hvor tæt på det rigtige svar skal algoritmens resultat være? Og hvor lille skal σ_0 være. I forbindelse med første spørgsmål skal der fastsættes to tærskelværdier for vurderingen. En tærskelværdi for lineære værdier, og en for ikke-lineære værdier. Med lineære værdier menes der konstanter. Det kan fx være flytningerne i helmert transformationen, eller konstanten i linjens ligning. Alle resultater for afstandsforsøgene vurderes ud fra denne lineære tærskelværdi. Denne tærskelværdi sættes til 2 gange spredningen, det vil sige at da der arbejdes med en spredning på 0.004 enheder i samtlige forsøg, sættes denne tærskel værdi til 0.008 enheder, grunden til at den sættes til dette er at da der formodes at være ligeså mange observationer over middel værdien, som under middel værdien (normal fordeling), burde 1 gange spredningen være en rimelig tærskelværdi, men for at være sikker, sættes den til 2 gange spredningen.

De ikke lineære tærskelværdier er fx hældnings koefficienten i linjens ligning, rotationen og skalaen i helmert transformation. Denne tærskelværdi skal sættes på en helt anden måde. Denne tærskelværdi sættes som en procent af det rigtige resultat. Det kan fx være ±10%, denne tærskelværdi sættes til ±2%.

Den sidste tærskelværdi der skal sættes er hvor lille σ_0 skal være for at være tilstrækkelig lille. Denne sættes til 0.05. Argumentet for at sætte disse tærskelværdier til det som er præsenteret her, er at det ser ud som om at de fleste robuste algoritmer kan overholde disse tærskelværdier.

Lineær	Afstand	50	±0.008	49.992 – 50.008
	b	3	±0.008	2.992 – 3.008
	tx	2	±0.008	1.992 – 2.008
	ty	4	±0.008	3.992 – 4.008

Rækken 'afstand' i skemaet, viser hvilket interval de fundne afstande skal ligge indenfor, for at blive godtaget. Rækken 'b', gælder for den fundne konstant i forbindelse at finde linjens ligning, og rækkerne 'tx' og 'ty' gælder for translationsvektoren, i forbindeles med transformationsforsøgene.

Ikke-lineær	ax	±0.02	0.98 – 1.02
	skala	±0.018	0.882 – 0.918
	rotation	±0.3	14.7 – 15.3

I skemaet ses en oversigt over tærskelværdierne. I kolonnen hvor der står ±0.02 osv. vises hvilken tærskelværdi 2% bliver til, 'ax', står for hældningskoefficienten i forbindelse med linjens ligning. Skala og rotation er i forbindelse med helmert transformationen. Tærskelværdierne er udregnet i forhold til det rigtige resultat. Det gøres her opmærksom på at 'ax' ledet i linjens ligning er 1, skalaen i forbindelse med helmert transformationen er 0.9, og rotationen i helmert transformationen er 15 grader. Sidste kolonne i skemaet, viser hvilket interval resultat skal ligge indenfor, for at blive godtaget.

Ud fra de fastsatte kriterier laves der en sammentælling, for hvor mange af problemerne som algoritmerne løser. Dette sættes i forhold til hvor mange problemer som algoritmerne blev stillet overfor. Dette udregnes som en procent sats.

I første omgang opsættes et binært skema, som viser om algoritmerne har klaret opgaven, indenfor de angivne tærskelværdier. Algoritmerne får et plus, hvor opgaven er løst indenfor de givne kriterier, eller et minus, hvis ikke opgaven er løst inden for det givne kriterie.

	Data snooping	IRLS	RANSAC	RIL
Afstand	+	-	+	-
	+	+	+	-
	-	-	+	-
Ret linje	+	-	+	+
	-	-	+	+
	-	-	+	+
Helmert	-	-	+	+
	+	+	+	-
	+	-	-	-
Sum af succeser	5	2	8	4
Succesrate	55.56%	22.22%	88.89%	44.44%

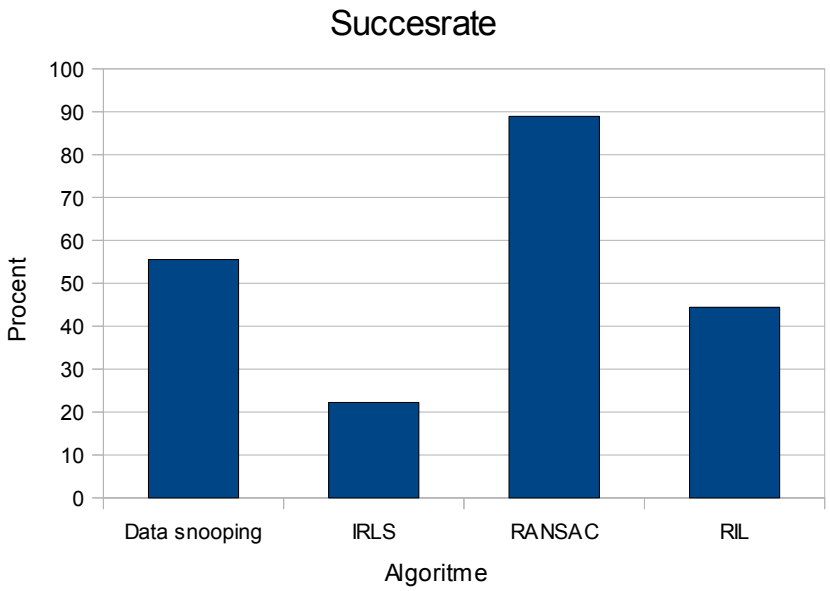


Illustration 12:Figuren viser algoritmernes succesrate.

I forbindelse med fortolkningen, af skemaet og figuren på forrige side, er det vigtigt at være opmærksom, på at der i vurderingen er gjort brug, af den viden som der ligger bag eksperimenterne. Denne viden omfatter, at det rigtige resultat er kendt, samt at antallet af grove fejl er kendt. Det er viden, som ikke er tilgængelig i virkligheden, når der foretages robust udjævning, på data der er indsamlet, med instrumenter, frem for konstruerede data, som er tilfældet i dette projekt. I virkeligheden vil det være tæt på umuligt at vurdere løsningen ud fra de fundne parametre. Derimod vil en løsning kunne vurderes ud fra σ_0 , som vil være den eneste størrelse, pånær residualerne, som der kan vurderes på. I næste skema, udregnes der en ny succesrate. Til forskel fra forrige vurderes der nu udelukkende på om σ_0 er tilstrækkelig lille. Der fortsættes med en tærskelværdi på 0.05 for σ_0 .

	Data snooping	IRLS	RANSAC	RIL
Afstand	+	+	+	+
	+	+	+	+
	-	-	+	+
Ret linje	+	-	+	+
	-	-	+	+
	-	-	+	+
Helmert	-	+	+	+
	+	+	+	+
	+	+	+	+
Sum af succeser	5	5	8	4
Succes rate	55.56%	55.56%	100%	100%

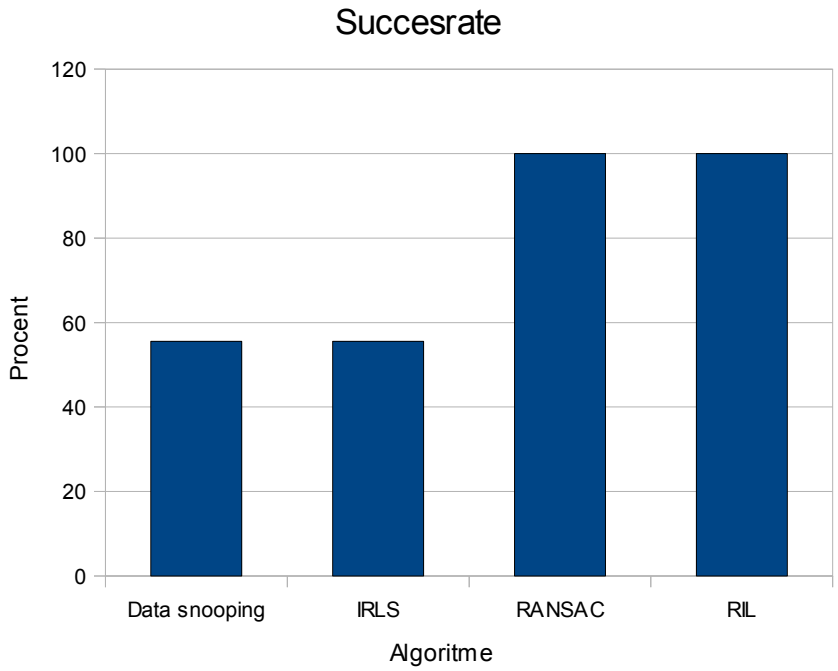


Illustration 13:Figuren viser succesraten, vurderet udelukkende ud fra σ_0 .

Umiddelbart ser det ikke godt ud i første skema og graf, da IRLS og RIL klarer sig dårligt, i forhold til de andre. Dette forhold er umiddelbart overraskende. Grunden til at resultaterne ser ud som de gør i første udregning af succesraten, er at der er gjort brug af viden omkring forsøgene. At det rigtige

resultat er kendt, samt at antallet af grove fejl er kendt. Alle de forsøg som RIL ikke klarer i første succesrate, er på grund af at RIL fjerner for mange observationer. 3 af de forsøg som IRLS ikke har klaret, i første succesrate, gælder samme problemstilling. Der er enten fundet for mange eller for få fejl i datasættet. I forsøgene hvor der regnes på en ret linje, fejler IRLS hver gang, men dette hænger sammen med at IRLS starter med et L_2-estimat. Hvis dette estimat går tilstrækkelig galt, kan IRLS ikke rette op på det, ligesom det er demonstreret i foranalysen. Hvis IRLS i forsøgene hvor der regnes på en ret linje, havde været skrevet med en generel mindste kvadraters algoritme, havde resultaterne for disse forsøg sandsynligvis taget sig anderledes ud.

8.2 Problemsøgning

I foregående afsnit blev der klarlagt at nogle algoritmer finder for få fejl, mens andre finder for mange fejl. I dette afsnit søges det at klarlægge hvilke forhold, som kunne medføre sådanne resultater.

8.2.1 Normalisering af residualer

Både IRLS og RIL gør brug af et robust estimat for de normaliserede residualer. Dette estimat ser således ud: $u = \frac{r}{s}$, hvor s er givet ved: $s = \frac{med\left(\left|r - med\left(r\right)\right|\right)}{0.6745}$

Denne måde at normalisere residualer på, adskiller sig meget fra den måde MKP formelsættet normalisere residualer på, hvor Qr matricen udregnes først:

$Qr = W^{-1} - A \cdot (A^T \cdot W \cdot A)^{-1} \cdot A^T$,som derefter anvendes til at normalisere residualet: $w_i = \frac{r_i}{\sqrt{qr_i^{\,2}}}$.

I IRLS og RIL algoritmerne vurderes normaliserede residualer på ca. ±3 til at være behæftet med en grov fejl, i tråd med teorien fra MKP. Der søges altså her at dannes en sammenhæng mellem de robuste metoder og MKP. Da formlerne for de normaliserede residualer er hvidt forskellige, er det ikke sikkert, at denne sammenhæng eksisterer. Dette betyder at hvor et normaliseret residual på ca. ±3, normaliseret efter MKP, vurderes til at være behæftet med en grov fejl, er det ikke nødvendigvis ensbetydende med at et normaliseret residual på ±3, normaliseret efter det robuste udtryk, er behæftet med en grov fejl. Hvis sidste nævnte er tilfældet, vil det betyde at de tærskelværdier der er sat i IRLS og RIL, er sat forkert.

8.2.2 Vægtfunktionen

En anden grund til at resultaterne for IRLS og RIL ser ud som de gør kan ligge i den vægtfunktion der er anvendt. I disse algoritmer er det valgt at bruge Huber-vægtfunktionen. Problemerne kan skyldes 'knækket' i vægtfunktionskurve, som beskrevet i foranalysen. Et hvert residual, som falder indenfor tune-værdien bliver automatisk tildelt vægten 1, ligegyldigt hvor stort residualet end måtte være. Det vil sige at der er ikke noget at 'forhandle' med, så længe residualet ligger inden for tune-værdien. Omvendt, hvis et enkelt residual, ligger uden for tune værdien, skal denne enkelte observation, alene

bære minimeringen af residualvektoren.

