UNIVERSITETET I OSLO
Fysisk Institutt

Simulering av kjøling, behov og potensiale ved utstråling

Espen Holtebu

29. januar 2010

Abstract

In this thesis a simulation program originally designed for simulation of heating demand and solar heating potential for builings (SolDat) is developed to also be used for cooling purposes (SolEC). SolDat is capable of weather simulation. Therefore it is focused on developing statistical weather models, which in a better way can generate the weather variables.

Currently SolDat simulates normalized clearness index, ambient temperperature and cold water temperature. For solar heating simulations this is adequate. For simulation of radiative cooling, however, the additional varibles cloud cover, cloud base height and dew point temperature must be introduced. Models for these variables are developed and tested.

In addition, the model for clear weather index is enhanced by an addition of data from solar radiation measurements from 33 weather stations throughout Norway. Using these data, Norway is divided into five climatic zones. This means that the weather can be simulated based on local climate of the location in question.

A model for cooling demand calculation is implemented in SolEC and currently working. Implementation of radiative heat loss for cooling purposes is attempted, but failed due to lack of time.

Forord

Da jeg først ble kjent med energibegrepet i naturfagstimene, ble jeg umiddelbart interessert. Jeg har alltid lurt på hva denne energien er, som kan bevege seg gjennom vegger og får ting til å bevege seg. Energiproblemtatikken i verden vekket etterhvert min oppmerksomhet, sammen med ønsket å bidra til en forandring. Jeg har en stor forkjærlighet til programmering, og at det kunne kombineres i en energioppgave gjorde valget enkelt.

Masterstudiet har vært en veldig lærerik og spennende periode. Det har til tider godt tungt, men de gode venne og kollegene Svetlana har gjort arbeidplassen trivelig. Etter litt diskusjon kom arbeidet raskt igang igjen.

Oppgaven besto opprinnelig av tre deler: Utvikling av simuleringsprogram for beregning av kjøling, teoretisk og eksperimentell beskrivelse av et kjølelager og analyse av et naturlig drevet kjølesystem. Oppgavene var strakk seg over et ganske vidt felt og arbeidsmengen ble stor. Jeg valgte derfor kun å fokusere på simuleringsprogrammet, som fikk endret navn fra SolDat til SolEC (Solar Energy and Cooling).

En stor takk til mine veiledere, professor John B. Rekstad og Dr. Michaela Meir, som gjorde dette prosjektet mulig. Samtidig ønsker jeg å takke Georg M. Meir, for godt samarbeid om utvikling av det grafiske brukergrensesnittet til programmet. Til slutt en takk til familien for støtte og korrekturlesing.

Innhold

Kapittel 1

Introduksjon

Verdens energibehov øker i rask takt sammen med befolkningsveksten og krav til økt levestandard. I 2002 var forbruket nær fordoblet (Rekstad and Meir 2009). Gjennomsnittstemperaturen på jorda har økt de siste 40 årene (CICERO 2009). Temperaturøkningen fører med seg et økende behov for kjøling.

Kjølebehovet blir i dag stort sett dekket av klimaanlegg som benytter elektrisitet. Elektrisiteten er med få unntak produsert av forurensende kraftverk basert på ikke-fornybare energikilder. Som et eksempel går 5% av all energien produsert i USA med til kjøling ved hjelp av klimaanlegg (ACE 2007). Globalt brukes omtrent halvparten av det totale energiproduksjonen til kjøling og oppvarming (Rekstad and Meir 2009).

Kjøling via utstråling til nattehimmelen er et lovende alternativ til konvensjonell kjøling. Strålingskjøling går ut på å benytte den naturlige energioverføringen fra et varmt legeme til omgivelser som holder lavere temperatur. Atmosfæren er transparent for en del av strålingsspekteret, og netto energi som slipper gjennom atmosfæren, kan utnyttes til kjøling ved hjelp av radiatorer. Radiatorene som benyttes, er gjerne de samme som brukes i solfangeranlegg, men er konstruert forskjellig. Anlegget kan dermed både benyttes til kjøling og oppvarming alt etter behov.

Potensialet for solvarme er studert grundig gjennom mange år med forskning. Det er kun de siste tiår at kjøling ved hjelp av naturlige prosesser har fått særlig oppmerksomhet. Naturlige kjøleprosesser er spennende forskningsfelt, og denne oppgaven inneholder i hovedsak beregninger for kjølebehov og kjølepotensial.

Det er tidligere gjort forsøk på å kartlegge potensialet for kjøling via utstråling. Noen av de fremste var fysikerne Martin and Berdahl (1984). De utviklet modeller

for den atmosfæriske emittansen som funksjon av duggpunktstemperatur, skydekke og skyhøyde. I tillegg så de på kjølepotensialet ved hjelp av værdata fra 193 forskjellige værstasjoner i USA. Det er også utført beregninger for kjølepotensialet i Europa (Argiriou et al. 1993) med utgangspunkt i atmosfæremodellene utviklet av Berdahl and Martin (1984) og Berdahl and Fromberg (1982).

Gruppa for Energifysikk på Blindern har gjort en rekke forsøk med solfangere som radiatorer. Storås (1997) gjorde de første forsøkene, og fant at utstrålingen mot en klar nattehimmel fra en flate med omgivelsestemperatur kunne bli ca. 50 Wm^{-2}. Meir et al. (2002) videreførte dette arbeidet med store stråleflater. Degnes-Ødemark (2009) har utført pyrgeometermålinger gjennom et helt år for å kartlegge kjølepotensialet i Oslo.

I denne oppgaven vil jeg undersøke om kjølepotensialet ved utstråling kan beregnes uten å foreta lokale målinger og forsøk. Til dette formålet må det utvikles et simuleringsprogram som kan beregne kjølebehov og kjølepotensial ut fra ulike klimabetingelser. I dag er et slikt program ikke tilgjengelig. Avanserte programmer som TRNSYS (T. E. S. S. 1975), kan i prinsippet også beregne kjøling, men en systematisk behandling av de faktorer som bestemmer kjølepotensialet er ikke dokumentert.

Solenergigruppa har utviklet flere programmer for å beregne energiutbyttet fra solvarmeanlegg. Disse ble videreutviklet og et samlet program ble ferdig juni 1991 (Ingebretsen 1991). Programmet ble skrevet i Turbo Pascal versjon 5.5 og var kommandolinjebasert. For å gjøre det mer brukervennlig ble det skrevet om til Java (Haugen 2000). Her ble også nye og mer avanserte modeller som baserte seg på et større datagrunnlag utviklet og implementert. Programmet kalles i dag SolDat, og beregner utetemperatur og andre værparametere etter flere statistiske modeller.

Målet for denne oppgaven har vært å utvikle et komplett program som samtidig kan beregne oppvarmingsbehov, kjølebehov, solenergipotensial og kjølepotensial ved utstråling til nattehimmelen. Værforhold og utetemperatur er hovedfaktorene som styrer både kjølebehov og kjølepotensial. Utvikling av modeller som kan benyttes i programmet for beregning av atmosfæriske variable er en sentral del av oppgaven. Innstrålingsmålinger fra store deler av Norge er undersøkt for å skaffe et større statistisk grunnlag for modellene brukt i simulering av værvarable. Samtidig er det forsøkt å implementere disse i modellene og foreta enkle simuleringer. Programmet er tilpasset moderne anlegg for kombinert soloppvarming og

strålingskjøling, og kan komme til å bli et viktig verktøy når den beste løsningen for termisk komfort skal beregnes.

Kapittel 2

Bakgrunn

Det er ikke funnet enkle og gode simuleringsprogrammer for beregning av kjølebehov og kjølepotensial, men flere mer avanserte programmer eksisterer. Denne oppgaven tar sikte på å utvikle et allerede eksisterende simuleringsprogram beregnet for oppvarming basert på solfangere. Ved å endre og legge til kildekode kan programmet også benyttes for simulering av kjølepotensiale. Dette ved å simulere bruk av teknologi som baserer seg på kjøling via utstråling.

Hovedgrunnen til at det er ønskelig å simulere energipotensialet for solenergisystemer er for å kunne dimensjonere systemer riktig etter klimatiske forhold. Dette er fordi det ofte er en misforhold mellom tilgang på energi og behov. Det er store døgnlige og sesongmessige variasjoner i energitilførselen, og det stiller krav til lagringsmedier, slik som termiske varmelagre og kuldelagre. Simulering av energibehov og potensial muliggjør systemdesign spesialtilpasset en bestemt bolig eller et bygg.

Godt og dårlig vær kommer i perioder, og periodene er i variabel lengde. Grensen mellom værtypene er definert i seksjon 4.3.1. Behov og potensial avhenger av lengden på værperiodene. Det finnes generelt tre ulike scenarioer som kan oppstå når energitilgangen er avhengig av værforhold:

- Ved lange godværsperioder er det god tilgjengelighet på energi så lenge energilageret er designet for å holde på den termiske energien gjennom døgnet. Det gjelder både solenergi og utstrålingsenergi. I dette tilfellet trenger ikke lagringsmediet ha stor kapasitet. Energitapet vil også ha mindre betydning da energien er tilgjengelig ved behov.
- Er været svært skiftende og periodelengdene korte, vil det stilles større krav

til energilageret. For å øke utnyttelsesgraden av fornybar energi bør lageret kunne holde på energien så lenge som mulig, og helst fram til neste godværsperiode.

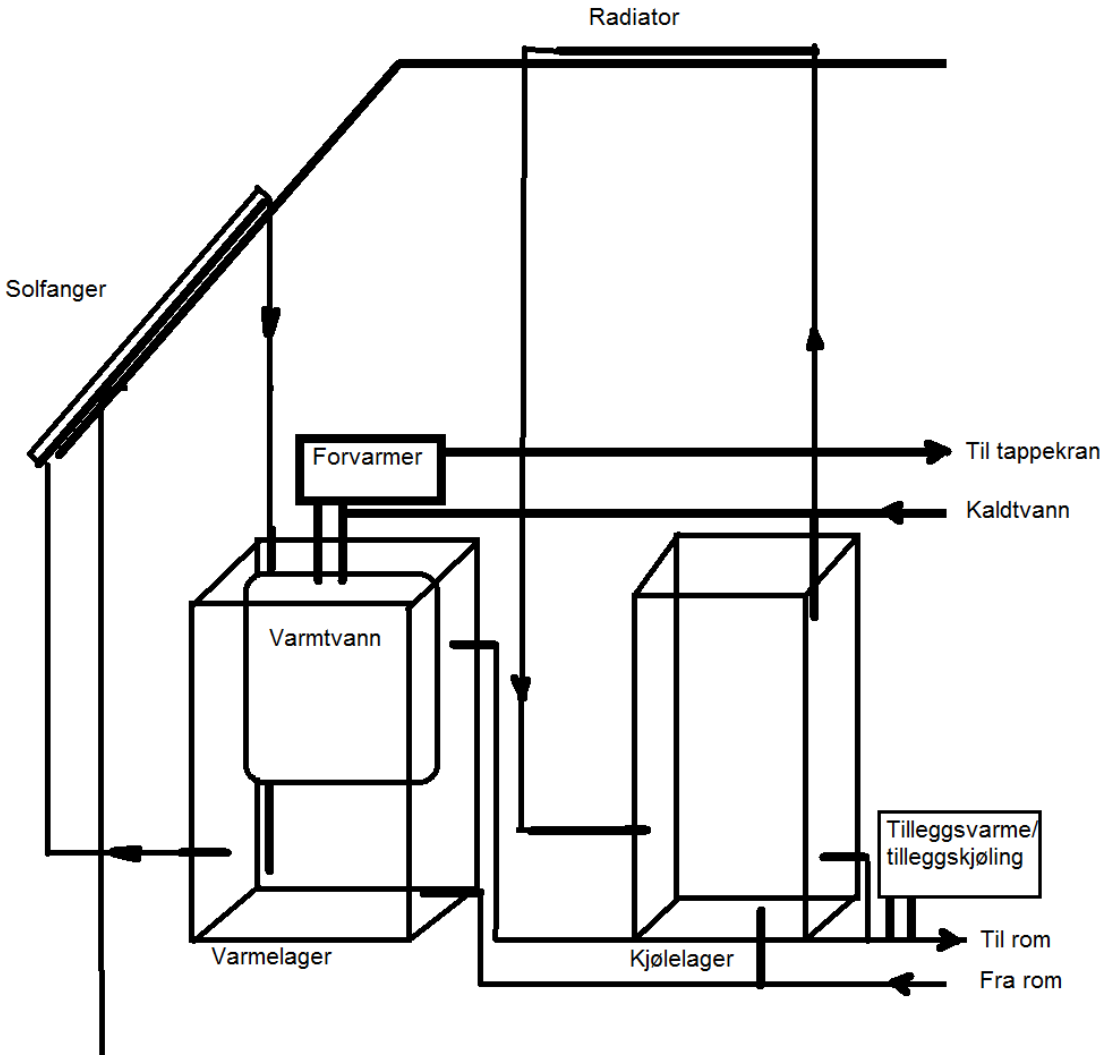
- Ved lange og dårlige værperioder er været stort sett overskyet, og tilgangen på solenergi være knapp og lite forutsigbar. I en slik situasjon er man avhengig av andre energikilder. Når varmelageret holder samme temperatur som omgivelsene, er maksimalt kjølepotensial oppnådd, og det vil ikke forekomme netto strålingstap.

Det er vanlig å benytte typiske meteorologiske år (TMY) ved solenergisimuleringer i TRNSYS. Dette er historiske data basert på målinger et visst antall år. For bestemte geografiske områder er det dermed nødvendig med kjennskap til værstatistikk, noe som ikke alltid er tilgjengelig. En annen begrensning ved bruk av værdata er at hver eneste simulering benytter nøyaktig det samme været. Det kan være nyttig når man ønsker å dimensjonere et anlegg, men årlige variasjoner blir ikke synlige. Ved eksplisitt å simulere været er det kun nødvendig å kjenne til gjennomsnittstemperatur og årlige klarværsprofiler. På denne måten kan en få en idé om hvordan været endrer seg fra år til år og se hvordan systemet vil fungere ved skiftende klima.

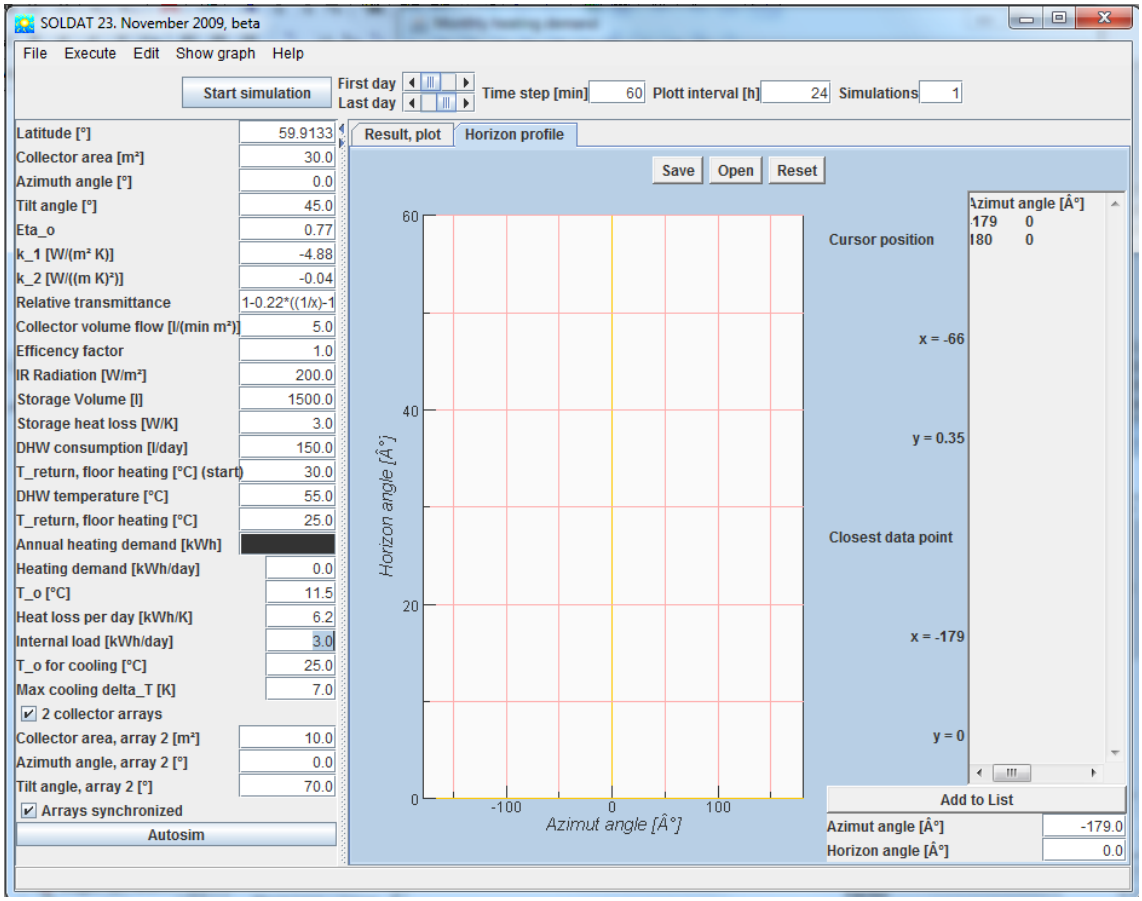
2.1 Simuleringsprogrammer

Det er funnet flere programmer for simulering av kjøling og oppvarming av bygninger. Allerede i 1974 ble det utviklet et program for simulering av soloppvarming og kjøling av bygninger (Winn et al. 1974). Det kalles SIMSHAC (et akronym for Simulation Model for Solar-Heated-And-Cooled buildings), og er modulbasert. Det vil si at brukeren kan sette sammen systemet slik han/hun ønsker selv. Kombinasjoner av flatplatekollektorer, varmtvannslager, pumper, varmevekslere, rør osv. kan settes sammen.

EnergyPlus (Crawley, Lawrie, Pedersen, and Winkelmann 2000), er et modulbasert simuleringprogram. Det baserer seg på å koble inn flere moduler, deriblant moduler for strålingsbasert kjøling.



Figur 2.1: SolDat baserer seg på en enkel kombianlegg der varmtvannberederen befinner seg inne i varmelageret. Solfangere forsyner varmelageret med varme. I nytt program er et enkelt kjølelager og radiatorer introdusert.



Figur 2.2: Skjermbilde av SolDat med tilhørende behovsprofiler.

2.2 Prinsippet som SolDat bygger på

SolDat utfører beregninger for et anlegg som leverer både varmtvann og romvarme, et såkalt kombianlegg. Varmelageret og varmtvannsberederen er kombinert i en tank, der varmtvann produseres i en innertank. Solfangere er tilknyttet varmelageret med en enkel rørforbindelse der varmetap fra rør er inkludert i solfangereffektiviteten. Prinsippet er vist i figur 2.1.

Varmebehovet blir beregnet ut fra utetemperaturen eller fra årlige behovsprofiler. Døgnlig forbruk av varmtvann må oppgis. Et skjermbilde av programmet, med behovsprofiler, er vist i figur 2.2.

2.3 Stråling

Alle legemer med en temperatur $T > 0$ K emitterer en viss mengde stråling R gitt i Wm^{-2}, og følger Stefan-Boltzmanns lov

$$R = \sigma \epsilon T^4 \tag{2.1}$$

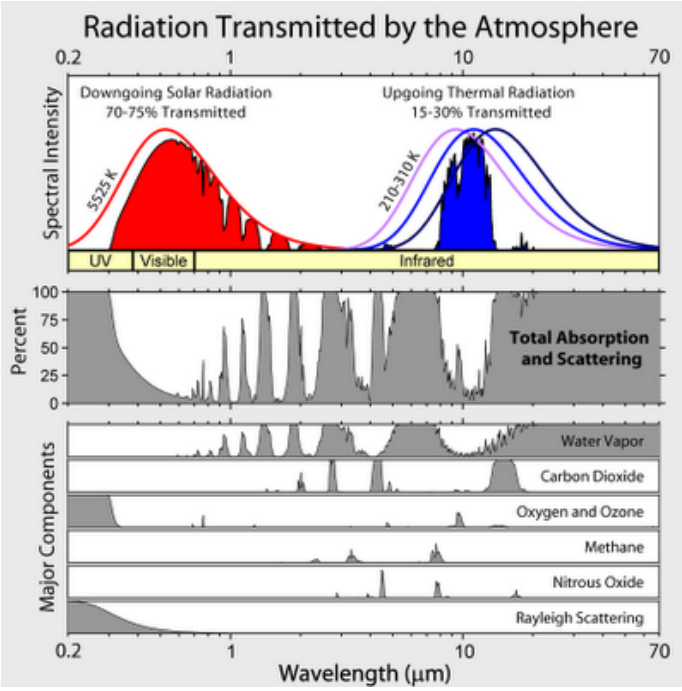
hvor $\sigma = 5{,}670400 \times 10^{-8}\ \mathrm{Wm}^{-2}\mathrm{K}^{-4}$ er Stefan-Boltzmanns konstant og ϵ er emittansen til legemet. Emittansen er en størrelse som sier hvor mye stråling legemet emitterer i forhold til et ideelt sort legeme.

2.4 Atmosfæren

Atmosfæren som befinner seg i lag rundt jorda består av flere forskjellige gasser. Gassene, som har innvirkning på transmittansen til solinnstrålingen og den termiske utstrålingen, er vist i figur 2.3. Atmosfæren består av 78,1% nitrogen og 20,9 % oksygen, og er gjennomsiktig for den delen av solspekteret som har høyest intensitet. I gjennomsnitt vil omkring 30% av ekstraterrestriell solinnstråling bli reflektert av atmosfæren. Solinnstrålingen som treffer atmosfæren er stort sett direkte. All stråling kommer fra samme punkt og samme retning dersom Sola blir sett på som en punktkilde. Atmosfæren vil transmittere omtrent 70% av solinnstrålingen, resten blir reflektert. Av den transmitterte innstrålingen vil atmosfæren spre en andel i alle retninger. Dette kalles diffus stråling.

Utstråling fra legemer følger et strålingsspekter avhengig av temperaturen til legemet. Den spektrale intensiteten til to legemer, sola og jorda, er vist øverst på figur 2.3. Figuren viser at solspekteret befinner seg i et bølgelengdeområde omtrent to størrelsesordener lavere enn jordas spekter.

Termisk, infrarød stråling karakteriserer strålingen fra legemer med temperaturer på nivå med jordoverflaten. I det karakteristiske bølgelengdeområdet finnes det atmosfæriske vinduet, der infrarød utstråling transmitteres. I dette området av spekteret er det få aktive gasser. Karbondioksid og vanndamp er de to viktigste drivhusgassene. Det betyr at de har høy termisk absoptans. Disse gassene er aktive i det atmosfæriske vinduet som vist i figur 2.3. Det atmosfæriske vinduet svekkes på grunn av økt CO_2-innhold i atmosfæren. I 2007 var CO_2-konsentrasjonen økt til 0,038% (Agency 2007). Termisk stråling som unnslipper atmosfæren er også



Figur 2.3: Transmittert og absorbert stråling i atmosfæren. Stråling absorbert av forskjellige drivhusgasser er også vist. Kilde: (Rhode 2007)

i stor grad påvirket av skydekket, som stort sett består av mikroskopske vanndråper, i tillegg til støvpartikler.

2.4.1 Klarhetsindeks og klarværsindeks

Klarhetsindeksen er definert som andel solinnstråling som treffer en horisontal flate mot den samme innstrålingen dersom atmosfæren ikke hadde eksistert. Momentan klarhetsindeks, k, er definert ved

$$k = \frac{G}{G_o} \tag{2.2}$$

hvor G_o er ekstraterrestriell innstråling og G er innstråling ved jordoverflaten, begge mot horisontal flate og gitt i Wm^{-2}. Klarhetsindeksen forteller sier noe om klar atmosfæren er, og avhenger av veilengden gjennom atmosfæren og skydekket. Ved lave solhøyder vil solinnstrålingen ha lengre vei å gå gjennom atmosfæren og bli tilsvarende redusert. Ved fullstendig skydekke vil innstrålingen som slipper gjennom atmosfæren være minimal og $k \to 0$. Er himmelen skyfri vil $k \to 0{,}74$ ved store solhøyder. Klarhetsindeksen vil, på grunn av refleksjon i atmosfæren, aldri nå verdier i nærheten av 1 under normale forhold på Jorda.

Normalisert klarhetsindeks eller såkalt klarværsindeks er andelen global innstråling mot horisontal flate mot tilsvarende global innstråling ved skyfri himmel. Klarværsindeksen er i motsetning til klarhetsindeksen uavhengig av solhøyden og beregnes gjerne empirisk da modeller for klarværsinnstråling ikke er gode nok til å gi riktig verdi for klarværsinnstråling. Når solinnstråling ved klarvær er kjent kan klarværsindeksen, k_c, beregnes som

$$k_c = \frac{G}{G_c} \tag{2.3}$$

hvor G_c er global solinnstråling ved klarvær. Klarværsindeksen følger et av-på mønster i større grad enn klarhetsindeksen. Enter skinner Sola, ellers er den bak en sky, og det påvirker verdien i stor grad.

Ved sammenligning av likn. (2.3) og (2.2) kan transmittansen for atmosfæren finnes som en funksjon av klarhetsindeks og klarværsindeks. Transmittansen, τ_c, til atmosfæren er da gitt ved

$$\tau_c = \frac{G_c}{G_o} = \frac{G_c}{G_{on} sin\theta_z} \tag{2.4}$$

der θ_z er senitvinkelen definert i seksjon 2.5 og G_{on} er ekstraterrestriell fluks normalt på solinnstrålingsretningen. τc benyttes for beregne solinnstrålingen som transmitteres ned til jordoverflaten. Det er nødvendig å kjenne transmittansen når komponentene diffus, direkte og reflektert solinnstråling skal beregnes.