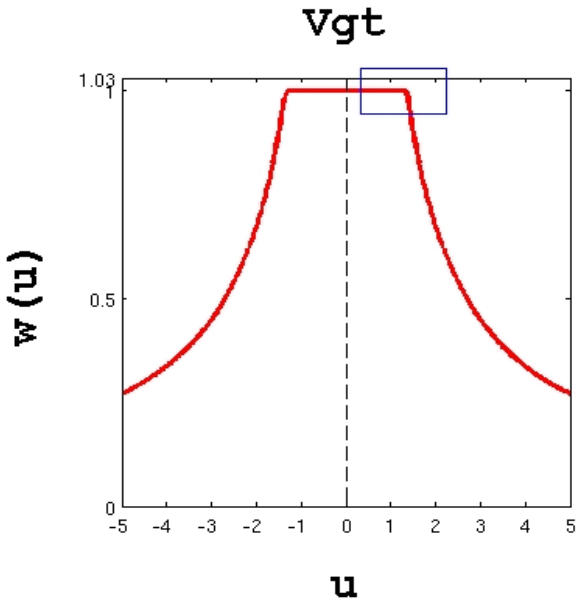


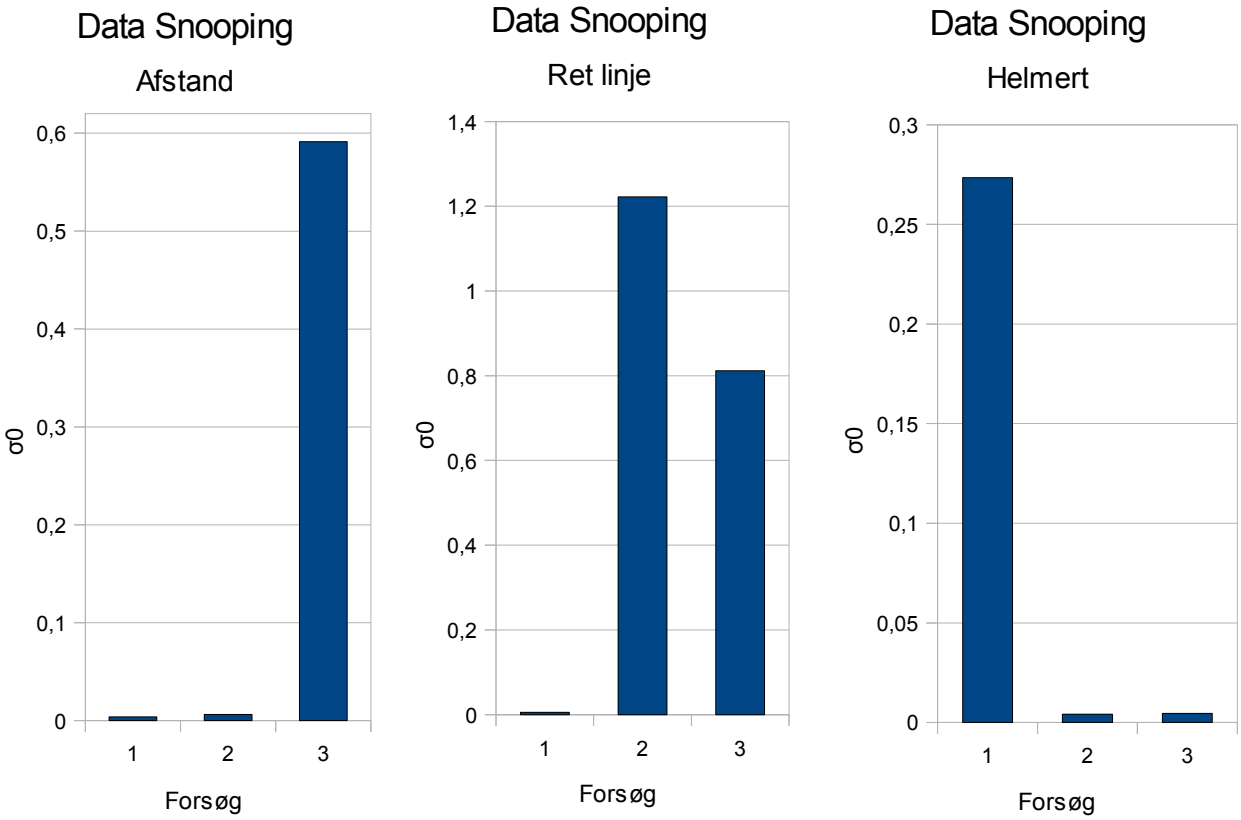
Illustration 14:Billedet illustrerer et muligt problem ved at anvende Huber vægte.

8.2.3 Andre forhold

Umiddelbart virker data snooping og IRLS en smule ustabile, da hverken IRLS eller data snooping for fundet frem til en brugbar løsning, da der udjævnes på afstandsdata, hvori der indgår 50 % grove fejl i datasættet, derimod kan de finde frem til en helmert transformation, ud fra et data sæt, hvori der indgår 50% fejl, hvis der ses bort fra at det rette resultat kendes. Dette kan muligvis skyldes at L_2-estimatet ikke er ligeså påvirket af de grove fejl, i helmert transformationen, som i afstandsmålingen, på grund af geometrien. Hertil er det bemærkelsesværdigt at i første forsøg, med at finde en ret linje, ud fra 5 observationer og 1 grov fejl, kan data snooping håndtere situation, mens det går galt for IRLS. Dette kan skyldes at den grove fejl medfører et residual der er større, end den tærskelværdi som data snooping anvender, men alligevel stor nok til at IRLS ikke kan håndtere situationen.

8.3 Data Snooping resultater

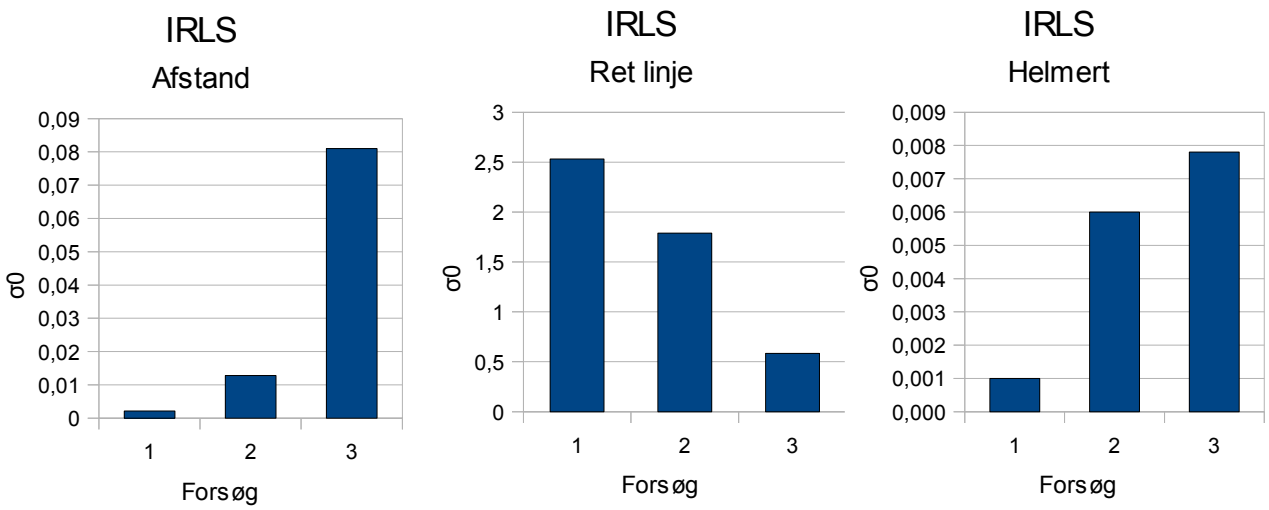
I dette afsnit gennemgås resultaterne for data snooping. I første omgang sættes σ_0 fra data snooping algoritmerne op i grafer, forsøg for forsøg.



Tallene på x-aksen henviser til forsøgs nummeret. I forsøg 1 er der 5 observationer og 1 grov fejl, i forsøg 2 er der 10 observationer og 3 grove fejl, og i forsøg 3 er der 20 observationer og 10 grove fejl, dette gælder også for efterfølgende grafer. Umiddelbart ser data snooping ud til at håndtere udjævning af en afstand ganske udmærket, med et datasæt med op til 30% grove fejl. Data snooping er dog ikke så god til at håndtere udjævning af en ret linje. I første forsøg går det godt, hvilket undrer, da IRLS ikke kan håndtere denne situation. I forbindelse med udjævning af en transformation kommer data snooping med igen, pånær i første forsøg. Dette formodes i høj grad at skyldes geometrien i første forsøg. Det er muligt at den grove fejl ødelægger estimatet på en sådan måde at data snooping ikke kan håndtere situationen.

8.4 IRLS resultater

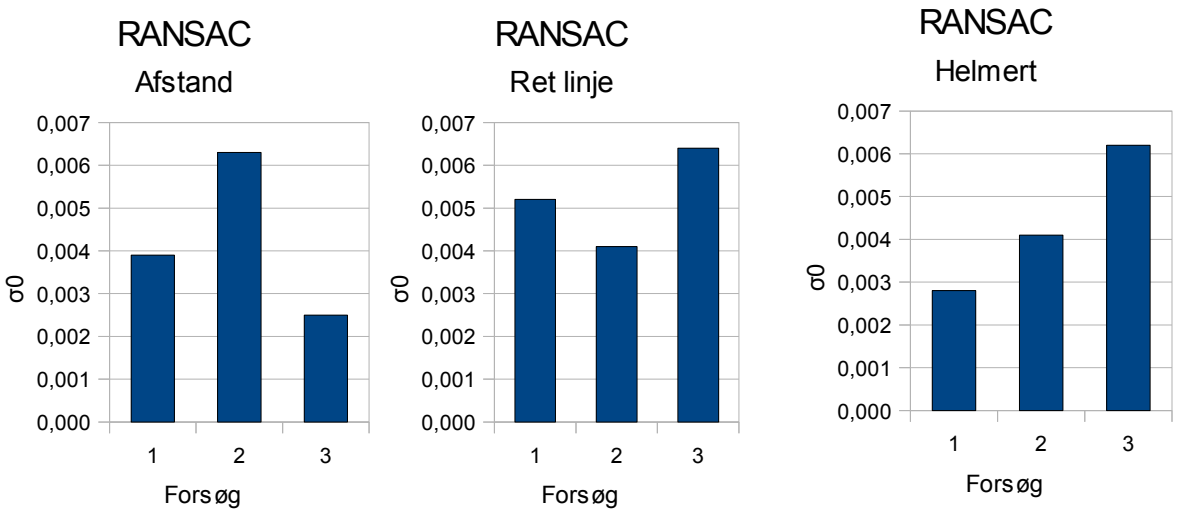
I dette afsnit gennemgås resultaterne fra IRLS, først sættes σ_0 fra IRLS op i grafer, forsøg for forsøg.



For IRLS er der en generel tendens til at σ_0 bliver større når antallet af grove fejl i datasættet stiger. Her ses der bort fra udjævningen af en ret linje, hvor IRLS på intet tidspunkt kommer frem til det rigtige resultat, eller et brugbart resultat. Det kan umiddelbart se ud som om at IRLS, med Huber vægte ikke kan håndtere 50% grove fejl. Dette er vurderet ud fra at det burde være nemmere for IRLS at håndtere afstandsforsøgene end transformationsforsøgene. Grunden til at IRLS håndterer transformationen, med 50% grove fejl og ikke afstanden med 50% grove fejl, kan skyldes geometrien i de punkter, som der transformeres over. Det vil sige at det ser ud som om at geometrien har større betydning for success end antallet af gode observationer, i forbindelse med transformationen. Her menes at geometrien blandt de gode observationer, er vigtigere end antallet af gode observationer.

8.5 RANSAC resultater

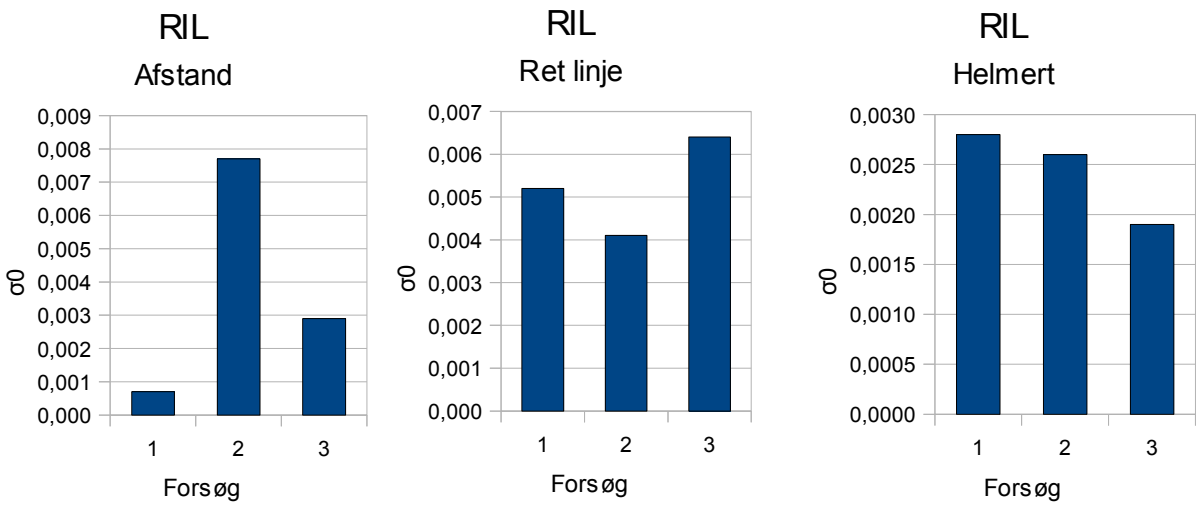
I dette afsnit gennemgås resultaterne fra RANSAC, først sættes σ_0 fra RANSAC op i grafer, forsøg for forsøg.



RANSAC er den algoritme i projektet, der overordnet klarer sig bedst i alle forsøgene, når der bl.a. vurderes på om den finder alle fejlene og kun fejlene. Ud fra de to første grafer, for afstand og linje, kan der ikke udledes nogen tendens, for hvordan RANSAC håndtere et stigende antal fejl i datasættet. Derimod ser det ud som om at σ_0 stiger, når antallet af grove fejl stiger, i forbindelse med transformationen. På grund af RANSACs 'trial and error' tilgang til problemet, virker det mere som en tilfældighed end som systematik. Data snooping og IRLS lader til at indikere at geometrien har større betydning i forhold til transformationen end antallet af grove fejl i datasættet. Dette kan muligvis også have indflydelse på de resultater som RANSAC opnår i disse opgaver. Med det menes at RANSACs σ_0 stiger, i transformationsforsøgene, har nok mere noget med geometrien at gøre end antallet af grove fejl, alternativt kan det skyldes at residualvektoren bliver større, da antallet af gode observationer stiger.

8.6 RIL resultater

I dette afsnit gennemgås resultaterne fra RIL, først sættes σ_0 fra RIL op i grafer, forsøg for forsøg.



Ligesom ved RANSAC, ser det heller ikke ud til at der er noget system i hvordan σ_0 udvikler sig når procentdelen af grove fejl stiger, i datasættet. I forbindelse med transformationen, ser det ud som om at σ_0 falder, når procentdelen af grove fejl stiger. Forklaringen på dette har sandsynligvis noget med antallet af observationer at gøre. Det vil sige, antallet af gode observationer. RIL starter, jf. implementeringsafsnittet, med en RANSAC algoritme, derefter en IRLS algoritme og til sidst et alm. MKP estimat. Når antallet af gode observationer stiger, er der flere overbestemmelser, som IRLS delen kan begynde at vægte på. Det formodes, umiddelbart at RANSAC algoritmen fjerner alle de grove fejl, da dennes tærskelværdier er præcis de samme, som for den enkeltstående RANSAC algoritme. Dette betyder at IRLS-delen kan begynde at vægte de tilbageværende observationer, så residualvektoren bliver mindre. Gennem alle forsøgene, pånær udjævning af en ret linje, har RIL haft tendens til at fjerne for mange observationer. Dette betyder at når σ_0 beregnes i RIL, så har residualvektoren, som bruges til at regne dette ud, haft færre elementer, end residualvektoren, for de andre algoritmer. Der er to bud på hvorfor RIL sorterer for mange observationer fra. Det første bud er at tærskelværdien for frasorteringen af observationer, er sat for højt. Denne tærskelværdi er sat så den stemmer nogenlunde overens med den tærskelværdi som data snooping fra sorterer efter. Da residualerne er normaliseret, på to forskellige måder, er det ikke sikkert at tærskelværdien kan sættes ens for de to algoritmer. Et andet bud, kan være at det er en forkert vægt funktion der anvendes. Når det normaliserede residual, ligger indenfor Huber's tune værdi, har algoritmen ikke noget at forhandle med. Det betyder, at så snart et residual ligger uden for denne tune-værdi, skal denne observation, bære hele minimeringen af residualvektoren. En vægtfunktion som bi-square, som præsenteret i foranalysen, kunne måske have givet andre resultater, i forbindelse med RIL algoritmen.