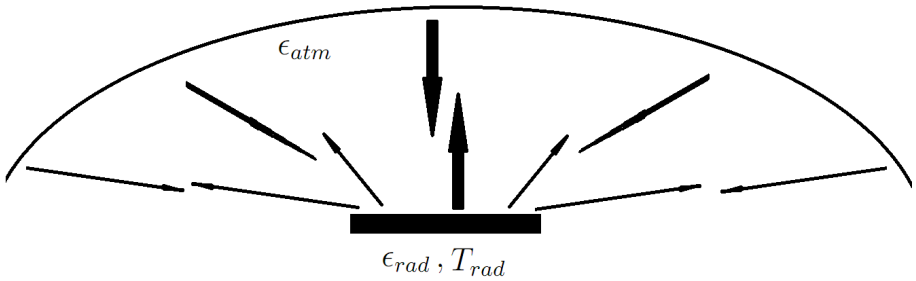
Ved innsetting av lign. (2.2) og (2.3) finnes uttrykket

$$\tau_c = \frac{k}{k_c} \tag{2.5}$$

Av Haugen (2000) er det funnet et empirisk uttrykk for transmittansen gjennom atmosfæren ved klarvær. Denne modellen er gitt ved

$$\tau_c = \begin{cases} 0{,}35 & \cos\theta_z < 0{,}0798 \\ 0{,}74 - 0{,}65e^{-6{,}4\cos\theta_z} & \cos\theta_z > 0{,}0798 \end{cases} \tag{2.6}$$

Ved store senitvinkler, dvs. $\cos\theta_z < 0,0798$ vil ikke $\tau_c \to 0$ slik det ville vært forventet dersom all stråling var direkte. Diffusandelen vil dominere totalt i dette tilfellet og transmittansen varierer mellom omtrent 0,1 og 0,5. Det er derfor valgt å bruke gjennomsnittsverdien $\tau_c = 0,35$ for store senitvinkler.



Figur 2.4: Strålingsbalanse for en horisontal radiator med synsfelt mot atmosfæren. Radiatoren vil ha et strålingstap så lenge radiatortemperaturen er høyere enn himmeltemperaturen. Kilde: egen tegning.

2.4.2 Strålingstransport

Jordoverflaten emitterer termisk stråling ut mot atmosfæren. Også atmosfæren som omgir jorda har en endelig temperatur og emitterer termisk stråling. Strålingen foregår i to retninger. En del av den atmosfæriske utstråling forsvinner ut i universet, mens en nesten like stor andel stråler ned mot jordoverflaten. På grunn av det atmosfæriske vinduet er ikke atmosfæren et ideelt sort legeme.

Det er to måter å se atmosfæren på. Dersom den anses som et sort legme kan det defineres en ekvivalent atmosfærisk temperatur, T_{atm}. Atmosfærisk innstråling, R_{atm}, er da på formen

$$R_{atm} = \sigma T_{atm}^4 \tag{2.7}$$

Et annet alternativ er gi atmosfæren en temperatur lik utetemperaturen T_a. I dette tilfellet er atmosfære gitt en emittans, ϵ_{atm}, som gir innstrålingen

$$R_{atm} = \sigma \epsilon_{atm} T_a^4 \tag{2.8}$$

Jordverflaten mottar altså termisk stråling fra atmosfæren i tillegg til solinnstrållingen. Men siden jordoverflaten normalt holder høyere temperatur enn T_{atm}, er utstråling herfra større enn den termiske strålingen den mottar fra atmosfæren. To legmer som er i termisk likevekt vil ha strålingsbalanse. Dette er bakgrunnen for Kirchhoffs lov, som sier at emittansen til et legme vil være lik absorptansen til legemet. Varmetapet P_{rad}, ved utstråling for en radiator er gitt ved differansen mellom utgående og innkommende stråling, og kan ved Kirchhoffs lov skrives som

$$P_{rad} = A_{rad} \epsilon_{rad} \left(\sigma T_{rad} - R_{atm} \right) \tag{2.9}$$

hvor ϵ_{rad} er flatens emittans, og A_{rad} er radiatorens areal i m^2. For en horisontal flate er R_{inn} atmosfærisk innstråling $R_{atm} = \sigma\epsilon_{atm}T_a^4$, hvor ϵ_{atm} er atmosfærisk emittans over hemisfæren der atmosfæren har en temperatur lik utetemperaturen T_a. Utgående stråling er gitt ved Stefan-Boltzmanns lov hvor T_{rad} er temperaturen til flaten. For et panel er den gitt ved paneltemperaturen som igjen er en funksjon av temperaturen i varme/kjølelageret. En andel av denne strålingen vil treffe og absorberes av skyggende objekter og bakken, hvor det er antatt at absorptansen til bakken og objektene er like. Strålingsbalansen for en horisontal flate som kun har synsfelt mot atmosfæren er illustrert i figur 2.4.

For skyfri himmel er det funnet (Berdahl and Fromberg 1982) et uttrykk for atmosfærisk emittans ved klarvær, ϵ_0, som er utviklet for klimaet i USA:

$$\epsilon_0 = 0{,}711 + 0{,}0056T_d + 0{,}000073T_d^2 + 0{,}013\cos\left(\frac{2\pi t_s}{24}\right) \tag{2.10}$$

Her er t_s antall timer fra midnatt og T_d duggpunktstemperaturen i °C.

I ettertid er det funnet en alternativ metode for å bestemme atmosfærisk emittans ved skyfri himmel (Øystein Godøy 2004):

$$\begin{aligned} \epsilon_0 =& 1 - (1-\xi)e^{-\sqrt{1{,}2+3{,}0\xi}} - 0{,}05\frac{p_0 - p}{p_0 - 710} \\ &\text{hvor } \xi = c\frac{e_0}{T_a} \end{aligned} \tag{2.11}$$

og p er lokalt atmosfærisk trykk, $p_0 = 1013{,}15$ hPa, e_0 er vanndamptrykk ved bakkenivå og $c = 46{,}5$ cmK hPa^{-1}. Denne relasjonen er beregnet for et større geografisk område enn likn. (2.10) og kan da synes å gjelde mer generelt. Det er likevel valgt å benytte likn. (2.10) i simuleringene, da den kun avhenger av duggpunktstemperaturen, som er lettere tilgjengelig.

Atmosfærisk emittans (og absorptans) vil påvirkes av skydekke og skyhøyde. Skyer kan ha en stor innvirkning på hvor mye termisk stråling som slipper ut gjennom atmosfæren. I tillegg har skytemperaturen mye å si for hvor mye termisk stråling som emitteres ned mot jordoverflaten.

Ved overskyet vær har (Martin and Berdahl 1984) funnet en empirisk justering av klarværsemittansen. Denne er gitt på formen

$$\epsilon_{atm} = \epsilon_0 + (1-\epsilon_0)\epsilon_c n e^{-\frac{z_h}{z_*}} \tag{2.12}$$

Skydekket har en viss tykkelse. z_c er høyden til skydekkets basisnivå i km normalt på jordoverflaten (senit), $z_* = 8{,}2$ km, ϵ_c er den hemisfæriske emittansen satt lik 1 for $z_h < 4$ km, n er hemisfærisk andel skydekke.

I (Olseth and Skartveit 1994) er det gjennomgått 23 modeller for ϵ_{atm} og de fant at likn. (2.12) gir best resultater. (Degnes-Ødemark 2009) har benyttet samme likn. for beregning av atmosfærisk temperatur og den er stort sett funnet å ligge innenfor ± 3 K for Oslo.

I (Sugita and Brutsaert 1993) er det presentert en relasjon for atmosfærisk innstråling som funksjon av ϵ_0 og k. Hemisfærisk emittans er funnet fra denne relasjonen ved Stefan-Boltzmanns lov ved å dividere med σT_a^4. Den blir da på formen

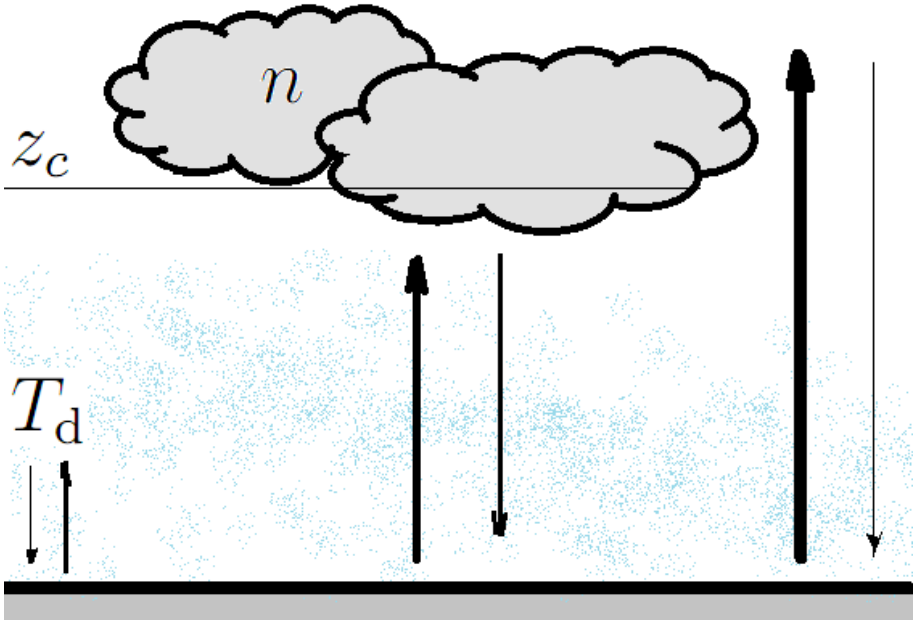
$$\epsilon_{atm} = 1{,}02\epsilon_0 k^{-0{,}0227} \tag{2.13}$$

En annen modell for atmosfærisk emittans ved skydekke er funnet i (Øystein Godøy 2004) og er basert på skydata fra sattelittbilder.

$$\begin{aligned} \epsilon_{atm} &= \epsilon_0 + (1 - \epsilon_0)C \\ \text{der } C &= \sum n_i C_i \end{aligned} \tag{2.14}$$

I likn. (2.14) er n_i andel skydekke av skytype i og C_i bidraget av skytype i. C_i kan beregnes ved klarværsindeks når skytypen er kjent. Simulering av skytyper er ansett som for avansert for simuleringsprogrammet.

Likn. (2.12) er brukt for å finne den totale atmosfæriske emittansen i programmet. Relasjonen avhenger av varible som påvirker den atmosfæriske innstrålingen i ulik grad, og påvirkningen er illustrert i figur 2.5. Høy duggpunktstemperatur (se seksjon 2.7) medfører at en stor andel av utstrålingen blir absorbert lavt i atmosfæren. Økende skydekke n gir høyere atmosfærisk innstråling. Skydekkets påvirkning avtar med skyhøyden z_h, siden skyenes temperatur avtar med høyden. Dersom det er skyfritt og T_d er lav, vil en stor andel av utstrålingen fra en flate på jordoverflaten transmitteres gjennom atmosfæren.



Figur 2.5: Strålingstransport er avhengig av variablene duggpunktstemperatur T_d, skydekke n og høyden til skydekkets nedre del (Kilde: egen tegning).

2.5 Definisjon av vinkler og tid

Figur 2.6 viser alle vinklene som er relevante for innstrålingen mot en flate. For å finne innfallsvinkelen mellom solinnstrålingen og normalen til en flate, samt synsfeltet mot atmosfæren, er det nødvendig med kjennskap til disse vinklene. Alle vinkler er her gitt i grader.

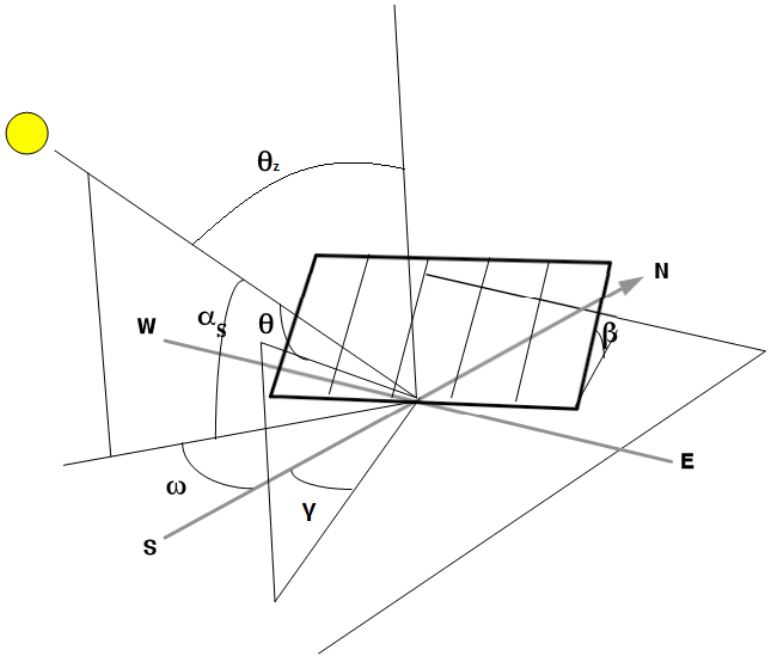
En flate plassert på jordoverflaten er definert ved fire vinkler. For det første har den en posisjon gitt ved breddegrad ϕ og lengdegrad L. I tillegg kan den ha en hellingsvinkel β og en asimutvinkel γ. Asimutvinkelen beskriver flatens kompassretning. Øst er negativ og vest positiv.