8.7 Samlet overblik

Umiddelbart kan det være svært at overskue resultaterne af forsøgene, når der kigges på tabellerne. Derimod kan der i udregningen af succesraten dannes et billede af at nogen af algoritmerne ser ud til at håndtere problemerne bedre end andre. Især den enkeltstående RANSAC algoritme har vist sig at være stabil. Kun i få tilfælde får de andre algoritmer en σ_0 , som er mindre end den RANSAC opnår. Dette er når der tages forbehold for at RANSAC er den metode, som overordnet, kommer tættest på at fjerne alle fejlene, uden at fjerne for meget. Hvis der kun kigges på σ_0 ser RIL algoritmen ud til at klare sig bedre.

9 Konklusion

I problemformuleringen blev det valgt at undersøge en række robuste udjævningsmetoder, for at finde ud af om der skulle være en eller nogen metoder, som er bedre end andre. I projektets start og lige op til forsøgene, blev det forventet at der var forskelle, men dog ikke store forskelle. Der har under hele projektet perioden været bekymring for om resultaterne ville være klare nok til at der kunne drages pålidelige konklusioner ud fra dem. Resultaterne af forsøgene har vist en spredning af algoritmernes ydelse, som ikke helt var forventet. At én algoritme kunne klare sig bedre end en anden, i et forsøg, også omvendt i næste forsøg, var umiddelbart forventet. Det var dog ikke forventet at der skulle tegne sig et så klart billede, som der gjorde. Den algoritme, som har præsteret den bedste ydelse, i forhold til at opnå et godt resultat, en lille σ_0 og finde alle fejlene, og kun fjerne fejlene, har været RANSAC. Havde IRLS gjort brug af generel mindste kvadraters princip i stedet for alm. mindste kvadraters princip, havde resultaterne muligvis taget sig anderledes ud. Havde dette været tilfældet, kunne IRLS algoritmen muligvis have klaret sig ligeså godt som RANSAC. Det var umiddelbart overraskende at RIL algoritmerne sorterede flere observationer fra, end der var fejlramt, men dette er sandsynligvis et justeringsspørgsmål. RIL skal enten bruge en anden tærskelværdi, for frasortering af observationer, eller en anden vægtfunktion, evt. begge. RIL er den eneste algoritme som i hvert eneste forsøg har fra sorteret alle fejl, men har som nævnt også sorteret for mange observationer fra. Det vil sige, observationer, som ikke var behæftet med grove fejl. Dette kan være et stort problem for datasæt, med få observationer.

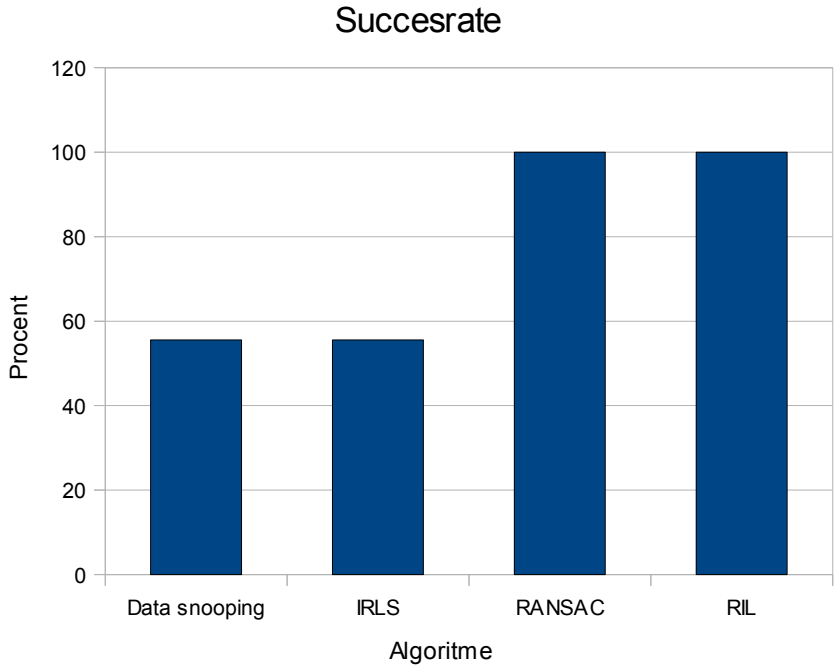


Illustration 15:Figuren viser succesraten, vurderet udelukkende ud fra σ_0 .

Overordnet ser det også ud som om at nogen udjævningsmetoder egner sig bedre til visse opgaver end

andre. Umiddelbart ser det ud som om at IRLS ikke håndterer udjævning af en ret linje særlig godt, men dette formodes at være på grund af at der bruges en alm. mindste kvadraters metode i IRLS og dette resultat er derfor ikke særlig overraskende. RANSAC og RIL har i den forbindelse, vist sig at være meget alsidige i forhold til at løse opgaverne.

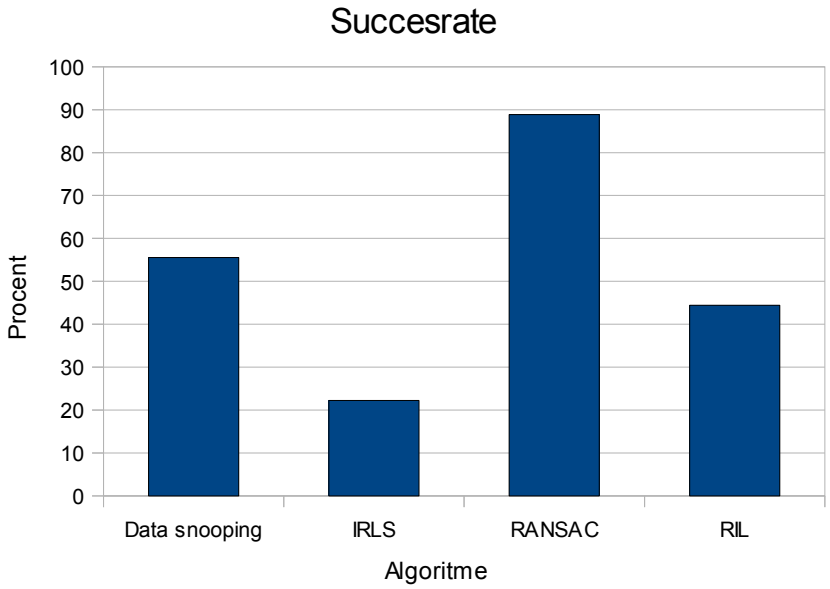


Illustration 16:Figuren viser algoritmernes succesrate.

Det ser ud som om at algoritmer, afledt af RANSAC og algoritmer, som er kombinerede, klarer sig bedre end andre algoritmer. I dette projekt var der en forventning om at en algoritme, som starter med en RANSAC algoritme, som derefter fortsætter i en IRLS algoritme og ender med et MKP estimat, ville være bedre rustet til at håndtere grove fejl. Denne antagelse har til dels vist sig at holde stik. RIL algoritmen har i hvert forsøg der er foretaget, fået fjernet alle grove fejl, problemet med den, er at den har fjernet for mange observationer.

Som det kan ses af resultaterne, har formodningen om at nogle robuste udjævningsmetoder er bedre end andre, holdt. Dette virker umiddelbart naturligt. Selvom undertegnede ikke er bevidst om det, synes det logisk at der er en kontinuær udvikling, af robuste udjævningsmetoder. En sådan udvikling vil nødvendigvis medføre at nogle robuste udjævningsmetoder er bedre end andre, på en eller anden skala. Dette kommer i høj grad an på hvad der søges. I nogle situationer er det måske vigtigere at udjævningsmetoden er hurtigere, frem for at den når frem til resultat med en lille spredning på vægtenheden. Andre metoder, som i dette projekt, er måske designet til at levere et pålideligt resultat, med en lille spredning på vægtenheden, på bekostning af beregningstid. I visse situationer kan man blive fanget af opgavens natur, som derefter dikterer hvilken robust udjævningsmetode der anvendes. Dette kan fx være polygonudjævning. Her er det, som nævnt, ikke muligt at anvende RANSAC, umiddelbart. En sådan opgave, formodes, bedst løst af IRLS. I situationer, som ikke fordrer en bestemt måde at foretage robust udjævning på, kan det være relevant at gøre sig tanker om hvilken metode der skal anvendes. Fx hvis der udjævnes på en afstand, koordinat, linje eller transformation. I disse situationer er det relevant at gøre sig tanker om hvilken robust udjævningsmetode, for bedst at indfri opgavens krav.

Litteratur liste

[P. Cederholm, Modelling Based on Coordinates, kursusgang 4, slide 4, 2009] Slide fra undervisning af Peter Cederholm.

[2002 John Fox, Robust Regression – Appendix to An R and S-PLUS Companion to Apllied Regression, s. 1]

Et appendix til et 'R and S-PLUS Companion to Applied Regression

[J. Heikkilä, Robust Regression, Graduate course on Advanced statistical signal processing, s. 23]

Kursus materiale fra Janne Heikkilä, kursus i Robust Regression.

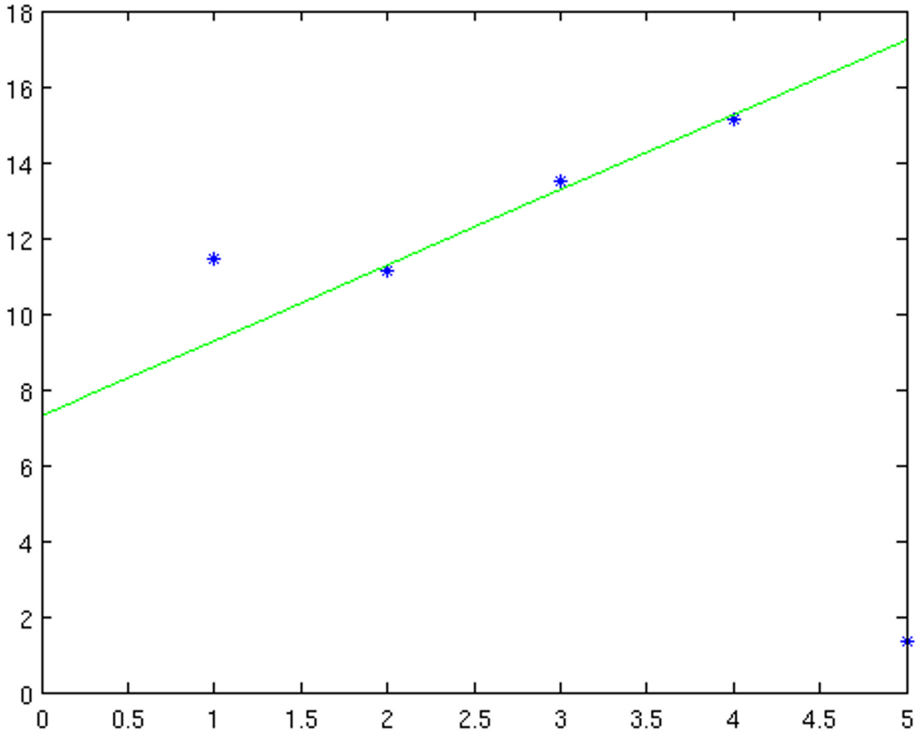
[Jens Juhl 2010] Citat fra vejleder.

[M. Zuliani 2009 "RANSAC4Dummies.pdf" s. 14]

Kort artikel om RANSAC, skrevet af Marco Zuliani.

Sammenligning af robuste udjævnings metoder

Bilag



af
L10MS.02
Jacob Collstrup

Indholdsfortegnelse

1 Script til generering af afstand....2
2 Script til generering af linjedata....2
3 Scripts til generering af transformationsdata....3
4 MKP+data snooping kode....7
5 IRLS kode....13
6 RANSAC kode....17
7 RIL kode....31
8 Succesrate....51
9 Individuelle resultater fra programmerne....51
10 Plot af linjer....51
10.1 Data Snooping:....51
10.2 IRLS....53
10.3 RANSAC....54
10.4 RIL....56
11 Afstandsdata....57
11.1 Afstandsdata, 5 observationer, 1 grov fejl....58
11.2 Afstandsdata, 10 observationer, 3 grove fejl....58
11.3 Afstandsdata, 20 observationer, 10 grove fejl....58
12 Ret linje data....59
12.1 Retlinje, 5 observationer, 1 grov fejl....60
12.2 Ret linje, 10 observationer, 3 grove fejl....61
12.3 Ret linje, 20 observationer, 3 grove fejl....61
13 Transformationsdata....62
13.1 Transformationsdata, 5 observationer, 1 grov fejl....63
13.2 Transformationsdata, 10 observationer, 3 grove fejl....64
13.3 Transformationsdata, 20 observationer, 10 grove fejl....65

1 Script til generering af afstand

```
%Dette script genererer den data der skal bruges til udjævning af
en afstandsmåling.


b = normrnd(50,0.004,[20,1]);
```

2 Script til generering af linjedata

```
clear all
close all
clc


%Dette script genererer data til approksimation af en linie.


x = [];


for i = 1:20
    x = [x;i];
end


for i = 1:20
   Y_n = x + 3;
end


Y = normrnd(Y_n,0.004)
```

3 Scripts til generering af transformationsdata

```
close all
clear all
clc

format short g

%Dette script genererer data til en 2d transformation.

x_n = normrnd(0,1,[20,1]);

y_n = normrnd(0,1,[20,1]);

x_m = x_n*10;

y_m = y_n*10;

p = size(x_n,1);

x = [];

y = [];

for i = 1:p
    x = [x; x_m(i) + 30;];
    y = [y; y_m(i) + 30;];
end

plot(x,y,'*')
```

```
close all
clear all
clc

%Dette program laver koordinaterne i TIL systemet.

input = [28.978 19.109
27.586 30.326
33.192 35.525
33.129 41.006
21.351 45.442
29.699 30.859
28.351 15.084
36.277 22.577
40.933 19.384
41.093 53.505
21.363 23.844
30.774 37.481
17.859 28.076
18.865 38.886
29.932 22.352
45.326 15.977
22.303 15.776
33.714 34.882
27.744 28.226
41.174 28.039];

s = 0.9;

rot = 15;

tx = 2;

ty = 4;
```

```
tv = [tx ty]';

a11 = cosd(rot);

a12 = -sind(rot);

a21 = sind(rot);

a22 = cosd(rot);

A = [a11 a12
    a21 a22];

m = size(input,1);

til = [];

for i = 1:m
    xf = input(i,1);
    yf = input(i,2);

    fra = [xf yf]';

    til = [til, A*s*fra + tv];
end
```

```
close all
clear all
clc

%Dette script sætter støj på X- og Y-koordinaterne i TIL systemet,
til brug ved transformationsforsøgene.

input  = [22.74 27.362
18.917 36.789
22.58 42.615
21.248 47.365
9.976 48.478
20.63 37.745
23.133 23.717
28.278 32.077
33.069 30.386
25.26 60.086
15.017 29.705
20.022 43.752
10.985 32.567
9.342 42.199
22.814 30.404
37.682 28.447
17.714 22.91
23.183 42.177
19.544 35
31.263 37.966];

noise_x = normrnd(0,0.004,[20,1]);

noise_y = normrnd(0,0.004,[20,1]);

x = input(:,1);
y = input(:,2);
```

```
xt = x + noise_x;

yt = y + noise_y;
```

4 MKP+data snooping kode

```
close all
clear all
clc



%This program takes n number of observations of 1 dimension
(Distances,
%angles, etc. and makes robust adjustment on them. The idea is to
have a
%number of observations that, besides being affected by gaussian
noise,
%also contain an unknown number of gross errors.


%This particular script uses the ordinary least squares method and
data-snooping.


b = load('dist20.txt');


n = 1; %Number of unknowns


while 1
    m = size(b,1);


    C = eye(m);


    A = ones(m,1);


    N = A'*C*A;
```

```
    Qr = (inv(C) - A*inv(N)*A');

    xhat = inv(A'*C*A)*A'*b;

    r = A*xhat - b;

    W = [];

    for i = 1:m
        W = [W, r(i)/sqrt(Qr(i,i))];
    end

    index = find(abs(W) == max(abs(W)));

 if abs(W(index)) > 2.8
     b(index) = [];
 else
     break
 end
end


sigma_0 = sqrt((r'*r)/(m-n))
```

```
close all
clear all
clc

%This script estimates a straight line from a dataset. It uses
Data
%snooping to detect and remove outliers.

input = load('line_data.txt');

x = input(:,1);
y = input(:,2);

n = 2; %Number of unknowns



while 1

%x = input(:,1);
%y = input(:,2);

p = size(y,1);

A = [];

for k = 1:p
    A = [A;x(k) 1;];
end

b = y;

m = size(b,1);

C = eye(m);
```

```
N = A'*C*A;

Qr = (inv(C) - A*inv(N)*A');

W = [];

xhat = inv(A'*A)*A'*b;

r = A*xhat - b;

for i = 1:m
    W = [W, r(i)/sqrt(Qr(i,i))];
end

index = find(abs(W) == max(abs(W)));

if abs(W(index)) > 2.8
    y(index) = [];
    x(index) = [];
else
    break
end
end

sigma_0 = sqrt((r'*r)/(m-n))
```

```
tic
close all
clear all
clc

%Dette script udregner transformationsparametrene til en 2d
helmert. Til
%udregningen anvendes alm. MKP og Data snooping.

input = load('Helmert_data20.txt');

xf = input(:,1);
yf = input(:,2);



xt = input(:,3);
yt = input(:,4);



q = 2; %Number of unknowns

while 1

n = numel(xf);

A = [xf -yf ones(n,1)  zeros(n,1);...
     yf  xf zeros(n,1) ones(n,1)];

b = [xt;yt];

xhat = inv(A'*A)*A'*b;

C = eye(numel(b));
```

```
N = A'*C*A;

Qr = (inv(C) - A*inv(N)*A');

W = [];

xhat = inv(A'*A)*A'*b;

r = A*xhat - b;

m = size(b,1);

for i = 1:m
    W = [W, r(i)/sqrt(Qr(i,i))];
end

[cuti cutj] = find(abs(W) == max(abs(W)));

if abs(W(cuti,cutj)) > 2.8
    xf(cuti) = [];
    yf(cuti) = [];

    xt(cuti) = [];
    yt(cuti) = [];
else
    break
end
end

xhat = inv(A'*A)*A'*b;

s = sqrt(xhat(1)^2+xhat(2)^2)            % Skala og rotation ser
ud til at vÃ¦re 'sat sammen'?
omega = atan2(xhat(2),xhat(1))/pi*180    % Her regnes Omega om fra
radianer til grader.
```

```
tx = xhat(3)
ty = xhat(4)

sigma_0 = sqrt((r'*r)/(m-q))

toc
```

5 IRLS kode

```
close all
clear all
clc
%This program takes n number of observations of 1 dimension
(Distances, angles, etc. and makes robust adjustment on them. The
idea is to have a number of observations that, besides being
affected by gaussian noise, also contain an unknown number of
gross errors.


%This particular script uses the IRLS method.


b = load('dist10.txt');


A = ones(size(b));


W = diag(A);


x_old = 0;


p = 1;


while 1
    xhat = inv(A'*W*A)*A'*W*b;
    r = A*xhat - b;
    W = robvegt(r,1);
    W = diag(W);
if max(abs(xhat-x_old)) < 0.0001
break
end
x_old = xhat;
end


xhat
```

```
sigma_0 = mad(r,1)/0.6745

close all
clear all
clc

%Dette script approksimerer en linie i 2d udfra en punktmÃ¦ngde.
Der anvedes
%alm. MKP i scriptet.



input = load('line_data.txt');

x = input(:,1);
y = input(:,2);

plot(x,y,'*')

p = size(y,1);

A = [];

for k = 1:p
    A = [A;x(k) 1;];
end

b = y;

m = size(b,1);

W = eye(m);

p = 1;
```

```
x_old = 0;

while 1
    xhat = inv(A'*W*A)*A'*W*b;
    r = A*xhat - b;
    W = robvegt(r,1);
    W = diag(W);
if max(abs(xhat-x_old)) < 0.0001
break
end
x_old = xhat;
end

sigma_0 = mad(r,1)/0.6745

a = xhat(1);
c = xhat(2);

hold on
fplot(@(x) a*x+c, [0 5], 'r')
```

```
close all
clear all
clc
tic
%Dette script udregner transformationsparametrene til en 2d
helmert. Til
%udregningen anvendes IRLS.

input = load('Helmert_data20.txt');

xf = input(:,1);
yf = input(:,2);



xt = input(:,3);
yt = input(:,4);

n = numel(xf);

A = [xf -yf ones(n,1)  zeros(n,1);...
     yf  xf zeros(n,1) ones(n,1)];

b = [xt;yt];



x_old = zeros(size(A,2),1);
W = eye(numel(b));
iteration = 1;
while 1
  xhat = inv(A'*W*A)*A'*W*b;
  r = A*xhat-b;
  W = robvegt(r,1);
  W = diag(W);
  if (max(abs(xhat-x_old)) < 0.0004) | (iteration > 1000000)
```

```
    break
  end
  x_old = xhat;
  iteration = iteration + 1;
end

sigma_0 = mad(r,1)/0.6745

s = sqrt(xhat(1)^2+xhat(2)^2)
omega = atan2(xhat(2),xhat(1))/pi*180
tx = xhat(3)
ty = xhat(4)
toc
```

6 RANSAC kode

```
close all
clear all
clc
This program takes n number of observations of 1 dimension
(Distances, angles, etc. and makes robust adjustment on them. The
idea is to have a number of observations that, besides being
affected by gaussian noise,
%also contain an unknown number of gross errors.

%This particular script uses the median.

b = load('dist20.txt');

n = 1; %Number of unkowns

%A = ones(size(b))';

xhat = median(b);
```

```
res = b - xhat;

res = abs(res);

ind = find(res > 0.015);

b(ind) = [];

m = size(b,1);

A = ones(size(b));

xhat2 = sum(b)/m

r2 = A*xhat2 - b;

sigma_0 = sqrt((r2'*r2)/(m-n))
```

```
close all
clear all
clc

%Dette script approksimerer en ret linie vha RANSAC.

%The script is based on the following pseudo-code

% input:
%     data - a set of observations
%     model - a model that can be fitted to data
%     n - the minimum number of data required to fit the model
%     k - the maximum number of iterations allowed in the
algorithm
%     t - a threshold value for determining when a datum fits a
model
%     d - the number of close data values required to assert that
a model fits well to data
% output:
%     best_model - model parameters which best fit the data (or
nil if no good model is found)
%     best_consensus_set - data point from which this model has
been estimated
%     best_error - the error of this model relative to the data
%
% iterations := 0
% best_model := nil
% best_consensus_set := nil
% best_error := infinity
% while iterations < k
%     maybe_inliers := n randomly selected values from data
%     maybe_model := model parameters fitted to maybe_inliers
%     consensus_set := maybe_inliers
%
%     for every point in data not in maybe_inliers
%         if point fits maybe_model with an error smaller than t
```

```
%             add point to consensus_set
%
%     if the number of elements in consensus_set is > d
%         (this implies that we may have found a good model,
%         now test how good it is)
%         better_model := model parameters fitted to all points in
consensus_set
%         this_error := a measure of how well better_model fits
these points
%         if this_error < best_error
%             (we have found a model which is better than any of
the previous ones,
%             keep it until a better one is found)
%             best_model := better_model
%             best_consensus_set := consensus_set
%             best_error := this_error
%
%     increment iterations
%
% return best_model, best_consensus_set, best_error

% IndlÃ¦sning af punkter:

input = load('line_data.txt');

x = input(:,1);
y = input(:,2);

plot(x,y,'*')

n = 2;

best_model = 0;

res_old = inf;
```

```
best_error = inf;

best_score = 0;

t = 0.02;

best_consensus_set = [];

xhat = [];

m = size(x,1);

indices = nchoosek(1:m,2);

k = size(indices,1);

iteration_best = 0;

for i = 1:k
    ind = indices(i,1:2);
    x_rest = x;
    x_rest(ind) = [];
    y_rest = y;
    y_rest(ind) = [];
    %*************************************************************
*********
    %Now to extract the indices from the input matrix
    %*************************************************************
*********

    x_use = x(ind);
    y_use = y(ind);

    %*************************************************************
```

```
*********
    %Here we calculate the model based on the extracted
coordinates
    %*************************************************************
*********

    A = [x_use(1) 1
        x_use(2) 1];

    b = y_use;

    xhat = inv(A'*A)*A'*b;

    model = xhat;
    %*************************************************************
*********
    %Here we begin to test if the found model, xhat, fits the data
    %*************************************************************
*********
    score = 0;

    consensus_set_build = [];

    this_error = [];

    x_test = x;

    x_test(ind) = [];

    y_test = y;

    y_test(ind) = [];

    p = m - n;
```

```
    A_test = [];

for j = 1:p
    A_test = [A_test;x_test(j) 1;];
end

    b_test = y_test;

    res = A_test*xhat - b_test;

    test = find(abs(res) < t);

    consensus_set_build = [consensus_set_build; x_test(test)
y_test(test)];

    score = size(consensus_set_build, 1);

    res = sum(abs(res));

    this_error = res;

    build = [x_use y_use];

    consensus_set = [build; consensus_set_build];

    score = size(consensus_set,1);

    if (this_error < best_error) & (score > best_score)
        best_consensus_set = consensus_set;
        best_error = this_error;
        best_model = model;
        iteration_best = k;
    end
```

```
end

%Here we calculate a normal least squares estimate on the
%best_consensus_set

xls = best_consensus_set(:,1);
yls = best_consensus_set(:,2);

q = size(xls,1);

Als = [];

for h = 1:q
    Als = [Als;xls(h) 1;];
end

xhatls = inv(Als'*Als)*Als'*yls

r = Als*xhatls - yls;

sigma_0 = sqrt((r'*r)/(q-n))



a = xhatls(1);
c = xhatls(2);

hold on
fplot(@(x) a*x+c, [0 5], 'r')
axis equal
```

```
clear all
close all
clc
tic

%This program calculates a 6 or 7 parameter transformation, it
uses a
%RANSAC algorithm for detecting outliers. The program uses the
follwing
%scripts by Peter Cederholm: rot3d.m, trans3d.m and trans3pt. The
script is
%based on the following pseudo-code

% input:
%     data - a set of observations
%     model - a model that can be fitted to data
%     n - the minimum number of data required to fit the model
%     k - the maximum number of iterations allowed in the
algorithm
%     t - a threshold value for determining when a datum fits a
model
%     d - the number of close data values required to assert that
a model fits well to data
% output:
%     best_model - model parameters which best fit the data (or
nil if no good model is found)
%     best_consensus_set - data point from which this model has
been estimated
%     best_error - the error of this model relative to the data
%
% iterations := 0
% best_model := nil
% best_consensus_set := nil
% best_error := infinity
% while iterations < k
%     maybe_inliers := n randomly selected values from data
%     maybe_model := model parameters fitted to maybe_inliers
```

```
%     consensus_set := maybe_inliers
%
%     for every point in data not in maybe_inliers
%         if point fits maybe_model with an error smaller than t
%             add point to consensus_set
%
%     if the number of elements in consensus_set is > d
%         (this implies that we may have found a good model,
%         now test how good it is)
%         better_model := model parameters fitted to all points in
consensus_set
%         this_error := a measure of how well better_model fits
these points
%         if this_error < best_error
%             (we have found a model which is better than any of
the previous ones,
%             keep it until a better one is found)
%             best_model := better_model
%             best_consensus_set := consensus_set
%             best_error := this_error
%
%     increment iterations
%
% return best_model, best_consensus_set, best_error

input = load('Helmert_data20.txt');

xfra = input(:,1);
yfra = input(:,2);



xtil = input(:,3);
ytil = input(:,4);
```

```
fra = [xfra, yfra];

til = [xtil, ytil];

m = size(xfra,1);

res_old_trans = ones(1,4)*1000000;

res_old = res_old_trans';

best_consensus_set = [];

model = [];

xhat = [];

t = 0.02;

best_model = [];

best_error = inf;

iteration_best = 0;

best_score = 0;

%*****************************************************************
*********
%Here RANSAC's main loop starts (line 106)
%*****************************************************************
*********

indices =  nchoosek(1:m,2);
```

```
k = size(indices,1);

for i = 1:k
    ind = indices(i,1:2);

    %*************************************************************
*********
    %Now to extract the indices from the input matrix
    %*************************************************************
*********

    xf = xfra(ind);

    yf = yfra(ind);

    xt = xtil(ind);

    yt = ytil(ind);

    %*************************************************************
*********
    %Here we calculate the model based on the extracted
coordinates
    %*************************************************************
*********

    Ar = [xf -yf ones(2,1)  zeros(2,1);...
        yf  xf zeros(2,1) ones(2,1)];

    br = [xt;yt];

    xhat = inv(Ar'*Ar)*Ar'*br;

    model = xhat;
    %*************************************************************
*********
```

```
    %Here we begin to test if the found model, xhat, fits the data
    %*************************************************************
*********
    score = 0;

    consensus_set_build = [];

    this_error = [];

    x_test = xfra;

    y_test = yfra;

    xt_test = xtil;

    yt_test = ytil;

    q = size(x_test,1);

    A_test = [x_test -y_test ones(q,1)  zeros(q,1);...
        y_test x_test zeros(q,1) ones(q,1)];

    b_test = [xt_test;yt_test];


    res = A_test*xhat - b_test;

    res = reshape(res,length(res)/2,2);

    res = abs(res);

    [cuti cutj] = find(res > t);

    consensus_set = input;
```

```
    consensus_set(cuti,:) = [];

    score = size(consensus_set,1);

    this_error = sum(sum(abs(res)));

    if (this_error < best_error) & (score >= best_score)
        best_consensus_set = consensus_set;
        best_score = score;
        best_error = this_error;
        best_model = model;
        iteration_best = k;
    end

end

conxf = best_consensus_set(:,1);

conyf = best_consensus_set(:,2);

conxt = best_consensus_set(:,3);

conyt = best_consensus_set(:,4);

r = size(conxf,1);


A = [conxf -conyf ones(r,1)  zeros(r,1);...
     conyf  conxf zeros(r,1) ones(r,1)];

b = [conxt;conyt];

xhatls = inv(A'*A)*A'*b;
```

```
m = size(b,1);


n = 2; %Number of unknowns


rls = A*xhatls - b;


sigma_0 = sqrt((rls'*rls)/(m-n))


s = sqrt(xhatls(1)^2+xhatls(2)^2)            % Skala og rotation
ser ud til at vÃ¦re 'sat sammen'?
omega = atan2(xhatls(2),xhatls(1))/pi*180    % Her regnes Omega om
fra radianer til grader.
tx = xhatls(3)
ty = xhatls(4)
toc
```