Deklinasjonen δ tilsvarer solhøyden idet Sola passer lokalmeridianen, det vil si når Sola står på sitt høyeste gjennom dagen. Deklinasjonen δ varierer med dagen i året og er gitt på formen

$$\begin{aligned}\delta = \frac{180}{\pi}(&0{,}006918 - 0{,}399912\cos B + 0{,}070257\sin B - \\ &0{,}006758\cos 2B + 0{,}000907\sin 2B - \\ &0{,}002697\cos 3B + 0{,}00148\sin 3B \\ \text{hvor } B = &(N-1)\frac{360}{365}\end{aligned} \tag{2.15}$$

der viN er dagen i året. En enklere variant er på formen

$$\delta = 23{,}45 \sin\left(360 \frac{284 + N}{365}\right) \tag{2.16}$$



Figur 2.6: Innfallvinkel for et plan med hellingsvinkel β og asimutvinkel γ. Redigert utgave fra (Rekstad and Meir 2009)

Soltid og lokaltid sammenfaller sjelden, og i tilfelle kun noen få ganger i året. Tiden da Sola er på sitt høyeste varier gjennom året og er beskrevet av tidsligningen E:

$$\begin{aligned} E = 229{,}2(&0{,}000075 + 0{,}001868 \cos B - 0{,}032077 \sin B \\ &- 0{,}014615 \cos 2B - 0{,}04089 \sin 2B) \end{aligned} \tag{2.17}$$

Når E er funnet er soltiden, t_s, gitt i timer etter midnatt

$$t_s = t_{std} + 4(L_{std} - L_{lok}) + E \tag{2.18}$$

hvor t_{std} er timer etter midnatt i lokaltid, L_{std} er standardmeridianen og L_{lok} er lokalmeridianen.

Timevinkelen ω er vinkelen mellom solas posisjon og den lokale meridianen. Den endrer seg med 15° per time og avhenger av soltiden. ω er negativ om morgenen,

positiv om ettermiddagen og kl. 12 er $\omega = 0°$.

Når alle vinklene er kjent kan cosinus til innfallsvinkelen, θ, beregnes som

$$\begin{aligned}\cos\theta =& \sin\delta\sin\phi\cos\beta - \sin\delta\cos\phi\sin\beta cos\gamma \\ &+ \cos\delta\cos\phi\cos\beta\cos\omega + \cos\delta\sin\phi\sin\beta\cos\gamma\cos\omega \\ &\cos\delta\sin\beta\sin\gamma\sin\omega\end{aligned} \tag{2.19}$$

Når flaten er horisontal faller en del ledd bort og innfallsvinkelen blir forenklet til

$$\cos\theta_z = \cos\phi\cos\delta\cos\omega + \sin\phi\sin\delta \tag{2.20}$$

θ_z kalles også for senitvinkelen og er komplimentvinkelen til solhøyden α_s.

2.6 Solinnstråling

Hvis sola anses som et sort legeme har den en temperatur på overflaten på 5777 K. For et plan vendt normalt på innstrålingsretningen utenfor atmosfæren er solinnstrålingen omtrent konstant. Denne verdien har fått navnet solarkonstanten og er gitt ved $G_{sk} = 1367\ \mathrm{Wm}^{-2}$. Variasjon i jord-sol-avstanden utgjør en årlig variasjon i G_{sk} på ±3,3%. Fluksen normalt på innstrålingsretningen kan beskrives ved relasjonen

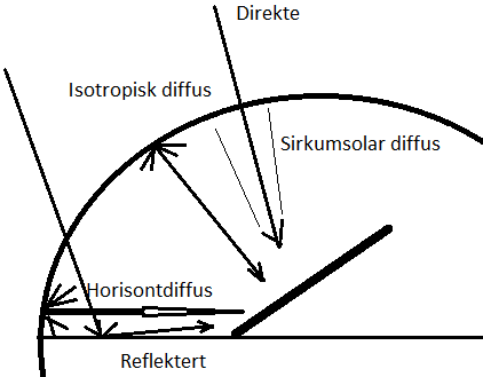
$$\begin{aligned}G_{on} =& G_{sk}(1{,}000110+ \\ &0{,}034221\cos B + 0{,}001280\sin B+ \\ &0{,}000719\cos 2B + 0{,}000077\sin 2B)\end{aligned} \tag{2.21}$$

2.6.1 Komponenter av solinnstrålingen

Solinnstrålingen som treffer jorda kan deles opp i komponentene direkte, diffus og reflektert innstråling. Summen av komponentene kalles for global innstråling og er gitt ved

$$G_T = G_{b,T} + G_{d,T} + G_{r,T} \tag{2.22}$$

hvor G_T er global, $G_{b,T}$ er direkte, $G_{d,T}$ er diffus og $G_{r,T}$ er reflektert innstråling mot en skråflate. Som i tidligere oppgave (Haugen 2000) er det valgt å benytte HDKR-modellen (Hay, Davies, Klucher og Reindl) gitt i (Duffie and Beckman



Figur 2.7: Komponenter av solinnstråling som treffer en flate. Kilde: egen tegning

2006b), og beskrevet under, for å beregne størrelsene på komponentene. I følge denne modellen er strålingen fra hemisfæren anisotrop. Komponentene er skissert i figur 2.7. Enklere modeller håndterer diffus innstråling som isotrop, men det har vist seg at den har noe forskjellig intensitet etter innfallsretning. De tre hovedkomponentene blir behandlet hver for seg.

2.6.2 Diffus innstråling

Andelen diffus innstråling er definert som $d = G_{d,T}/G_T$. Diffusandelen kan beregnes, uttrykt ved klarværsindeksen, som

$$\begin{aligned} d &= 1 - (1 - d_{cr}) \left(\frac{k_c}{k_{cr}}\right)^4 \text{ for } k < k_{cr} \\ d &= \frac{k_{cr} d_{cr} + k_c - k_{cr}}{k_c} \text{ for } k > k_c \\ \text{hvor } d_{cr} &= 0{,}0336 + 0{,}0477m \\ k_{cr} &= 0{,}887e^{-0{,}0933m} \\ \text{og } m &= \frac{1}{\cos\theta_z} \end{aligned} \tag{2.23}$$

der m er atmosfærisk masse.

HDKR-modellen tar hensyn til at de tre diffuskomponentene har forskjellig intensitet. Den totale diffuse innstrålingen er da gitt ved

$$\begin{aligned} G_{\mathrm{d,T}} &= Gd \left(1 - \frac{(1 - dG)}{G_o}\right) \left(\frac{1 + \cos\beta}{2}\right) \left(1 + \sqrt{1 - d} \sin^3\left(\frac{\beta}{2}\right)\right) \\ &= G_o \left(1 - (1 - d)\, k\right) kdF_s \left(1 + \sqrt{1 - d} \sin^3\left(\frac{\beta}{2}\right)\right) \cos\theta_z \end{aligned} \tag{2.24}$$

når sirkumsolar diffus innstråling er ekskludert fra dette uttrykket.

F_s er synsfeltet for himmelen. Det forteller hvor stor del av hemisfæren flaten er eksponert mot. Uttrykket for F_s er

$$F_s = \left(\frac{1+\cos\beta}{2}\right) \tag{2.25}$$

2.6.3 Direkte innstråling

Den sirkumsolare diffuse innstrålingen spres i mindre grad enn de andre komponentene. Ettersom differansen i innfallsvinkel mellom direkte og diffus sikumsolar innstrålinge er liten, iberegnes den sirkumsolare diffuse komponenten sammen med den direkte komponenten. Den direkte komponenten av innstrålingen, inkludert sirkumsolar diffus innstråling er dermed gitt ved

$$\begin{aligned} G_{b,T} &= (G_b + G_d A_i)\, R_b \\ &= \left(G\,(1-d) + dG\frac{(1-d)\,G}{G_o}\right)\frac{\cos\theta}{\cos\theta_z} \\ &= G\,(1-d)\left(1+d\frac{G}{G_o}\right)\frac{\cos\theta}{\cos\theta_z} \\ &= G_o\,(1-d)\,k\,(1+kd)\cos\theta \end{aligned} \tag{2.26}$$

hvor $R_b = \frac{G_{b,T}}{G_b}$ og $A_i = \frac{G_b}{G_o}$ er anisotropiindeksen. G er globalinnstrålingen mot en horisontal flate $G = G_o k \cos\theta_z$.

2.6.4 Reflektert innstråling

Jordoverflaten, inkludert trær, bygninger fjell osv. har en viss albedo. Noe av den innstrålingen som blir reflektert av jordoverflaten vi igjen treffe flaten. Resultatet er den reflekterte komponenten av den globale innstrålingen. Skråstilte flater vil ha et synsfelt mot jordoverflaten F_g og er gitt ved

$$F_g = \left(\frac{1-\cos\beta}{2}\right) \tag{2.27}$$

Når det tas hensyn til F_g kan komponenten for reflektert stråling skrives som

$$G_{r,T} = G_o k \rho_g F_g \cos\theta_z \tag{2.28}$$

der ρ_g er bakken albedo.

2.7 Duggpunktstemperatur

Duggpunktstemperaturen tilsvarer den temperaturen lufta må avkjøles til, uten endring i lufttrykk, for at vanddamp konseres til vann. Det dannes vanndråper eller dugg på objekter. Som vist ved likn. (2.10) avhenger den atmosfæriske innstrålingen betydelig av duggpunktstemperaturen, som igjen er avhengig av andelen vanndamp i lufta. Det finnes flere forskjellige mål for luftfuktighet. Både vanndamptrykk, e målt i Pa, og relativ luftfuktighet, RH, blir ofte benyttet i litteraturen. Likn. (2.10) benytter T_d direkte, og er forholdsvis enkel å beregne.

For å simulere atmosfærisk emittans er det nødvendig å ha kjennskap til T_d dersom (2.10) skal benyttes. I (Dyer and Brown 1977) er det rapportert at T_d stort sett vil ha en konstant verdi gjennom døgnet og kan tilnærmes ved

$$T_{\mathrm{d}}(t) \approx T_{a,n} \tag{2.29}$$

der $T_{a,n}$ er utetemperaturens minimum gjennom døgnet i °C. Denne tilnærmingen har vært i utstrakt bruk, men (Butler 1992) viste at for et middels tørt klima slik som i India, vil ikke (2.29) alltid stemme. Det viste seg at T_d i monsunperioden (juli-september) hadde en døgnlig amplitude på ca 25 °C, mens det økte til opp mot 25 °C i den tørre delen av året.

Luftfuktigheten er avhenging av vær og årstid. I perioder med tørke vil luftfuktigheten bli redusert, i motsetning til fuktige perioder. Potensiell evapotranspirasjon er den samlede fordampninngen fra en overflate, og består av evaporasjon fra objekter og transpirasjon fra levende planter. Når potensiell evapotranspirasjon er høy, vil den bidra til å øke luftfuktigheten. Dette er tilfellet i kystområder. Her vil luftfuktigheten tilta i varme klarværsperioder, og kan observeres når det blir disig.

Kimball et al. (1997) så på sammenhengen mellom døgnlig gjennomsnitlig T_d, årlig

nedbør og døgnlig potensiell evapotranspirasjon:

$$T_d = T_{a,n}[-0{,}127 + 1{,}121\left(1{,}003 - 1{,}4444EF + 12{,}312EF^2 - 32{,}766EF^3\right)]$$
$$\text{der } EF = \frac{l_{Ep,dag}}{l_{P,ann}} \tag{2.30}$$

hvor $l_{Ep,dag}$ er potensiell døgnlig evapotranspirasjon, $l_{P,ann}$ total årlig nedbør og $T_{a,x}$ døgnets maksimale utetemperatur.

En enklere sammenheng er funnet av Hubbard et al. (Hubbard, Mahmood, and Carlson 2003):

$$T_{\mathrm{d}} = -0{,}0360T_{a,m} + 0{,}9679T_{a,n} + 0{,}0072\left(T_{a,x} - T_{a,n}\right) + 1{,}0119 \tag{2.31}$$

der $T_{a,m}$ er døgnets middeltemperatur.