7 RIL kode

```
close all
clear all
clc


%Dette script udjÃ¦vner pÃ¥ en afstand. Der tages udgangspunkt i
en median,
%pÃ¥ baggrund af denne median, fjernes de vÃ¦rste grove fejl.
Hvorefter der
%kÃ¸res en IRLS til at finde de sidste grove fejl. Tilsidste
anvendes der
%alm. udjÃ¦vning efter mindste kvadraters metode.



%Her indlÃ¦ses alle observationerne
b1 = load('dist20.txt');
```

```
n = 1; %Number of unknowns

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%
%
%
%                                RANSAC
%
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%


%Her findes medianen (LÃ¸sning i fÃ¸rste program stump)
res = median(b1);

%Her sÃ¦ttes lower bound, alt hvad der er mindre end dette fjernes
low1 = res - 0.015;

%Her sÃ¦ttes higher bound, alt hvad der er stÃ¸rre end dette
fjernes
high1 = res + 0.015;

%Her begyndes den kode der fjerner de 'dÃ¥rlige' observationer
m = size(b1,1);

l1 = [];

h1 = [];
for i = 1:m
    l1 = [l1; b1(i)<low1;];
    h1 = [h1; b1(i)>high1;];
end

il1 = find(l1);
```

```
ih1 = find(h1);

b1(il1) =[];
b1(ih1) =[];

b2 = b1;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%
%
%
%                                    IRLS
%
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%

%Design matrix
A = ones(size(b2));

%Residual vektor, baseret pÃ¥ forrige programstump
r = A*res - b2;

%VÃ¦gt vektor opstillet pÃ¥ baggrund af residualer fra forrige
programstump
[W,s] = robvegt(r,1);

%Her omsÃ¦ttes vektoren til en matrix
W = diag(W);

%Tom lÃ¸sning opstilles
x_old = 0;

%IRLS lÃ¸kke starter
while 1
    xhat = inv(A'*W*A)*A'*W*b2;
```

```
    r = A*xhat - b2;
    [W,s] = robvegt(r,1);
    W = diag(W);
if max(abs(xhat-x_old)) < 0.0001
break
end
x_old = xhat;
end

xhat;

%Herfra sorteres yderliger grove fejl fra

p = size(W,1);

cut = 0.5;

%Indekser der skal fjernes
vec = diag(W);

index = find(diag(W)<cut);

b3 = b2;

b3(index) = [];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%
%
%
%                          Mindste Kvadraters Estimat
%
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%
```

```
A = ones(size(b3));

xhat2 = inv(A'*A)*A'*b3;

r2 = A*xhat2 - b3;

m = size(b3,1);

sigma_0 = sqrt((r2'*r2)/(m-n))

xhat2
```

```
close all
clear all
clc
tic
%Dette script approksimerer en ret linie vha RANSAC.

%The script is based on the following pseudo-code

% input:
%     data - a set of observations
%     model - a model that can be fitted to data
%     n - the minimum number of data required to fit the model
%     k - the maximum number of iterations allowed in the
algorithm
%     t - a threshold value for determining when a datum fits a
model
%     d - the number of close data values required to assert that
a model fits well to data
% output:
%     best_model - model parameters which best fit the data (or
nil if no good model is found)
%     best_consensus_set - data point from which this model has
been estimated
%     best_error - the error of this model relative to the data
%
% iterations := 0
% best_model := nil
% best_consensus_set := nil
% best_error := infinity
% while iterations < k
%     maybe_inliers := n randomly selected values from data
%     maybe_model := model parameters fitted to maybe_inliers
%     consensus_set := maybe_inliers
%
%     for every point in data not in maybe_inliers
%         if point fits maybe_model with an error smaller than t
```

```
%             add point to consensus_set
%
%     if the number of elements in consensus_set is > d
%         (this implies that we may have found a good model,
%         now test how good it is)
%         better_model := model parameters fitted to all points in
consensus_set
%         this_error := a measure of how well better_model fits
these points
%         if this_error < best_error
%             (we have found a model which is better than any of
the previous ones,
%             keep it until a better one is found)
%             best_model := better_model
%             best_consensus_set := consensus_set
%             best_error := this_error
%
%     increment iterations
%
% return best_model, best_consensus_set, best_error

% IndlÃ¦sning af punkter:

input = load('line_data.txt');

x = input(:,1);
y = input(:,2);

plot(x,y,'*')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%
%
%
%                                       RANSAC
%
```

```
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%
n = 2;

best_model = 0;

res_old = inf;

best_error = inf;

best_score = 0;

t = 0.02;

best_consensus_set = [];

xhat = [];

m = size(x,1);

indices = nchoosek(1:m,2);

k = size(indices,1);

iteration_best = 0;

for i = 1:k
    ind = indices(i,1:2);
    x_rest = x;
    x_rest(ind) = [];
    y_rest = y;
    y_rest(ind) = [];
    %*************************************************************
```

```
*********
    %Now to extract the indices from the input matrix
    %*************************************************************
*********

    x_use = x(ind);
    y_use = y(ind);

    %*************************************************************
*********
    %Here we calculate the model based on the extracted
coordinates
    %*************************************************************
*********

    A = [x_use(1) 1
        x_use(2) 1];

    b = y_use;

    xhat = inv(A'*A)*A'*b;

    model = xhat;
    %*************************************************************
*********
    %Here we begin to test if the found model, xhat, fits the data
    %*************************************************************
*********
    score = 0;

    consensus_set_build = [];

    this_error = [];

    x_test = x;
```

```
    x_test(ind) = [];

    y_test = y;

    y_test(ind) = [];

    p = m - n;

    A_test = [];

for j = 1:p
    A_test = [A_test;x_test(j) 1;];
end

    b_test = y_test;

    res = A_test*model - b_test;

    test = find(abs(res) < t);

    consensus_set_build = [consensus_set_build; x_test(test)
y_test(test)];

    score = size(consensus_set_build, 1);

    res = sum(abs(res));

    this_error = res;

    build = [x_use y_use];

    consensus_set = [build; consensus_set_build];

    score = size(consensus_set,1);
```

```
    if (this_error < best_error) & (score > best_score)
        best_consensus_set = consensus_set;
        best_error = this_error;
        best_model = model;
        iteration_best = k;
    end

end

Ar = [];

q = size(best_consensus_set,1);

xr = best_consensus_set(:,1);

yr = best_consensus_set(:,2);

for l = 1:q
    Ar = [Ar; xr(l) 1;];
end

r = Ar*best_model - yr;



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%
%
%
%                                          IRLS
%
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%
```

```
%VÃ¦gt vektor opstillet pÃ¥ baggrund af residualer fra forrige
programstump
[W,s] = robvegt(r,1);

%Her omsÃ¦ttes vektoren til en matrix
W = diag(W);

%Tom lÃ¸sning opstilles
x_old = 0;

%IRLS lÃ¸kke starter
while 1
    xhatirls = inv(Ar'*W*Ar)*Ar'*W*yr;
    r = Ar*xhat - yr;
    [W,s] = robvegt(r,1);
    W = diag(W);
if max(abs(xhatirls-x_old)) < 0.001
break
end
x_old = xhatirls;
end

cut = 0.5;

index = find(diag(W)<cut);

b2 = yr;

b2(index) = [];

xr(index) = [];

u = size(xr,1);
```

```
Als = [];

for h = 1:u
    Als = [Als; xr(h) 1];
end

xhatls = inv(Als'*Als)*Als'*b2

rls = Als*xhatls - b2;

m = size(b2,1);

n = 2; %Number of unknowns

sigma_0 = sqrt((rls'*rls)/(m-n))

a = xhatls(1);
c = xhatls(2);

hold on
fplot(@(x) a*x+c, [0 5], 'r')
axis equal
toc
```

```
clear all
close all
clc
tic

%This program calculates a 6 or 7 parameter transformation, it
uses a
%RANSAC algorithm for detecting outliers. The program uses the
follwing
%scripts by Peter Cederholm: rot3d.m, trans3d.m and trans3pt. The
script is
%based on the following pseudo-code

% input:
%     data - a set of observations
%     model - a model that can be fitted to data
%     n - the minimum number of data required to fit the model
%     k - the maximum number of iterations allowed in the
algorithm
%     t - a threshold value for determining when a datum fits a
model
%     d - the number of close data values required to assert that
a model fits well to data
% output:
%     best_model - model parameters which best fit the data (or
nil if no good model is found)
%     best_consensus_set - data point from which this model has
been estimated
%     best_error - the error of this model relative to the data
%
% iterations := 0
% best_model := nil
% best_consensus_set := nil
% best_error := infinity
% while iterations < k
%     maybe_inliers := n randomly selected values from data
%     maybe_model := model parameters fitted to maybe_inliers
```

```
%     consensus_set := maybe_inliers
%
%     for every point in data not in maybe_inliers
%         if point fits maybe_model with an error smaller than t
%             add point to consensus_set
%
%     if the number of elements in consensus_set is > d
%         (this implies that we may have found a good model,
%         now test how good it is)
%         better_model := model parameters fitted to all points in
consensus_set
%         this_error := a measure of how well better_model fits
these points
%         if this_error < best_error
%             (we have found a model which is better than any of
the previous ones,
%             keep it until a better one is found)
%             best_model := better_model
%             best_consensus_set := consensus_set
%             best_error := this_error
%
%     increment iterations
%
% return best_model, best_consensus_set, best_error

input = load('Helmert_data20.txt');

xfra = input(:,1);
yfra = input(:,2);

fra = [xfra, yfra];

xtil = input(:,3);
ytil = input(:,4);
```

```
til = [xtil, ytil];

m = size(xfra,1);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%
%
%
%                                    RANSAC
%
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%

res_old_trans = ones(1,4)*1000000;

res_old = res_old_trans';

best_consensus_set = [];

model = [];

xhat = [];

t = 0.015;

best_model = [];

best_error = inf;

iteration_best = 0;

best_score = 0;

best_build = [];
```

```
%*************************************************************
*********
%Here RANSAC's main loop starts (line 106)
%*************************************************************
*********

indices =  nchoosek(1:m,2);

k = size(indices,1);

for i = 1:k
    ind = indices(i,1:2);

    %*************************************************************
*********
    %Now to extract the indices from the input matrix
    %*************************************************************
*********

    xf = xfra(ind);

    yf = yfra(ind);

    xt = xtil(ind);

    yt = ytil(ind);

    %*************************************************************
*********
    %Here we calculate the model based on the extracted
coordinates
    %*************************************************************
*********

    Ar = [xf -yf ones(2,1)  zeros(2,1);...
```

```
        yf  xf zeros(2,1) ones(2,1)];

    br = [xt;yt];

    xhat = inv(Ar'*Ar)*Ar'*br;

    model = xhat;
    %**************************************************************
*********
    %Here we begin to test if the found model, xhat, fits the data
    %**************************************************************
*********
score = 0;

    consensus_set_build = [];

    this_error = [];

    x_test = xfra;

    y_test = yfra;

    xt_test = xtil;

    yt_test = ytil;

    q = size(x_test,1);

    A_test = [x_test -y_test ones(q,1)  zeros(q,1);...
        y_test x_test zeros(q,1) ones(q,1)];

    b_test = [xt_test;yt_test];


    res = A_test*xhat - b_test;
```

```
    res = reshape(res,length(res)/2,2);

    res = abs(res);

    [cuti cutj] = find(res > t);

    consensus_set = input;

    consensus_set(cuti,:) = [];

    score = size(consensus_set,1);

    this_error = sum(sum(abs(res)));

    if (this_error < best_error) & (score >= best_score)
        best_consensus_set = consensus_set;
        best_score = score;
        best_error = this_error;
        best_model = model;
        iteration_best = k;
    end

end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%
%
%
%                                        IRLS
%
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%

conxf = best_consensus_set(:,1);
```

```
conyf = best_consensus_set(:,2);

conxt = best_consensus_set(:,3);

conyt = best_consensus_set(:,4);

n = size(conxf,1);


A = [conxf -conyf ones(n,1)  zeros(n,1);...
     conyf  conxf zeros(n,1) ones(n,1)];

b = [conxt;conyt];

x_old = zeros(size(A,2),1);

r = A*best_model - b;

[W,s] = robvegt(r,1);

W = diag(W);

iteration = 1;
while 1
  xhatirls = inv(A'*W*A)*A'*W*b;
  r = A*xhatirls-b;
  W = robvegt(r,1);
  W = diag(W);
  if (max(abs(xhatirls-x_old)) < 0.0001) | (iteration > 100000)
    break
  end
  x_old = xhatirls;
  iteration = iteration + 1;
```

```
end

cut = 0.5;

W = diag(W);

g = size(W,1);

W = reshape(W,length(W)/2,2);

[cuti cutj] = find( W < cut);

irlsxf = conxf;
irlsyf = conyf;

irlsxt = conxt;
irlsyt = conyt;

irlsxf(cuti) = [];
irlsyf(cuti) = [];

irlsxt(cuti) = [];
irlsyt(cuti) = [];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%
%
%
%                                Least Squares Estimate
%
%
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%

p = numel(irlsxf);
```

```
Als = [irlsxf -irlsyf ones(p,1)  zeros(p,1);...
     irlsyf  irlsxf zeros(p,1) ones(p,1)];

b2 = [irlsxt;irlsyt];



xhatls = inv(Als'*Als)*Als'*b2;

m = size(b2,1);

n = 2; %Number of unkowns

rls = Als*xhatls - b2;

sigma_0 = sqrt((rls'*rls)/(m-n))

s = sqrt(xhatls(1)^2+xhatls(2)^2)            % Skala og rotation
ser ud til at vÃ¦re 'sat sammen'?
omega = atan2(xhatls(2),xhatls(1))/pi*180    % Her regnes Omega om
fra radianer til grader.
tx = xhatls(3)
ty = xhatls(4)
toc
```

8 Succesrate

I dette bilag henvises der til filen 'Succesrate.ods', i bilagsmappen 'Databehandling'. Dette regneark indeholder de tal der lægger til grund for søjlediagrammerne med succesraterne.

9 Individuelle resultater fra programmerne

I dette bilag henvises der til filen 'Resultat gennemsnit.ods', i bilagsmappen 'Databehandling'. Dette regneark indeholder de tal der lægger til grund for søjlediagrammerne, der indgår i rapporten, hvor programmernes individuelle resultater bliver behandlet.

10 Plot af linjer

Dette bilag indeholder plots af de linjer som programmerne har fundet frem til, da der blev udjævnet på linjedata.

10.1 Data Snooping:

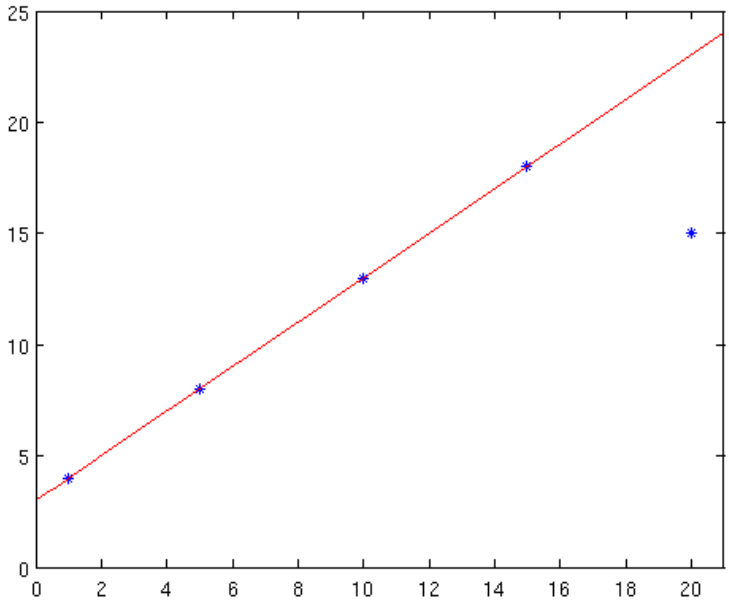


Illustration 1:Plottet viser den linje data snooping fandt frem til, i forsøget med 5 observationer og 1 grov fejl.

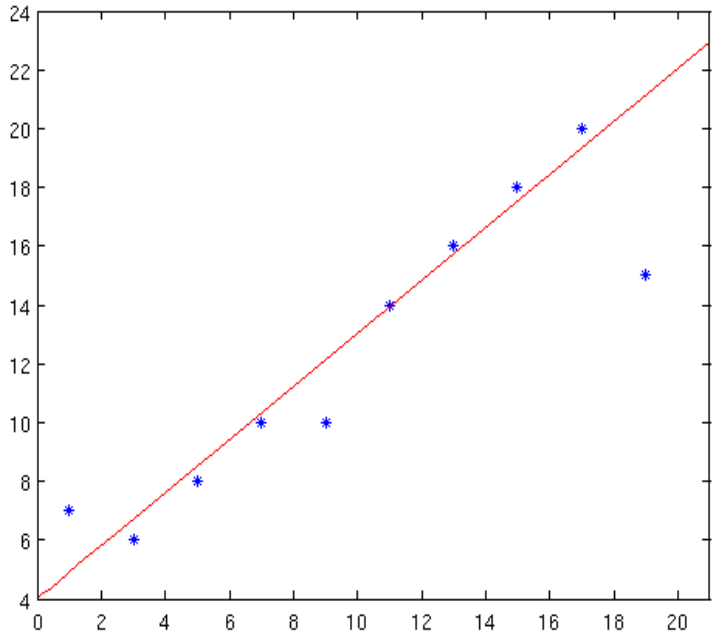


Illustration 2:Plottet viser den linje data snooping fandt frem til, i forsøget med 10 observationer og 3 grove fejl.

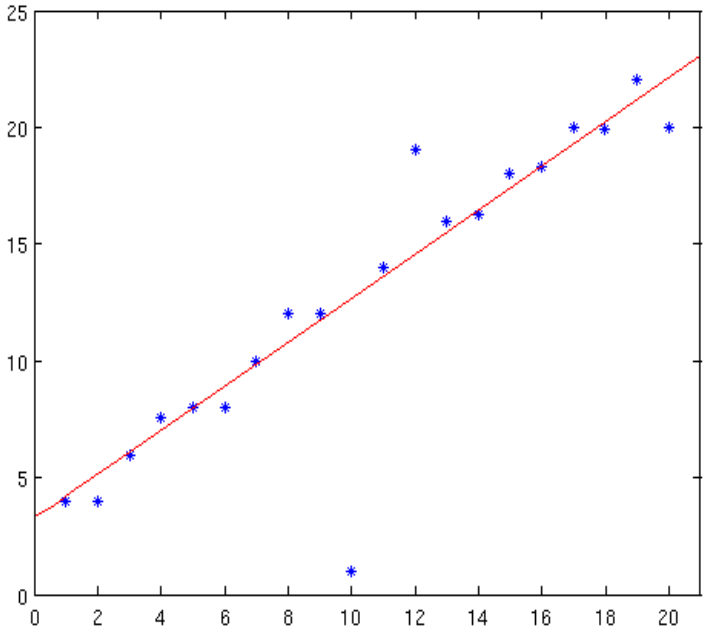


Illustration 3:Plottet viser den linje data snooping fandt frem til, i forsøget med 20 observationer og 10 grove fejl.

10.2 IRLS

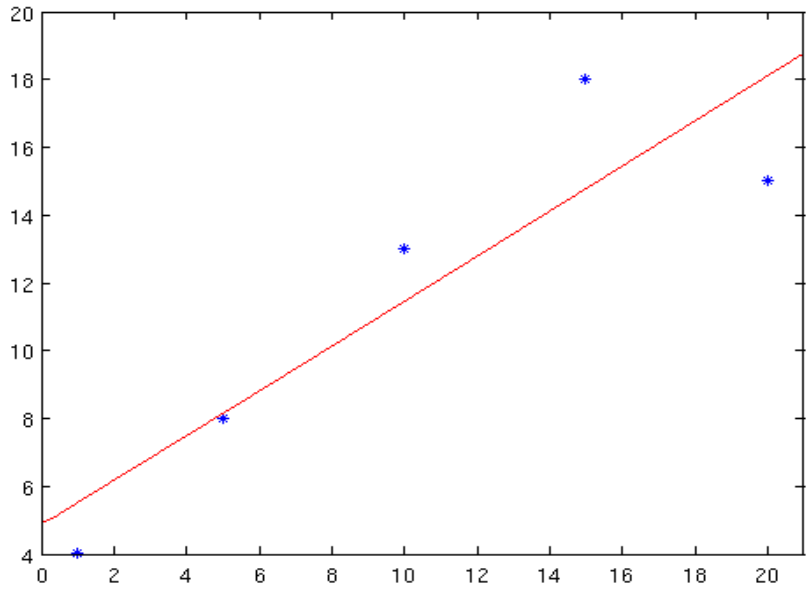


Illustration 4:Plottet viser den linje IRLS fandt frem til, i forsøget med 5 observationer og 1 grov fejl.

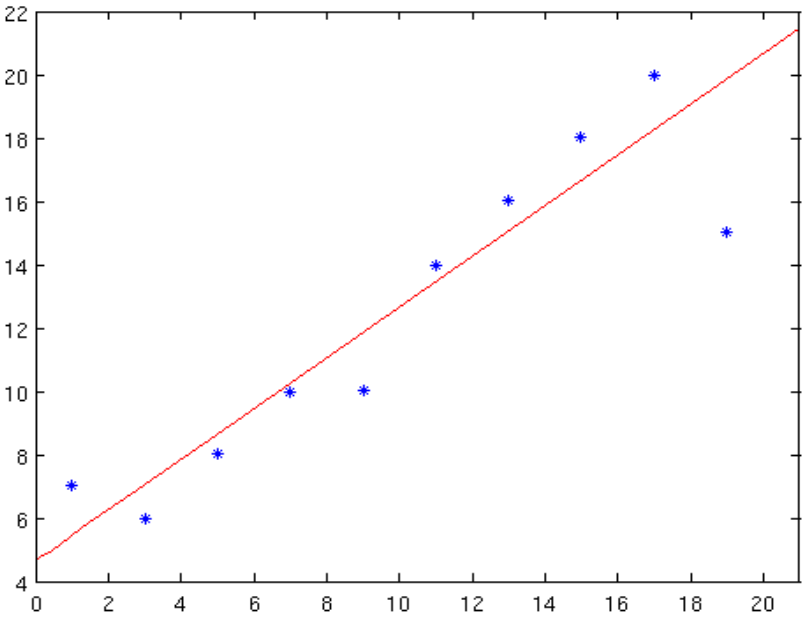


Illustration 5:Plottet viser den linje IRLS fandt frem til, i forsøget med 10 observationer og 3 grove fejl.

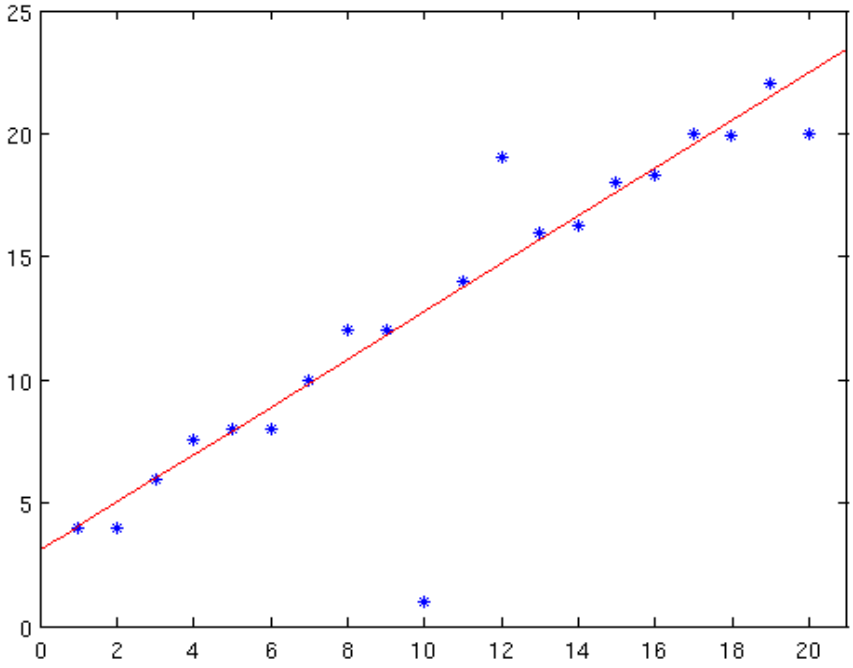


Illustration 6:Plottet viser den linje IRLS fandt frem til, i forsøget med 20 observationer og 10 grove fejl.

10.3 RANSAC

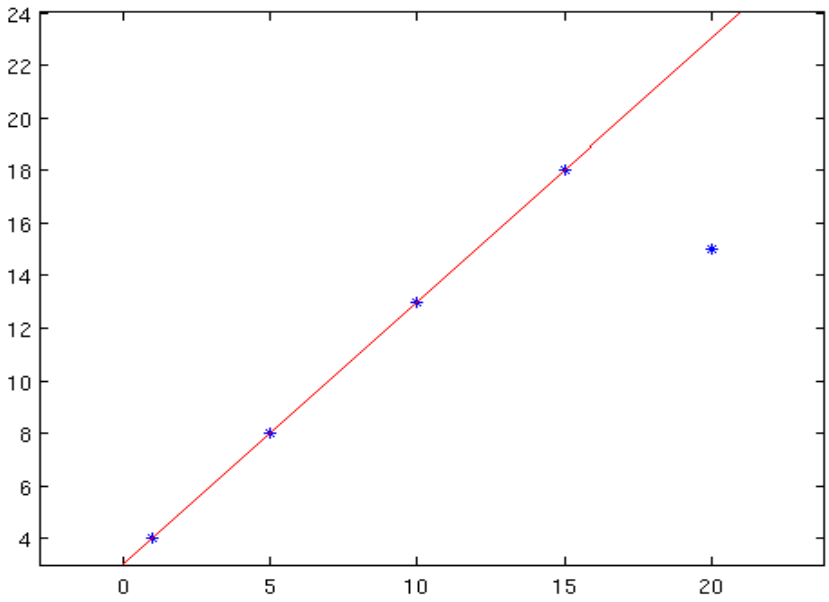


Illustration 7:Plottet viser den linje RANSAC fandt frem til, i forsøget med 5 observationer og 1 grov fejl.