2.8 Statistiske metoder for analyse av modeller

Jeg har benyttet fem forskjellige statistiske metoder for evaluering av i hvilken grad modellene beskriver virkeligheten. Metodene er presenteret nedenfor og beskrevet nærmere i (Legates and McCabe 1999). Det blir bemerket at flere metoder bør benyttes. En metode kan tilsi at modellen stemmer bra med virkeligheten, mens en annen kan si det motsatte. I alle formlene er O_i observert og P_i beregnet verdi ved målepunkt i. $\bar{O}$ og $\bar{P}$ er totalt gjennomsnittlig observert og beregnet verdi for hele datasettet.

Gjennomsnittlig absoluttavvik, MAE (Mean Absolute Error), er gitt ved

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|O_i - P_i| \tag{2.32}$$

der N er antallet målepunkter.

Roten av gjennomsnittlig kvadratavvik, $RMSE$ (Root Mean Square Error), er

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(O_i - P_i\right)^2} \tag{2.33}$$

Bestemmelseskoeffisienten, R^2, har uttrykket

$$R^2 = \frac{\sum_{i=1}^{N} \left(O_i - \bar{O}\right)\left(P_i - \bar{P}\right)}{\left[\sum_{i=1}^{N} \left(O_i - \bar{O}\right)^2\right]^{0,5} \left[\sum_{i=1}^{N} \left(P_i - \bar{P}\right)^2\right]^{0,5}} \tag{2.34}$$

og er kvadratet av korrelasjonskoeffisienten, r som kan defineres som $r = \sqrt{R^2}$. Bestemmelseskoeffisienten tar verdier mellom 0,0 og 1,0, høyere verdi tilsvarer bedre treffsikkerhet. R^2 beskriver andelen av variansen som kan tilskrives modellen. Det er to problemer med R^2. For det første kan R^2 gi høye verdier selv om stigningstallet mellom P_i og O_i er forskjellig fra 1. Det vil si at dersom $P_i = AO_i + B$, vil $R^2 = 1$ for enhver B og enhver $A \neq 0$. For det andre er den i likhet med $RMSE$ samt metodene nedenfor, oversensitiv til sterkt avvikende verdier, da de beregner kvadratet av avviket.

Effektivitetskoeffisienten er gitt ved

$$E = 1{,}0 - \frac{\sum_{i=1}^{N} \left(O_i - P_i\right)^2}{\sum_{i=1}^{N} \left(O_i - \bar{O}\right)^2} \tag{2.35}$$

og er forholdet mellom gjennomsnittlig kvadratfeil og modellens varians. Effektivitetskoeffisienten tar verdier mellom $-\infty$ og 1,0 hvor 1,0 er best. $E = 0$ betyr at gjennomsnittsverdien i datasettet er et like bra estimat som modellen. Fordelen med effektivitetskoeffisienten er at E minker når A og B avviker fra henholdsvis 1 og 0.

Avslutningsvis er enighetskoeffisienten definert som

$$d = 1{,}0 - \frac{\sum_{i=1}^{N} \left(O_i - P_i\right)^2}{\sum_{i=1}^{N} \left(\left|P_i - \bar{O}\right| + \left|O_i - \bar{O}\right|\right)^2} \tag{2.36}$$

der $PE = \sum_{i=1}^{N} \left|P_i - \bar{O}\right| \left(\left|O_i - \bar{O}\right|\right)^2$ er den største verdien $\left(O_i - P_i\right)^2$ kan ta for hver observasjon. d tar verdier mellom 0 og 1. Høyere verdier gir mer korrekte modeller.

Kapittel 3

Fra SolDat til SolEC, implementering av kjøling

Som beskrevet i innledingen baserer simuleringsprogrammet seg på SolDat. Siden det også beregner verdier for kjøling, har det fått navnet SolEC (Solar Energy and Cooling), og uttales solisi.

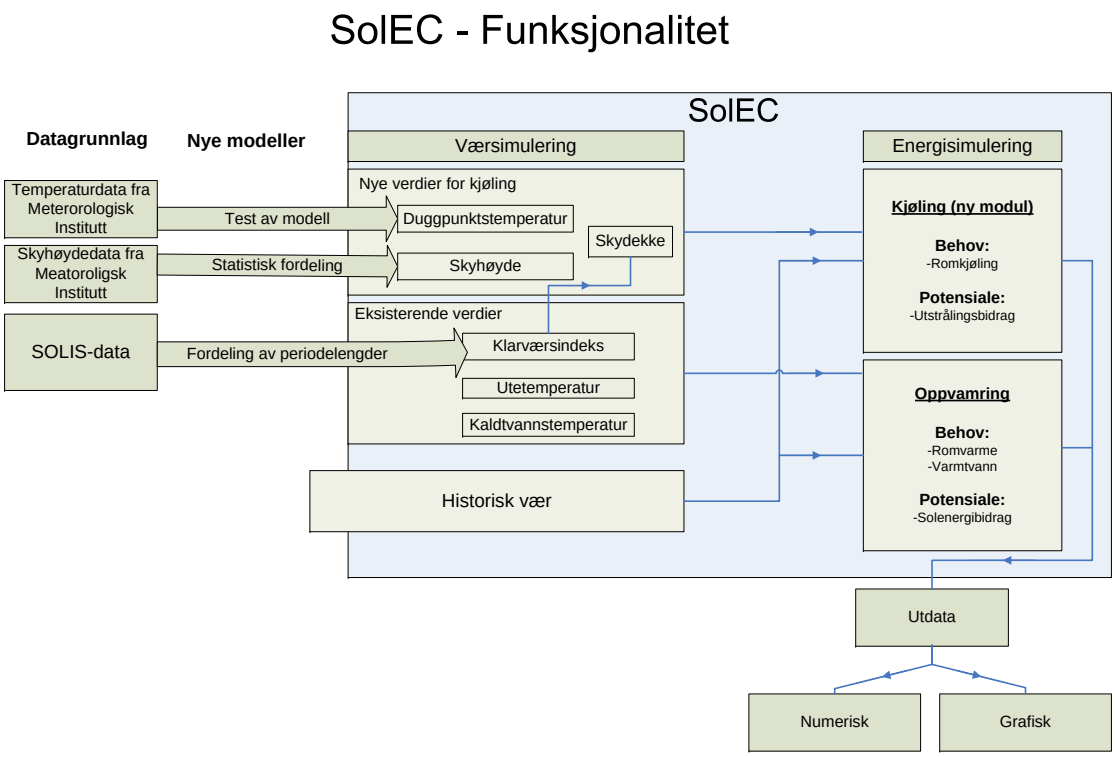
I SolEC er det tillegg til behov for kjøling av bygninger også implementert en forbedret metode for beregning av varmebehov. SolEC skal i tillegg beregne kjølepotensial ved utstråling, men fullstendig implementering av denne algoritmen er ikke nådd innenfor tidfristen. Til gjengjeld er algoritmer for generering av værparametere, som modellen for utstråling baserer seg på, ferdig implementert. Det ferdige programmet vil være oppbygd som vist skjematisk i figur 3.1.

Værparemetere kan finnes på to ulike måter.

1. Direkte værsimulering. Været simuleres automatisk av programmet ut i fra værparametere brukeren spesifiserer. Været varierer dermed for hver simulering med mindre brukeren spesifiser at simulering med samme vær er ønskelig. Frøet til den tilfeldige tallgeneratoren hentes da inn fra forrige simulering.

2. Været kan hentes fra historiske datasett. Dette er en fordel der det er tilgang på loka værdata, og simuleringene blir mer tilpasset det lokale klimaet.

Programmet gir resultater både grafisk og numerisk i tekstfiler.



Figur 3.1: Skjematisk fremstilling av SolEC

3.1 Beregninger

Verdiene blir utregnet for hvert tidssteg Δt. Δt kan gis verdier fra ett minutt til ett døgn, tidssteg større enn en time gir usikre resultater. Det er valgt å kun tillate tidsteg som deler dagen opp i et heltallig antall tidssteg. Dette for å forhindre forskyvninger fra en dag til neste. Det er kun mulig å velge fra et sett med forhåndsdefinerte verdier for tidssteget. Dersom et vilkårlig tidspunkt er indikert t_{i-1} vil neste tidspunkt være

$$t_i = t_{i-1} + \Delta t \tag{3.1}$$

Den nye versjonen av programmet tar i bruk Date- og GregorianCalendar-objekter definert av klassene i Javas API-bibliotek (Sun Microsystems 2009). Disse tar hensyn til skuddår og styrer problemfritt antall dager i hver måned. Grafene produsert av programmet er nå gitt som funksjon av dato, og ikke kun dagen i året.

3.2 Atmosfæremodell

For å beregne atmosfærisk emittans er likn. (2.10) benyttet for klart vær, og er justert ved likn. (2.12) for skydekke. Ut fra definisjonen for klarværsindeks er det et naturlig valg, som en indikator på skydekket. Klarværsindeks er beregnet ved solinnstrålingsmålinger. Pyranometere benyttes til å måle total solinnstråling, og har ideelt sett kun respons i solspekteret. Responsen har gjerne en bratt kant ved en bølgelengde på ca 4 μm. Terrestriell utstrålig er størst i det spektrale området $5 - 15$ μm. Derfor sier disse målingene lite om utstråling. Klarværsindeks beskriver sollyset som trenger gjennom skydekket. Siden skyer er nesten like ugjennomskinnelige for solinnstråling som for termisk stråling, vil termisk utstråling ved lav klarværsindeks være minimal.

Ved overskyet vær vil $k \to 0{,}1$, men $k \to 1$ ved helt klart vær. Klarværsindeks (og dermed også klarhetsindeks) blir beregnet i programmet ut fra flere sannsynlighetsfordelinger. Dette skjer uavhengig av om det er dag eller natt. Det vil si at det allerede implisitt foreligger informasjon om skydekket. Andelen skydekke, n, defineres dermed som

$$n = 1 - k_c \tag{3.2}$$

Siden $k_c > 0{,}8$ selv om skyene dekker store deler av himmelen, kan det være en bedre tilnærming å benytte gjennomsnittlig klarværsindeks over tid. Skyene beveger seg foran sola over tid og gjennomsnittlig klarværsindeks kan være en bedre indikator på skydekket. k_c er likevel benyttet, noe som skulle gi mindre usikkerhet ved korte tidssteg ($\Delta t < 10$ min).

Den atmosfæriske emittansen ϵ_{atm}, blir beregnet ved likn. (2.12). Beregning av ϵ_{atm} krever at verdier for duggpunktstemperatur, skyhøyde og klarværsindeks er kjent. Mye arbeid er lagt ned i å finne/utvikle modeller som kan benyttes til å beregne disse verdiene. Modeller for duggpunktstemperatur er beskrevet i seksjon 5.1 og modellen for skyhøyder i seksjon 5.2. Flere fordelinger av periodelengeder er utvikler for beregning av klarværsindeks og gitt i seksjon 5.3.

Den atmosfæriske innstrålingen mottatt av en flate med hellingsvinkel β, kan finnes ved hjelp av synsfeltene mot atmosfæren og bakken. Synsfelt mot himmelen ved klart vær $F_{s,c}$ er gitt ved (Cook 1989)

$$F_{s,c} = 1{,}000 + 0{,}02725\beta - 0{,}25421\beta^2 + 0{,}03372\beta^3 \tag{3.3}$$

For en atmosfære med en andel skydekke n, er synsfeltet, F_{atm} vektet med n:

$$F_{atm} = nF_s + (1-n)F_{s,c} \tag{3.4}$$

Total hemisfærisk emittans, ϵ_{hem} blir med disse hensyn beregnet ved

$$\epsilon_{hem} = F_{atm}\epsilon_{atm} + F_g(\epsilon_g + \rho_g\epsilon_{atm}) \tag{3.5}$$

der emittansen, ϵ_g, og reflektansen, ρ_g, til jordoverflaten er satt til henholdsvis 0,9 og 0,1.

3.3 Solfangere og radiatorer til kjøling

Både solfanger og radiator til kjøling er paneler med liknende egenskaper. Derfor er det naturlig i programmet å tilvirke et generisk solpanel, for så å lage underklasser ut fra disse til oppvarming og kjøling. En radiator vil ha størst virkningsgrad dersom den plasseres horisontalt. Det kan dermed være aktuelt med to adskilte panelsystemer, en til oppvarming og en til kjøling. Solfangerne er implentert som tidligere av Haugen (2000).