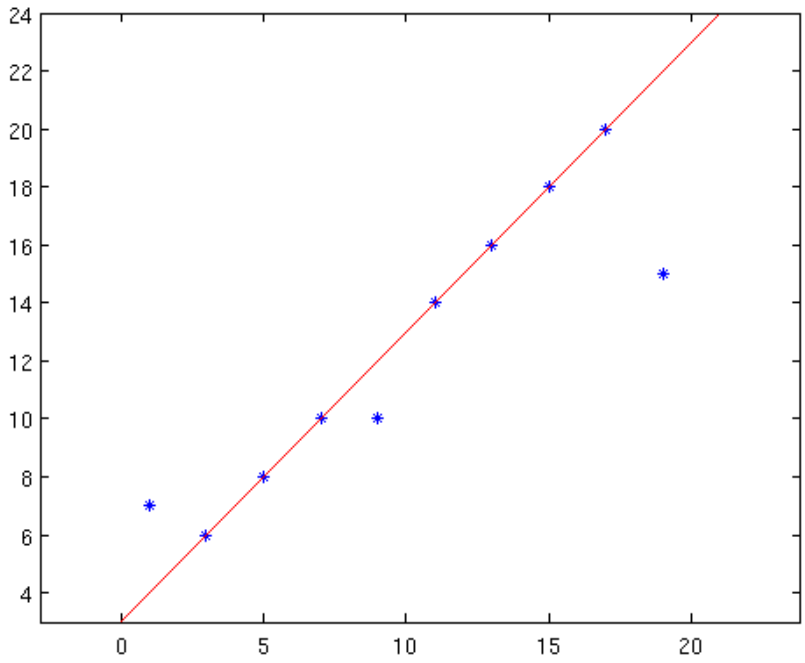


Illustration 8:Plottet viser den linje RANSAC fandt frem til, i forsøget med 10 observationer og 3 grove fejl

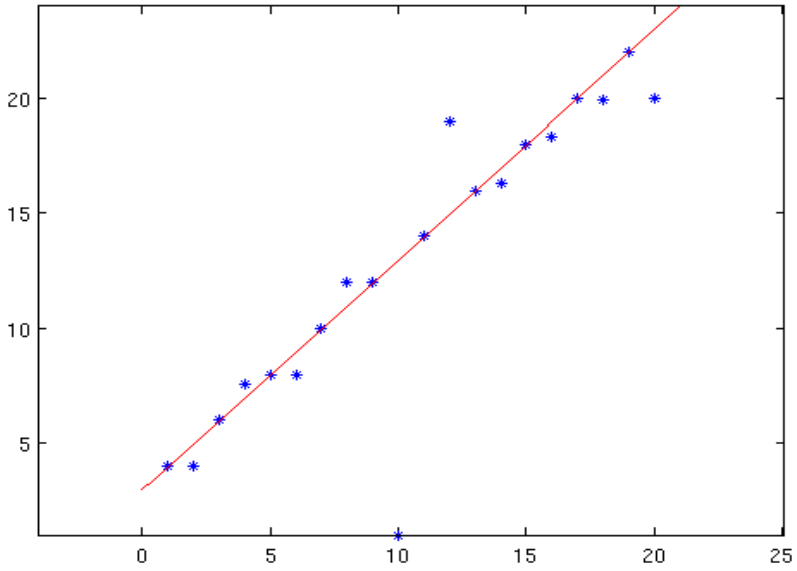


Illustration 9:Plottet viser den linje RANSAC fandt frem til, i forsøget med 20 observationer og 10 grove fejl

10.4 RIL

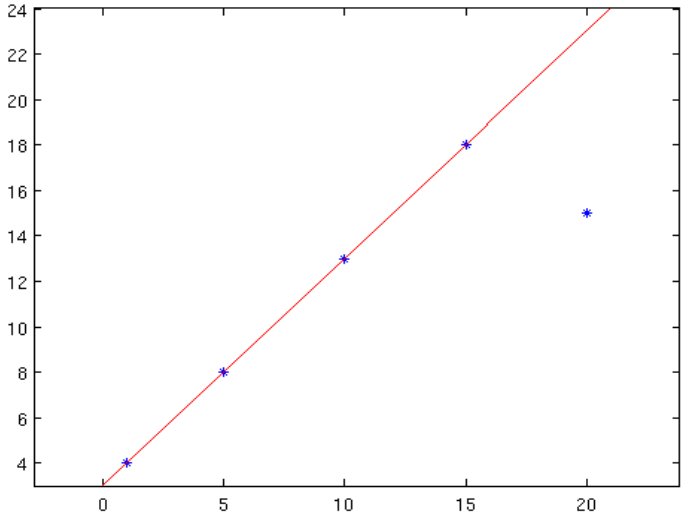


Illustration 10:Plottet viser den linje RIL fandt frem til, i forsøget med 5 observationer og 1 grov fejl

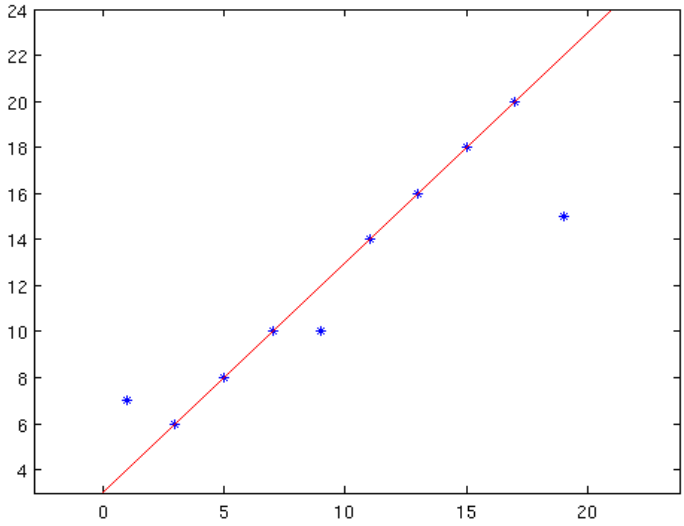


Illustration 11:Plottet viser den linje RIL fandt frem til, i forsøget med 10 observationer og 3 grove fejl

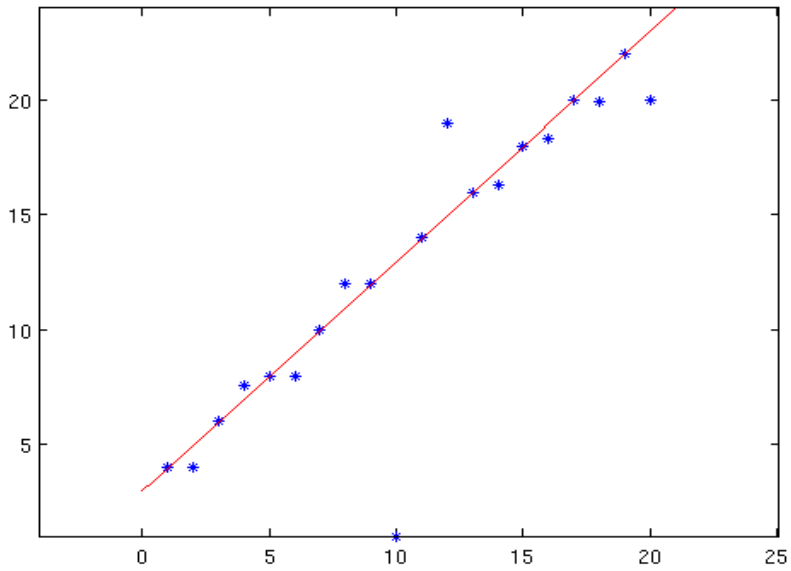


Illustration 12:Plottet viser den linje RIL fandt frem til, i forsøget med 20 observationer og 10 grove fejl

11 Afstandsdata

Her vises første vektoren med de oprindelige 20 observationer, herefter kommer der nogle vektorer, hvor man kan se hvilke grove fejl der er introduceret. Disse vektorer er frem kommet ved at trække vektoren med den grove fejl, fra vektoren uden grove fejl.

```
50.0036
49.9954
49.9957
49.9968
49.9882
50.0058
50.0013
49.9970
50.0055
49.9932
49.9996
49.9990
50.0013
```

```
50.0013
49.9965
49.9999
49.9993
50.0025
50.0044
50.0044
```

11.1 Afstandsdata, 5 observationer, 1 grov fejl

```
    10
     0
     0
     0
     0
```

11.2 Afstandsdata, 10 observationer, 3 grove fejl

```
    10
    -3
     0
     0
     0
     0
   -12
     0
     0
     0
```

11.3 Afstandsdata, 20 observationer, 10 grove fejl

```
   10.0000
   -3.0000
    0.6000
   -4.0000
    1.0000
```

```
   -0.1000
  -12.0000
    6.2000
   -7.5000
    2.0000
         0
         0
         0
         0
         0
         0
         0
         0
         0
         0
```

12 Ret linje data

Her vises først matricen med de oprindelige 20 observationer og et plot af disse punkter, herefter kommer der nogle plots af det punkter, der er indgået i de enkelte forsøg.

```
1 4.0022
2 5.0073
3 5.9910
4 7.0034
5 8.0013
6 8.9948
7 9.9983
8 11.0014
9 12.0143
10 13.0111
11 13.9946
12 15.0121
13 16.0029
14 16.9997
```

```
15 18.0029
16 18.9992
17 19.9995
18 21.0060
19 22.0056
20 23.0057
```

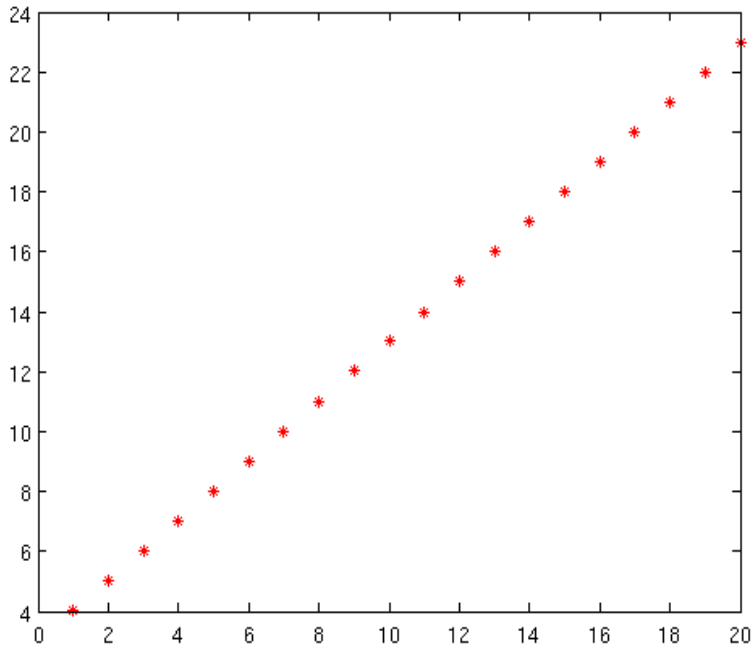


Illustration 13:Plottet viser de oprindelige 20 punkter, uden grove fejl i blandt.

12.1 Retlinje, 5 observationer, 1 grov fejl

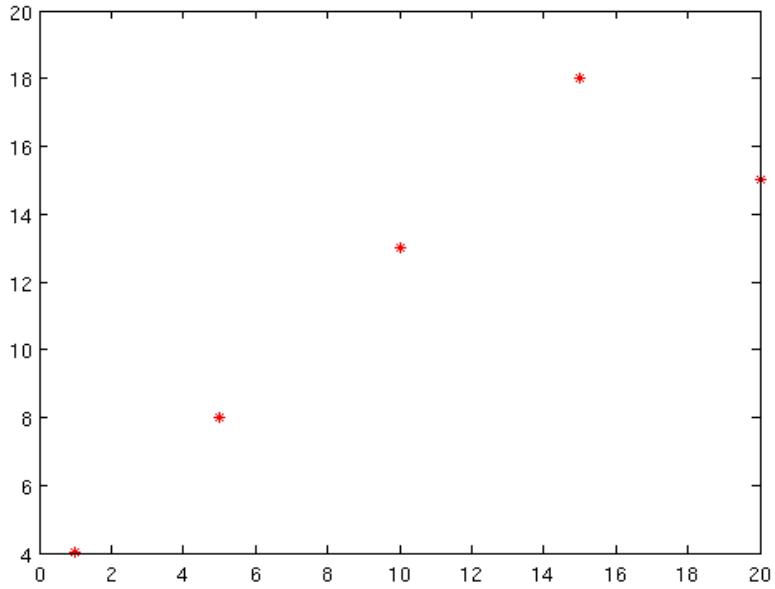


Illustration 14:Plottet viser punkterne, fra forsøget med 5 observationer og 1 grov fejl.

12.2 Ret linje, 10 observationer, 3 grove fejl

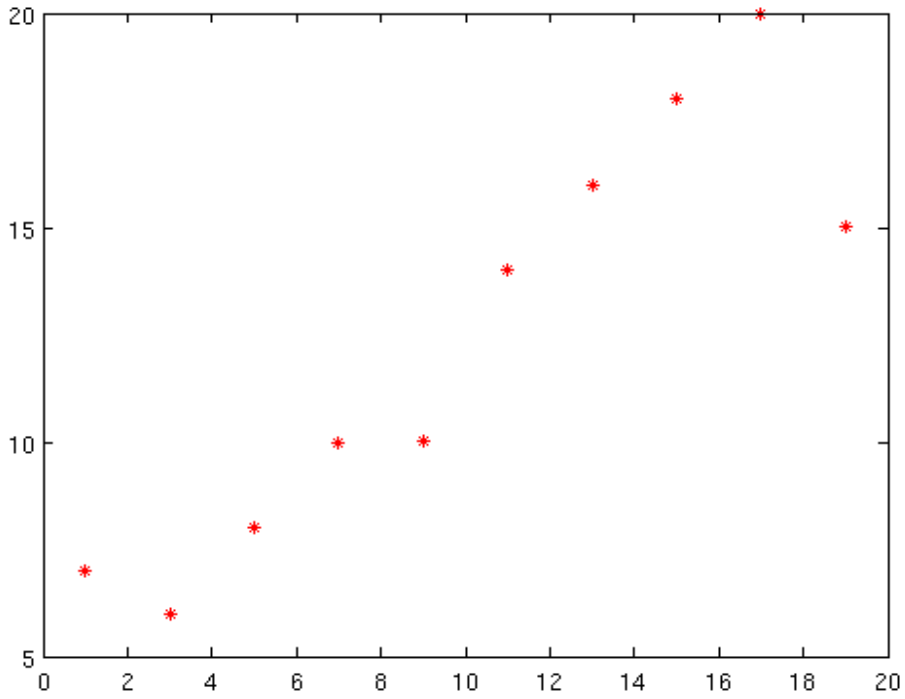


Illustration 15:Plottet viser punkterne, fra forsøget med 10 observationer og 3 grove fejl.

12.3 Ret linje, 20 observationer, 3 grove fejl

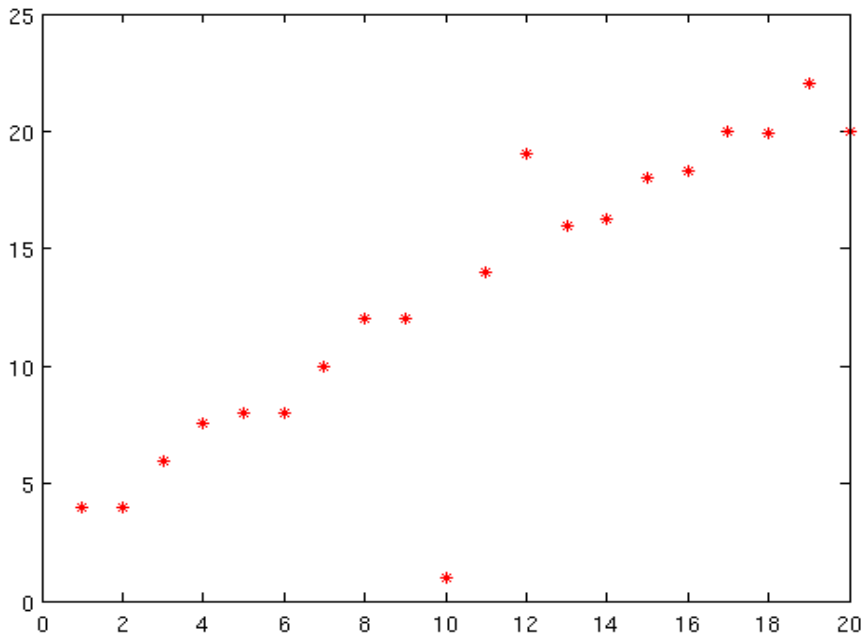


Illustration 16:Plottet viser punkterne, fra forsøget med 20 observationer og 10 grove fejl.

13 Transformationsdata

Her præsenteres nogle plots af de punkter, som transformationerne kører over, først vises et plot af alle 20 punkter, uden grove fejl. Herefter vises plots af de enkelte datasæt efterfulgt af en matrix, som viser hvilke grove fejl der er i datasættet. Denne matrix er fremkommet ved at trække matricen med grove fejl, fra den uden grove fejl. I plottene fra de enkelte forsøg, er de grove fejl markeret med rød.

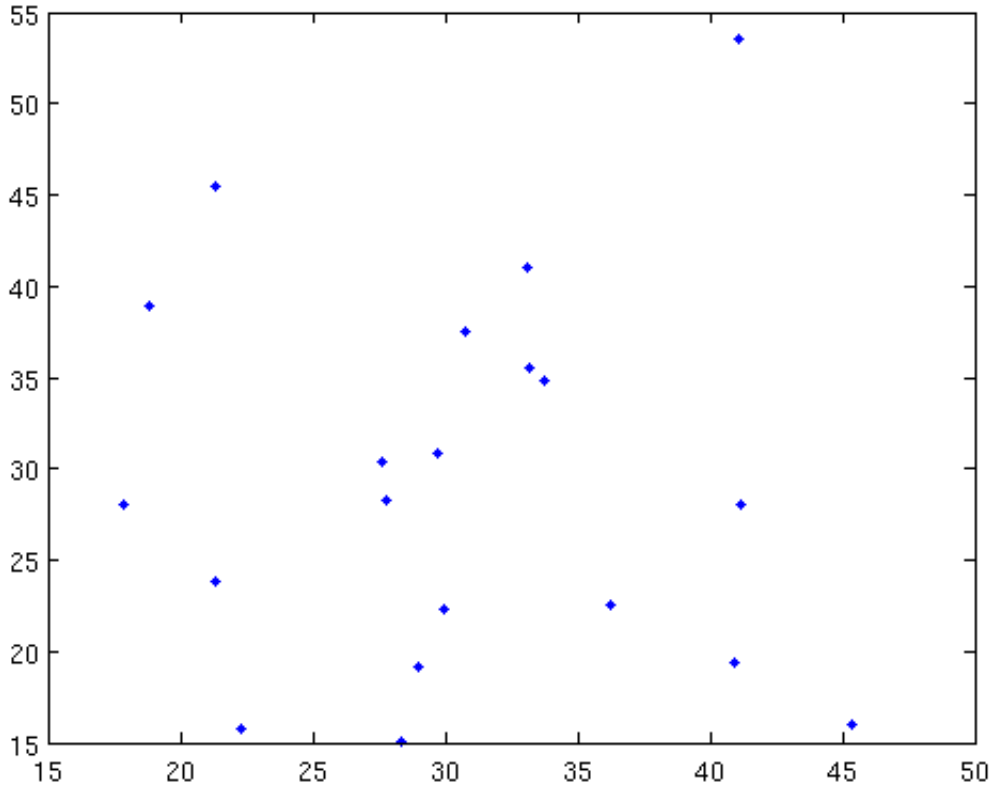


Illustration 17:Plot over de 20 punkter, som indgår i transformationerne.

13.1 Transformationsdata, 5 observationer, 1 grov fejl

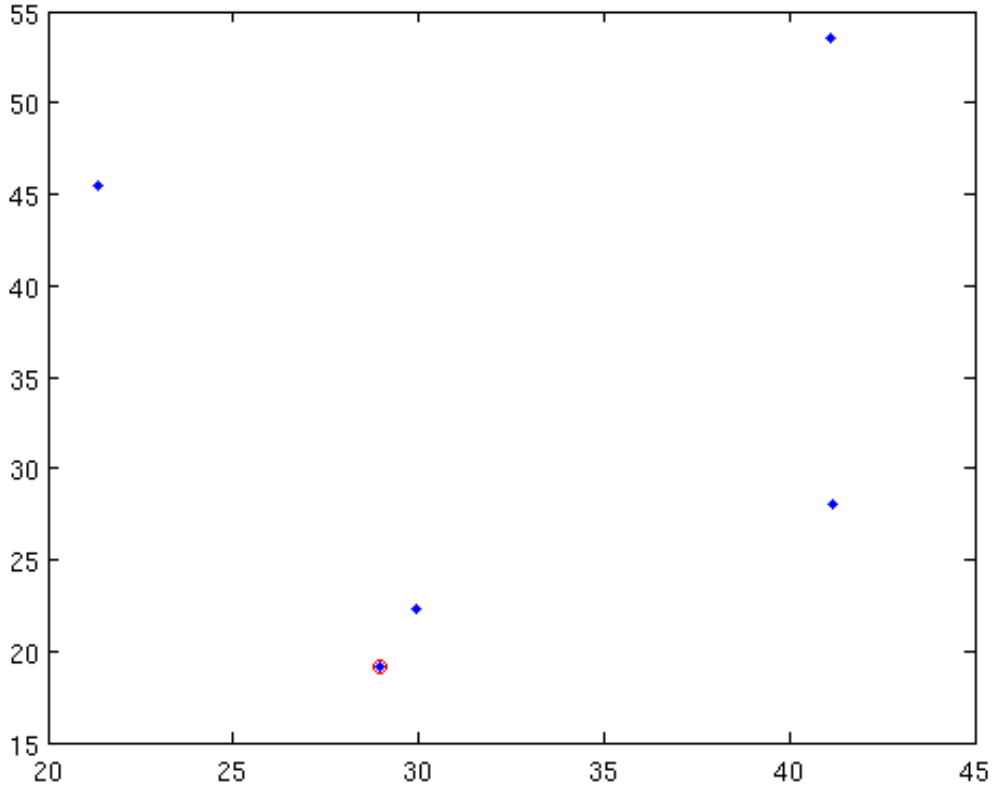


Illustration 18:Plottet viser de 5 punkter, der indgår i dette forsøg.

Fejlmatrix:

0	0	1	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

13.2 Transformationsdata, 10 observationer, 3 grove fejl

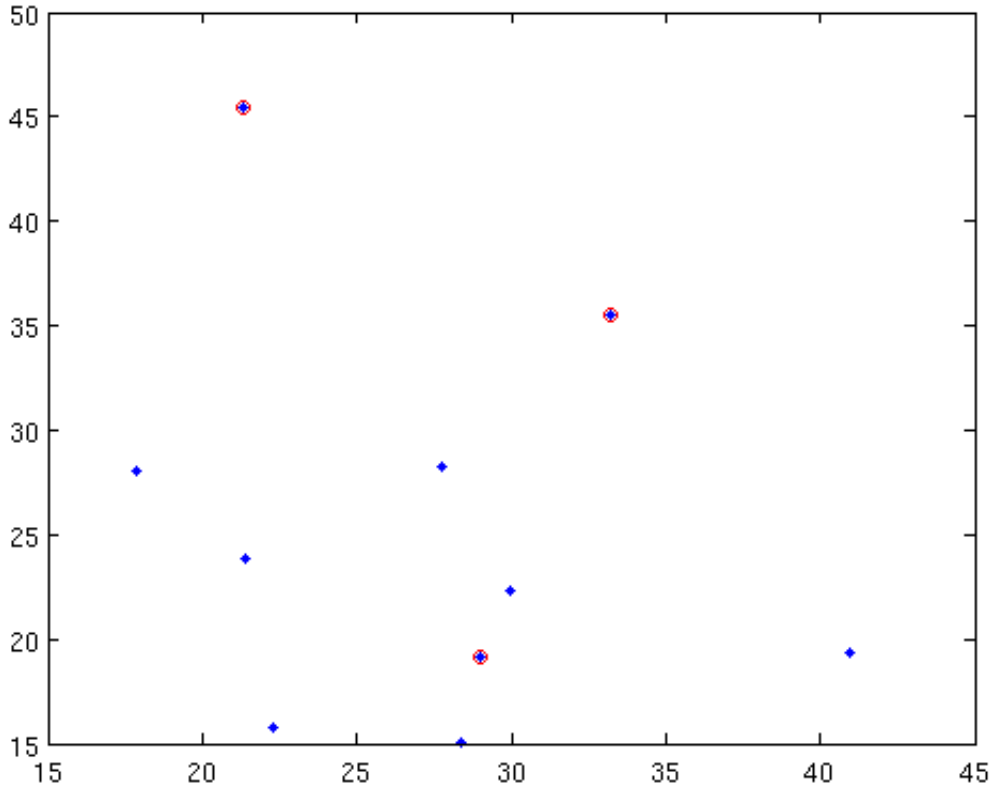


Illustration 19:Plottet viser de 10 punkter, der indgår i dette forsøg.

Fejlmatrix:

0	0	1.0000	0
0	0	12.6082	-5.8640
0	0	-12.6082	5.8640
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

13.3 Transformationsdata, 20 observationer, 10 grove fejl

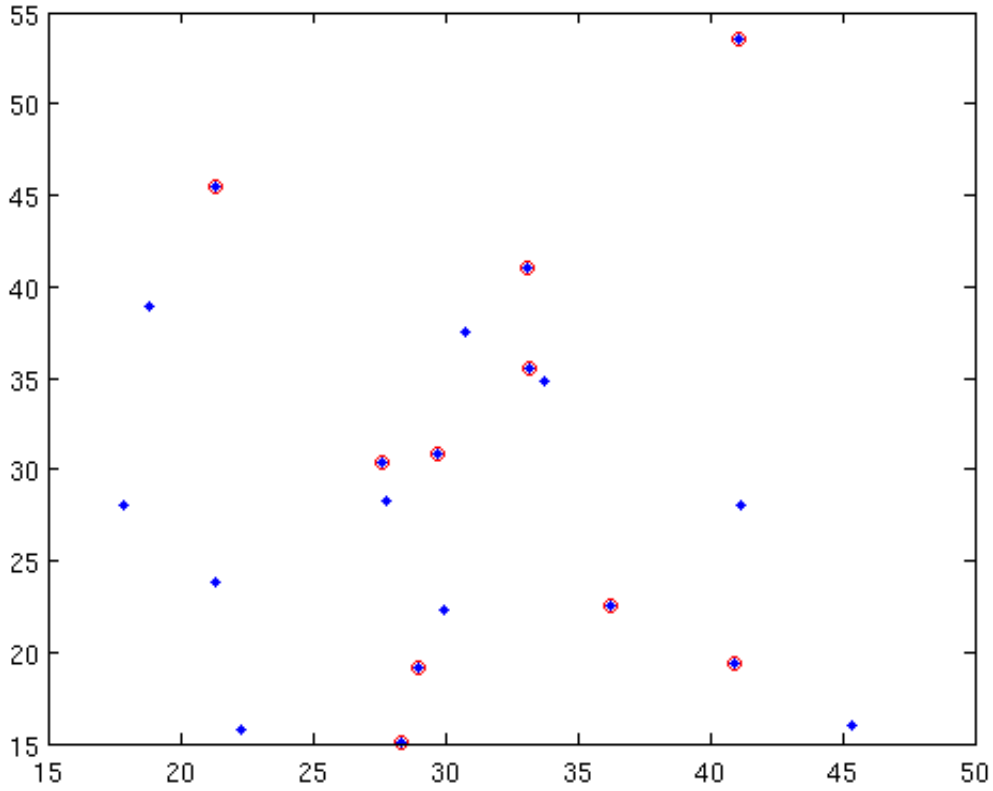


Illustration 20:Plottet viser de 20 punkter, der indgår i dette forsøg.

Fejlmatrix:

0	0	1.0000	0
0	0	-0.0200	0
0	0	12.6082	-5.8640
0	0	0.6210	9.6200
0	0	-12.6082	5.8640
0	0	-0.6210	-9.6200
0	0	0	0.2000
0	0	0	-0.9000
0	0	0	1.0000
0	0	-4.0000	0
0	0	0	0
0	0	0	0
0	0	0	0

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0