Varmetapet til en radiator er her kun beskrevet ut fra strålingstapet. Strålingstapet er gitt av likn. (2.9). Utstrålingen, P_{rad}, er proposjonal med T_{rad}^4, men for små temperaturforskjeller kan uttrykket lineæriseres (Cook 1989). Det finnes da på formen

$$P_{rad} = 4A\epsilon_{rad}\sigma T_a^3 \Delta T_{atm} \tag{3.6}$$

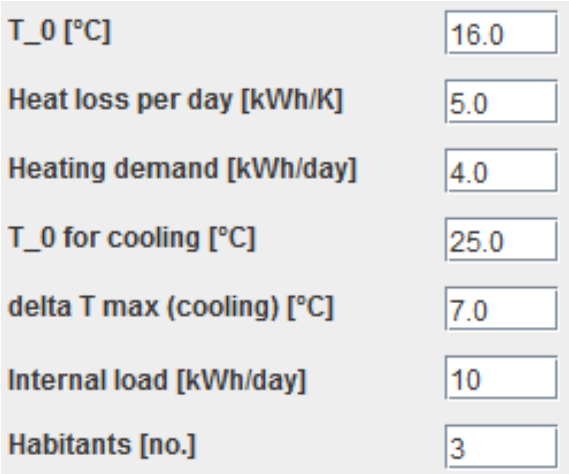
hvor $\Delta T_{atm} = T_{rad} - T_{atm} = T_{rad} - \epsilon_{atm}^{1/4} T_a$.

Tapskoeffisienten til radiatoren kan da skrives som

$$U_{rad} = 4\epsilon_{rad}\sigma T_a^3 \tag{3.7}$$

3.4 Energibehov

Behov for romvarme i boliger, varmtvann og kjøling blir beregnet av programmet. Parameterpanelet som blir benyttet i SolEC er gitt i figur 3.2.



Figur 3.2: Parameterpanelet for behov i SolEC

3.4.1 Romvarmebehov for boliger

Romvarmebehovet er den energien som kreves for å varme opp selve boligen, og varierer gjennom året enten etter brukergitt energibehov eller etter utetemperaturen.

Det er implementert to forskjellige måter å beregne behovet for romoppvarming (Q_{RV}). Det totale daglige behovet kan finnes ved å benytte en årsprofil. I denne profilen bestemmer brukeren selv den ønskelige fordelingen av romvarmebehov gjennom året. Når det totale behovet gjennom året er kjent vil det vektes mot fordelingen og dermed kan det døgnlige behovet finnes.

Q_{RV} kan også beregnes ut fra et oppgitt termisk varmetap fra boligen $\frac{dQ}{dT}$ gitt i kWh(K døgn)$^{-1}$, og momentan utetemperatur. Det er i tillegg definert en parameter Q_{min} tilsvarende en miniumumsbehov. Fortrinnsvis gjelder denne for rom hvor brukeren ønsker en konstant temperatur gjennom året, slik som f.eks. baderom. Varmebehovet finnes da ved

$$\Delta Q_{RV} = Q_{min} + (T_b - T_a)\,\frac{dQ}{dT} f_{dag} \tag{3.8}$$

hvor T_b er en brukerdefinert grensetemperatur for når bygningen trenger varme, og $f_{dag} = \Delta t/1440$ min er tidsstegets andel av døgnet. $\Delta Q_{RV} \geq Q_{min}$ bare når utetemperaturen er lavere enn grensetemperaturen. Tidligere versjon av programmet benytter middelet av utetemperaturen gjennom døgnet for å beregne ΔQ_{RV}. I denne utgaven er det endret til momentan verdi av utetemperatur. Ved sammenlikning av programversjonene er det funnet et avvik på opptil 20% i det årlige

varmebehovet. Siden utetemperaturen vil variere gjennom døgnet vil det være perioder der $T_a < T_b$, og dermed er det et varmebehov i dette tidsrommet. Ved bruk av gjennomsnittstemperatur vil ikke disse periodene detekteres og er grunnen til avviket. Bruk av momentantemperatur tar imidlertid ikke hensyn til bygningens varmekapasitet, som vil bidra til at store temperaturfluktuasjoner dempes.

3.4.2 Varmtvann

Varmtvannsbehovet baserer seg på ønsket bruk av tappevann. For varmtvannsbehov er det benyttet en normalisert tappevannsprofil. Denne kan forandres etter eget ønske og programmet foretar automatisk en normalisering. Ut fra oppgitt varmtvannsforbruk gjennom døgnet finnes det til enhver tid et varmtvannsbehov ,ΔQ_{VV}:

$$\Delta Q_{VV} = \Delta V_{VV} c_w (T_{VV} - T_{kv}) \tag{3.9}$$

hvor V_{VV} er det brukerspesifiserte varmtvannsforbuket i $l\text{dag}^{-1}$, c_w er den spesifikke varmekapasieten til vann og T_{VV} er den ønskelige varmtvannstemperaturen. V_{VV} følger årlig og døgnlig profil, som bestemmes av brukeren. Variasjon i varmtvannsbehovet kan for eksempel skyldes ferie.

3.4.3 Kjølebehov

Kjølebehovet er definert som den energien som til enhver tid kreves for å kjøle boligen ned til en brukergitt komforttemperatur. Kjølebehovet til romkjøling Q_{RK} er definert positivt.

Beregning av kjølebehov fungerer på samme måte som beregning av responsstyrt oppvarmingsbehov. Det er for øyeblikket ikke definert profiler for kjølebehov. Kjølebehovet beregnes kun når innetemperaturen er høyere enn den valgte temperaturgrense. Differansen mellom utetemperatur og valgt temperaturgrense er $\Delta T_{RK} = T_a - T_{b,k}$ hvor $T_{b,k}$ er en parameter brukt av programmet og er den kritiske innetemperaturen. Dersom denne overstiges finnes det et kjølebehov. Kjølebehovet er

nå

$$\Delta Q_{RK} = (Q_{last} + \frac{dQ}{dT}\Delta T_{RK}) f_{dag} \tag{3.10}$$

hvor Q_{last} er summen av all energi generert av interne varmekilder, slik som elektriske apparater og beboere som befinner seg inne i boligen, i løpet av et døgn. Q_{last} er oppgis i kWh(dag)$^{-1}$.

Når temperaturforskjellen $\Delta T_{b,RK}$ overskrides vil beboerne føle et kjølebehov. Ved å sette $\Delta Q_{RK} = 0$ kWh kan $\Delta T_{b,RK}$ finnes ved

$$\begin{aligned} 0 &= Q_{last} + \frac{dQ}{dT}\Delta T_{b,RK} \\ \Rightarrow \Delta T_{b,RK} &= -\frac{Q_{last}}{\frac{dQ}{dT}} \end{aligned} \tag{3.11}$$

Siden $Q_{last} \geq 0$ er $\Delta T_{b,RK} \leq 0$.

Det totale kjølebehovet for hvert tidssteg er dermed

$$\Delta Q_{RK} = (Q_{last} + \frac{dQ}{dT}\Delta T_{K,ute}) f_{dag} \quad , \Delta T_{RK} > \Delta T_{b,RK} \tag{3.12}$$

Det finnes altså kun et reelt kjølebehov når den kritiske temperaturforskjellen i likn. (3.11) overskrides.

I programmet kan det velges en maksimal grense for hvor mye temperaturen skal senkes, slik at kjølebehovet kan begrenses.

3.5 Energibidrag fra radiator og solfanger

Energibidraget er den andelen av energibehovet som blir generert av systemet. Solenergibidraget er altså andelen av romvarmebehovet og varmtvannsbehovet som blir generert av solfangerne. Kjølebidraget er andelen av kjølebehovet som blir tilfredsstilt av radiatorene.

Innenfor ethvert tidsintervall vil det foreligge et potensial $\Delta Q_{pot} \geq 0$ og et behov $\Delta Q_{last} \geq 0$. Ved oppvarmingsbehov (ΔQ_{RV}) vil energien hentes fra varmelageret, mens det ved kjølebehov ΔQ_{RK} vil tilføres kjølelageret. Samtidig finnes det et varmtvannsbehov (ΔQ_{VV}) der energien hentes fra varmelageret. Varmtvannet befinner seg i en innertank i varmelageret og er i termisk likevekt med vannet i

varmelageret. Potensielt kan det være tilgjengelig en energimengde for oppvarming ΔQ_v og for kjøling ΔQ_k fra solfangerne, men samtidig er det nødvendigvis noe varmetap fra varmelagrene. Dette gir til sammen en temperaturforandring i tankene tilsvarende $\Delta T = (\Delta Q_{pot} - \Delta Q_{last})/c_w V$ der V er volumet til lageret. Det gir altså en positiv energigevinst dersom det eksisterer et $\Delta Q_{pot} > \Delta Q_{last}$. Dette gir for kjøling og oppvarming relasjonene:

$$\Delta T_v = \frac{\Delta Q_v - \Delta Q_{VV} - \Delta Q_{tap}}{c_w V_v} \tag{3.13}$$

$$\Delta T_k = \frac{\Delta Q_k - \Delta Q_{RK} - \Delta Q_{tap}}{c_w V_k} \tag{3.14}$$

hvor V_v er volumet til varmelageret og V_k er volumet til kjølelageret. ΔT_v er temperaturforandringen i varmelageret og ΔT_k temperaturforandringen i kuldelageret etter et tidsteg. Temperaturen i varmelageret skal aldri synke lavere enn termostattemperaturen $T_{t,v}$. Samtidig er det nødvendig å sette en øvre grense for å ikke skade systemet. Denne er satt initielt til 95 °C. Kuldelageret har også en termostattemperatur $T_{t,k}$ som ikke skal overskrides når det er i bruk. Minimumstemperaturen er satt til 0 °C slik at ikke kjølelageret fryser. Temperaturøkningen fra likn. (3.13) legges til temperaturen til systemet og korrigeres til slutt slik at $T_{kv} < T_v(t_i + \Delta t) < T_{v,maks}$. Dersom $T_v(t_i + \Delta t) < T_{t,v}$ eller $T_k(t_i + \Delta t) > T_{t,k}$ trengs ekstra energi, $\Delta Q_{eks,v}$ og $\Delta Q_{eks,k}$, for å holde systemet på ønsket temperatur.

Andelen energi fra solfangerne a_s tilsvarer solenergiandelen av behovet til romoppvarming og varmtvannsforbruk. Tilsvarende er a_k andel kjølepotensiale fra radiatorene. Disse andelene er gitt ved behovet på dette tidspunktet og er uttrykt ved

$$a_v = 1 - \frac{\Delta Q_{eks,v}}{\Delta Q_{VV} + \Delta Q_{RV}} = \frac{\Delta Q_v}{\Delta Q_{VV} + \Delta Q_{RV}} \tag{3.15}$$

$$a_k = 1 - \frac{\Delta Q_{eks,k}}{\Delta Q_{RK}} = \frac{\Delta Q_k}{\Delta Q_{RK}} \tag{3.16}$$

der a_v er solenergiandelen og a_k er kjøleandelen.

Det er nødvendig å finne temperaturforandringen $\Delta T_v = T_{v,i} - T_{v,i-1}$ i varmelageret for å finne solenergiandelen i løpet av et tidssteg. Denne temperaturforskjellen beregnes eksplisitt. På grunn av at temperaturen i varmelageret ikke kan endre seg fritt, kan ikke temperaturforskjellen beregnes ut fra netto energioverføring.

Når $T_{t,v} < T_v$ etter at lagertemperaturen er justert etter varmebehovet, er det ikke behov for tilleggsoppvarming og $a_v = 1$. For kjølelageret gjelder det samme når $T_k < T_{t,k}$.

Når ΔT_v og ΔT_k er funnet, kan energibidrag fra solenergi og kjølebidrag ved stråling til nattehimmelen finnes ved

$$\Delta Q_{as} = a_s \left(\Delta Q_{RV} \Delta Q_{VV}\right) \tag{3.17}$$

$$\Delta Q_{ak} = a_k \left(\Delta Q_{RK}\right) \tag{3.18}$$

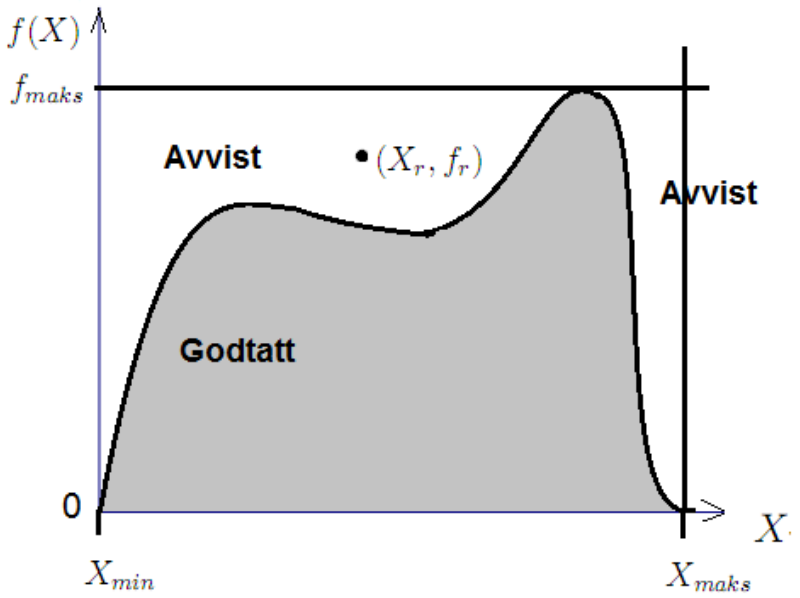
3.6 Monte Carlo-metoden

Monte Carlo-metoder er algoritmer som baseres på generering av en rekke tilfeldige tall. De egner seg for beregninger som er for kompliserte til å bli behandlet analytisk, og til beskrivelse av fenomener som er stokastiske av natur, slik som været.

Tilfeldig-tall-generatorer gir uniforme fordelinger av tall mellom 0.0 og 1.0. De fleste stokastiske prosesser er ikke uniform fordelte, heller ikke fordelingen av klarværsindeks. Det benyttes derfor en rekke forskjellig fordelinger for å generere døgnlig, timevis og momentan klarværsindeks. I SolEC benyttes de samme fordelingene som er implementert i tidligere program (Haugen 2000).

Fordelingene er relativt komplekse og avvisningsmetoden blir derfor benyttet ved generering av klarværsindeks. Avvisningsmetoden går ut på å finne maksimalpunktet f_{maks} for sannsynlighetstetthetsfordelingen $f(X)$, og sette et intervall X_{min} og X_{maks} for den stokastiske variabelen X. Utenfor dette intervallet skal sannynlighetsfordelingen ha neglisjerbar verdi. Grafisk sett vil i et $f(X), X$-diagram, hvor f er sannsynlighetsfordelingen, f_{maks} beregnes og en linje $f(X) = f_{maks}$ trekkes gjennom. To tilfeldige uniformt fordelte tall med indeks r vil plukkes ut, en X_r-verdi og en f_r-verdi. Dersom f_r treffer innenfor fordelingen $f(X_r)$ vil det anses som et treff og den stokastiske verdien $X = X_r$ plukkes ut. Dersom (X_r, f_r) kommer utenfor fordelingen, trekkes det nye par av tilfeldige tall helt til paret havner innenfor fordelingen. Metoden er illustrert i figur 3.3.

Siden periodelengdene er weibullfordelt (se seksjon 3.6.1), kan en annen metode



Figur 3.3: Avvisningsmetoden. Det velges en vilrkårlig $X_{min} < X_r < X_{maks}$ og $0 < f_r < f_{maks}$, helt til punktet (X_r, f_r) havner i det grå feltet (Kilde: egen tegning).

benyttes. Ved å invertere uttrykket for akkumulert fordeling og løse for periodelengden, er det kun nødvendig å plukke ut ett tilfeldig tall. Dette tallet tilsvarer den akkumulerte sannsynligheten og settes inn i uttrykket. Dette er den inverse metoden. Periodelengdene blir funnet ved en litt annen metode. Siden det er valgt å benytte periodelengder i hele dager er fordelingen diskret. En periodelengde kan da plukkes ut ved hjelp av en for-løkke:

1. En tilfeldig valgt akkumulert sannsynlighet blir plukket ut.
2. Periodelengden blir initielt satt til $L = 1$.
3. Periodelengden blir økt med en, helt til fordelingens akkumulerte sannsynlighet overskrider den tilfeldig valgte akkumulerte sannsynligheten.
4. Den forrige periodelengden blir returnert

3.6.1 Weibullfordeling

Det finnes flere ulike sannsynlighetsfordelinger som kan brukes til å modellere data, blant annet poisson-, normal-(gaussisk), lorentzfordeling etc. Det er vist at

sannsynligheten for neste periodelengde avtar som en eksponensialfunksjon av periodens lengde (Ingebretsen 1991). Weibullfordelingen har vist seg å passe bra, siden sannsynlighetstetthetsfordelingen avtar eksponensielt med den stokastiske variabelen.

Weibullfordelinger har et eget klassifiseringssystem. $F(l;\theta) = P(L \leq l)$ er sannsynligheten for at en stokastisk periodlengde, L_p, er kortere enn en bestemt periodelengde l. θ er de beregnede parameterne for fordelingen. $f(l;\theta) = \frac{dF(l;\theta)}{dl}$ er sannsynlighetstettheten til fordelingen (D. N. Prabhakar Murthy and Jiang 2004).

Modellen som tidligere er brukt for å finne fordelinger av værperioders varighet er på formen

$$F(l; l_0, \beta) = 1 - e^{-\left(\frac{l}{l_0}\right)^{\beta}} \tag{3.19}$$

med sannsynlighetstetthet

$$f(l; l_0, \beta) = \frac{\beta}{l_0}\left(\frac{l}{l_0}\right)^{\beta-1} e^{-\left(\frac{l}{l_0}\right)^{\beta}} \tag{3.20}$$

Likn. (3.19) er standard weibullfordeling med to parametere og en spesialvariant av den mer generelle fordelingen:

$$F(l; l_0, l_1, \beta) = 1 - e^{-\left(\frac{l-l_1}{l_0}\right)^{\beta}} \tag{3.21}$$

der l_0 er skaleringsparameteren, l_1 er forskyvningsparameteren og β er formparemeteren.

Siden fordelingen av periodelengder er diskret, er det forsøkt å finne en diskret weibullmodell som kan tilpasses de observerte periodelengdene. Ved invers metode er det nødvendig med en korreksjonsfaktor for å reprodusere den midlere periodelengden, og gjennomsnittlig periodelengde må gis som parameter. For å plukke ut neste periodelende i en diskret fordeling er det kun nødvendig å trekke ut et tilfeldig tall d i intervallet $(0,\ 1)$ og finne lengden tilsvarende $F(l) < d < F(l+1)$.

Det er funnet at den diskrete fordelingen $F(l; q, \beta_w)$, med patametere $0 < q < 1$ og $\beta_w > 0$, gir gode tilpasninger i flere tilfeller. Fordelingen stemmer bra for korte periodelengder, se seksjon 5.3. Denne modellen har akkumulert sannsynlighet på

formen

$$F(l) = 1 - q^{l^{\beta_w}} \tag{3.22}$$

og sannsynlighetststetthet på formen

$$f(l) = q^{(l-1)^{\beta_w}} - q^{l^{\beta_w}} \tag{3.23}$$

3.6.2 Weibulldiagram

Ved bruk av Weibull Probability Parer plot (WPP)-plot eller weibulldiagram, vil en fordeling som følger weibullmodellen vise seg som en rett linje med β som stigningstall og $\ln(-\ln q)$ som konstant. Likn. (3.22) kan enkelt skrives om til

$$\ln(-\ln(1-F)) = \beta_w \ln l + \ln(-\ln q) \tag{3.24}$$

og parameterne til fordelingen kan da finnes ved kurvetilpasning.

I denne oppgaven det valgt en enklere løsning for bestemmelse av parametere til weibullfordelingen. I (Khan, Khalique, and Abouammoh 1989) er det funnet at parameterne til den diskrete weibullfordelingen kan beregnes kun ved de første to verdiene $f(1)$ og $f(2)$. Et estimat for q er gitt ved

$$q = 1 - f(1) \tag{3.25}$$

Når q er funnet, kan β_w estimeres ved

$$\beta_w = \ln\left(\frac{q - f(2)}{\ln q}\right) / \ln 2 \tag{3.26}$$

3.6.3 Generalisert paretofordeling

Den generaliserte paretofordelingen er gitt ved

$$F(l; l_0, c) = 1 - \left(1 + \frac{(l-\mu)c}{l_0}\right)^{-\frac{1}{c}} \tag{3.27}$$

med lokalitets parameter μ, skaleringsparameter l_0 og formparameter c. Fordelingen utmerker seg til å beskrive avtagende haler av fordelinger. Det er ikke alltid fordelingen av periodelengder følger weibullfordelingen. Fordelingen får ofte

avtagende sannsynlighet for de ekstra lange periodelengdene og det er her GP-fordelingen kommer til sin rett. Parameterne l_0 og c kan bestemmes ved å bruke Matlab-programmet vist i seksjon C.5. Matlab-programmet benytter seg av ikke-lineær minste kvadraters metode.

Kapittel 4

Datagrunnlag

Oppgaven behandler og presenterer analyser av data fra flere forskjellige kilder. Datakildene for solinnstråling, atmosfærisk innstråling, temperaturer og skyhøyder er presentert i dette kapittelet.

4.1 Data for duggpunktstemperatur

Sikre verdier for utetemperatur og duggpunktstemperatur er hentet fra Meteorologisk Institutts tjeneste eKlima (Meteorologisk Institutt 2009a), fra Blindern værstasjon, og har oppløsning på en time. Det er sett på sammenhengen mellom duggpunktstemperatur og utetemperatur, med data på formen

```
#St.no;Year;Mnth;Date;Time(NMT);TA;TD
18700;2008;1;1;1;-2.2;-7.4
18700;2008;1;1;2;-2.0;-7.0
18700;2008;1;1;3;-1.6;-6.7
18700;2008;1;1;4;-1.2;-6.0
18700;2008;1;1;5;-1.3;-5.7
18700;2008;1;1;6;-1.4;-5.7
18700;2008;1;1;7;-1.8;-5.3
```

For innlesing er javaprogrammet vist i seksjon 2.7 brukt. Modellene presentert i seksjon benytter både $T_{a,n}$, $T_{a,x}$ og $T_{a,m}$. I innlesingsprogrammet blir disse verdiene beregnet for hvert døgn sammen med $T_{d,m}$.

Sammen med innlesningsprogrammet er det utviklet et bibliotek som beregner statistiske data for modellene, der likningene presentert i seksjon 2.8 er implementert. Denne funksjonen er benyttet for å beregne statistiske verdier av modellene for duggpunktstemperatur og atmosfærisk emittans (seksjon 7.6).

4.2 Skyhøydedata

Parameteren z_h i likn. (2.12) tilsvarer skyhøyden. For å bestemme atmosfærisk emittans er det nødvendig å ha kjennskap til skyhøyden, som gir et pekepinn på temperaturen til skyene. I denne oppgaven blir skyhøydedata fra årene 2000 til 2008 lest inn i javaprogrammet vist i seksjon C.1. Dette programmet beregner enkele, diskrete fordelinger for hvordan skyhøyden forandrer seg i tid.

Dataene fra Meteorologisk Institutt er gitt på formen

```
18700;2003;11;30;3;-
18700;2003;11;30;4;200.0
18700;2003;11;30;5;-
18700;2003;11;30;6;-
18700;2003;11;30;7;50.0
18700;2003;11;30;8;-
18700;2003;11;30;9;100.0
18700;2003;11;30;10;100.0
18700;2003;11;30;11;100.0
18700;2003;11;30;12;100.0
18700;2003;11;30;13;100.0
18700;2003;11;30;14;100.0
18700;2003;11;30;15;100.0
18700;2003;11;30;16;100.0
18700;2003;11;30;17;100.0
18700;2003;11;30;18;100.0
```

hvor den siste verdien i hver linje tilsvarer skyhøyden.

4.3 SOLIS-data

Som tidligere nevnt, er været uforutsigbart på tidskala lengre enn noen dager. Det er dermed ønskelig med statistiske modeller som kan brukes til å simulere værforhold som har betydning for potensiale og behov. Tidligere modell for periodelengder er utviklet kun fra værdata for Oslo. Siden det vil forekomme store klimatiske forskjeller avhengig av hvilket område det simuleres for, er det nødvendig med store mengder data fra flere områder i landet.

Innstrålingsmålingene er fra 33 målestasjoner i Norge. Disse dataene er produsert av forskjellige videregående skoler på slutten av 90-tallet(Hetland and Oterholm 2001). Videregående skoler som har deltatt i SOLIS-prosjektet og befinner seg i Norge, er listet opp i tabell 4.1, sammen med perioden målingene er blitt foretatt. For å øke det statistiske grunnlaget ble datasettene fra flere stasjoner som befinner seg innenfor et område slått sammen. Det medførte at målingene fra alle skolene kunne bli tatt med i beregningene av værperiodenes varighet.

I SOLIS-prosjektet var det valgt et standard oppsett for alle stasjonene. Målingene ble foretatt med et SOLDATA-pyranometer med hellingsvinkel på 45° og asimutvinkel på 0°. Grunnen til at de valgte denne hellingsvinkelen er for å måle varmtvannspotensialet for solvarmeanlegg. Målingene er foretatt hvert 3. minutt, lagret i en datalogger av typen Black Star 2308 I/O Interface", og integrert over tidsrom på et døgn. Dermed er total solinnstråling, betegnet med H, gjennom døgnet funnet for hver dag.

I denne oppgaven ble data fra hver skole hentet fra databasen (Naturfagsenteret), og lagret i en egen fil. Alle filene ble samlet i en mappe, og lest inn en etter en vha. javaprogrammet presentert i seksjon C.3. Innstrålingsdata var lagret på formen

```
#Tidspunkt Solinnstråling (kWh/m^2) Temperatur (C)
1995-01-01 5 -4.7
1995-01-02 8 -8.0
1995-01-03 7 -3.2
1995-01-04 4 1.6
1995-01-05 9 4.0
1995-01-06 8 4.6
1995-01-07 4 4.0
```

Tabell 4.1: Alle målestasjoner fra SOLIS-prosjektet inndelt etter område og fylke. Startdato og sluttdato for målingene er også vist.

Område	Fylke	Skole	Startdato	Sluttdato
Sørlandet	Aust-Agder	Møglestu vgs	7.12.1994	31.3.2000
		Tvedestrand vgs	1.7.1994	30.4.1995
	Telemark	Bamble vgs	13.1.1995	28.2.1999
		Porsgrunn vgs	1.11.1994	28.2.1999
		Vest-Telemark vgs	7.2.1992	29.2.2000
Vestlandet	Rogaland	Dalane vgs	1.1.1995	28.2.1998
	Hordaland	Bergen Katedralskole	1.1.1995	31.12.1997
Østlandet	Østfold	Kirkeparken vgs	1.4.1995	27.2.1999
	Akershus	Bjertnes vgs	1.1.1995	28.2.1999
		Eikeli vgs	1.8.1994	31.5.1996
		Rælingen vgs	1.1.1996	31.12.1997
		Jessheim vgs	1.6.1994	31.12.1998
	Oslo	Fysisk institutt	1.1.1995	28.2.1999
		Oslo katedralskole	1.1.1996	28.2.1999
	Hedmark	Solør vgs	1.1.1994	31.5.1997
	Oppland	Lillehammer vgs	1.2.1994	28.2.2001
		Otta vgs	1.5.1995	31.5.1996
Midt-Norge	Møre og Romsdal	Fræna vgs	1.10.1994	31.1.1998
		Ulstein vgs	8.1.1996	28.2.1999
	Sør-Trøndelag	Heimdal vgs	1.5.1994	31.1.2000
		Rissa vgs	1.1.1996	28.2.1999
		Gauldal vgs	1.6.1994	28.2.1999
	Nord-Trøndelag	Levanger vgs	1.12.1994	28.1.1999
		Olav Duun vgs	18.10.1995	30.1.1996
		Ytre Namdal vgs	1.11.1994	31.5.2000
		Ole Vig vgs	1.12.1995	30.6.1997
Nord-Norge	Nordland	Bodø vgs	1.12.1994	28.2.1999
		Moheia vgs	1.7.1994	28.5.1997
		Vestvågøy vgs	1.12.1994	31.3.2001
	Troms	Breivika t. fagskole	1.1.1995	28.2.1999
		Finnfjordbotn vgs	1.7.1994	31.8.2000
		Heggen vgs	1.2.1996	28.2.1999
		Sjøvegan vgs	1.11.1993	30.9.2000

der midterste kolonne representerer den totale solinnstrålingen gjennom døgnet.

Klarværsindeksen er definert ut fra likn. (2.3). For å bestemme klarværsindeksen har det vært nødvendig å finne maksimal solinnstråling gjennom døgnet, H_c.

Som en metode for å beregne klarværsinnstråling har jeg benyttet HDKR-modellen beskrevet i seksjon 2.6.1. Komponentene av solinnstrålingen ble funnet ved å benytte modellene for direkte, diffus og reflektert stråling. Klarværsindeksen ble da satt til 1 og diffusandelen beregnet ved likn. (2.23) med $k_c = 1$. Når $k_c = 1$, er klarhetsindeksen $k = \tau_c$. Uttrykket for klarværsinnstrålingen blir da

$$\begin{aligned} G_{45^\circ} =& G_o \tau_c [\frac{\sqrt{2}}{2} (1-d)(1+\tau_c d) + \\ & (1-(1-d)\tau_c) dF_s \left(1+\sqrt{1-d}\sin^3\left(\frac{\beta}{2}\right)\right)\cos\theta_z + \\ & \rho_g F_g \cos\theta_z] \end{aligned} \tag{4.1}$$

Den totale klarværsinnstrålingen gjennom døgnet ble beregnet ved å sette timevinkelen til $\omega = -\pi$ og øke denne med $\Delta\omega = 0{,}01$ radianer i en løkke. Innstrålingen ble beregnet ved hjelp av likn. (4.1) for hvert steg og multiplisert med tidsforskjellen mellom hvert steg $\Delta t = \Delta\omega/(180^\circ/\pi)/(15^\circ/\text{time})$. Total innstråling for døgnet er funnet når $\omega = \pi$.

En empirisk metode for å finne innstråling ved klarvær er å lineærinterpolere mellom toppunktene for maksimal observert innstråling. Maksimal observert innstråling som funksjon av dagen i året ble funnet ved å gå gjennom hele datasettet for hver enkelt stasjon. Interpoleringen ble startet ved den dagen i året innstrålingen er på sitt minimum. Dagen i året for minimal og maksimal innstråling inntreffer når deklinasjonen er maksimal og minimal, og for å finne disse dagene er den deriverte likning 2.16 funnet og løst med hensyn på dagen i året. Det er på grunn av begrenset antall data ikke sikkert at disse dagene har den minimale og maksimale innstrålingen. Gode verdier for minimal innstråling er funnet ved å benytte maksimal innstråling innenfor et tidsintervall på ± 5 dager. Innstrålingen gjennom dagen for maksimal innstråling er funnet ved å benytte den høyeste verdien i datasettet.

Maksimal innstråling for hver dag er beregnet ved å interpolere mellom toppverdiene gjennom hele året. Toppunktene ble funnet ved stigende innstråling både for synkende og stigende dag i året. Det ble lett etter topper i innstrålingen som hadde høyere verdi enn forrige topp og stigningstallet mellom toppene ble beregnet. Dagene mellom disse toppene fikk interpolerte klarværsverdier uavhengig av tid-

ligere innstrålingsverdi. Selv om datasettet er stort er der ikke sikkert at klarvær har inntruffet for hver dag i året. Døgnverdier større enn ekstraterrestriell verdi er funnet og erstattet med verdier tilsvarende HDKR-modellen.

For at denne metoden skal gi nøyaktige klarværsindekser, kreves store datasett med målinger over mange år. Slik vil det finnes målinger med klarvær for så mange dager i året som mulig. Stor nøyaktighet kreves ikke for å beregne om en dag har godt eller dårlig vær. De fleste datasett er ansett som store nok til dette formålet.

For å finne ut om en dag har bra eller dårlig vær er den gjennomsnittlige døgnlige klarværsindeksen funnet. Ved å benytte målt innstrålingsverdiverdi og klarværsinnstråling er klarværsindeksen funnet.

4.3.1 Periodelengder

Det er funnet en sammenhengende rekke med klarværsindekser med informasjon om manglende perioder. Ut fra denne rekken er periodelengdene funnet ved å telle antall sammenhengende dager med klarværsindeks over eller under $k_c = 0{,}5$. Hvis $k_c > 0{,}5$ for hver dag i perioden er den god, og dersom $k_c \leq 0{,}5$ er perioden dårlig.

Periodelengdefordelingen er funnet ved å se på neste periodelengde ved en gitt værtype. Det er bestemt grafisk via weibulldiagram om fordelingen er paretofordelt eller weibullfordelt. En rett linje betyr at weibullmodellen er det riktige valget, men dersom kurven er avtagende for lange periodelengder vil det ikke være en god modell. I dette tilfellet vil sannsynligheten for lange periodelengder bli for lav i forhold til de korte. Den generaliserte paretofordelingen blir da benyttet.

4.4 Data for atmosfærisk emittansmodell

Det er forsøkt å finne en modell for atmosfærisk emittans. Datagrunnlaget baserer seg på solinnstrålingsmålinger, termiske utstrålingsmålinger og temperaturdata.

Det er gjort et forsøk i å finne en relasjon mellom atmosfærisk innstråling og klarhetsindeks. Modellen vil være en funksjon av to variable siden siden ϵ_{atm} avhenger av ϵ_0 gjennom duggpunktstemperaturen i tillegg til tiden på døgnet. Den empiriske formelen vil da være på formen $\epsilon_{atm} = \epsilon_{atm}(k, \epsilon_0)$. Klarhetsindeksen er brukt i

Figur 4.1: Oppsett av pyrgeometer (forgrunn) og pyranometer (på skrå i bakgrunn).

første omgang da klarværsindeksen er vanskeligere å beregne.

Plassering av pyrgeometeret og pyranometeret som var benyttet for målte verdier i datasettene er vist i figur 4.1. Data for sommeren 2009 er hentet inn fra 3 forskjellige kilder.

4.4.1 Pyrgeometerdata

Et pyrgeometer måler atmofærisk innstråling fra hemisfæren dersom horisonten er fri. Det vil si at amosfærisk innstråling, med bølgelengder i området 4 μm til 40 μm, vil detekteres.

Verdier for atmosfærisk innstråling er hentet fra (Degnes-Ødemark 2009). Disse dataene er beregnet med et pyrgeometer plassert horisontalt på toppen av taket til Sollabben, se figur 4.1. Målingene er utført hvert halve minutt sammenhengende i en periode på over et år.

Pyrgeometeret som ble benyttet er av typen "Kipp & Zonen CG1 pyrgeometer" (Kipp & Zonen 2009). Dekkglasset transmitterer i overkant av 50% av innstråling med bølgelengder fra ca $4,5$ μm til 42 μm. Siden det er flatt har det et synsfelt på 150°. Innstråling fra senitvinkler større enn ca 75° vil altså ikke detekteres, like-

vel vil resultatene blir riktige ettersom kalibrering hos produsent blir utført mot innstråling fra et synsfelt på 180°.

Det oppstår problemer når pyrgeometeret benyttes til å måle atmosfærisk innstråling på dagtid. Ifølge manualen (Kipp & Zonen 2009) vil pyrgeometeret gi en positivt avvik på 25 Wm^{-2} ved normalt innfallende solinnstråling på 1000 Wm^{-2}. Dette skjer ved oppheting av dekkglasset som skal beskytte sensoren. Pyrgeometeret benytter et silisumbasert dekkglass. På baksiden av dekkglasset befinner det seg et lavpassfiler som fjerner stråling fra Sola med kortere bølgelengder enn ca 5 μm. Grunnen til det er at 99% av energien i sollyset befinner seg i fotoner med bølgelengder $\lambda \leq 4$ μm.

Data som er hentet i perioden er på formen

```
2009;05;07;11;31;33;0.138530;-0.000595;0.103800;-0.000010;22.672586;
  7.767120;282.888727;362.166282;-0.949915;9.553535
2009;05;07;11;32;03;0.138545;-0.000590;0.103811;-0.000012;22.670179;
  7.895614;282.915495;362.175285;-1.078374;9.555292
2009;05;07;11;32;33;0.138556;-0.000591;0.103814;-0.000013;22.668497;
  7.873002;282.924290;362.091997;-1.206832;9.539037
2009;05;07;11;33;03;0.138568;-0.000590;0.103825;-0.000015;22.666454;
  7.891774;282.950676;362.112609;-1.321768;9.543060
2009;05;07;11;33;33;0.138579;-0.000588;0.103836;-0.000013;22.664627;
  7.933541;282.978975;362.440567;-1.139222;9.607046
```

der atmosfærisk innstråling, gitt i Wm^{-2}, befinner seg i kolonne 14.

4.4.2 Pyranometerdata

Klarhetsindeks ble opprinnelig forsøkt beregnet ut fra egne målinger høsten 2009 med pyranometer plassert horisontalt. Det viste seg snart at solhøyden var for lav på denne tiden av året til at akseptable målinger kunne gjennomføres. Den store forskjellen i relativ respons som funksjon av innfallsvinkel funnet ved forsøk (se tillegg A) gjorde det umulig å beregne klarhetsindeksen tilfredsstillende.

Solinnstrålingsverdier er hentet fra et oppsett satt opp av en tidligere utvekslingsstudent. Måleinstrumentet var et pyranometer av typen SolData 458SPC og var plassert omtrent 2 m fra pyrgeometeret. Det hadde en hellingsvinkel $\beta = 32°$, og en asimutvinkel på $\gamma = 18°$. Pyranometeret ble kalibrert i juni 2006 (Gjessing

2006) og har en kalibreringskonstant på 158 $\mathrm{mV}\left(\mathrm{kWm}^{-2}\right)^{-1}$. Det var koblet opp mot en datalogger av typen Almemo 2890-9 fra Ahlborn, som logget innstrålingsverdier hvert annet minutt gjennom hele døgnet. Siden dataloggeren har en begrenset kapasitet på 512kB, var det nødvendig å overføre data før minnet ble fullt, og målingene ble da avbrutt. Det finnes derfor perioder uten data. Det viste seg at tiden på dataloggeren var innstilt feil og gikk 1 t og 40 min for fort i forhold til norsk sommertid. Dette ble det korrigert for ved å trekke denne tidsforskyvningen fra hvert tidssteg.

Data som ble lest inn var på formen

```
#"DATE:";"TIME:";;"M01: °C";"M02: °C";"M03: °C";"M04: °C";"M05: °C";
 "M06: °C";"M07: °C";"M08: °C"
"04.05.09";"15:21:12";146.05;79.6;81.2;79.6;81.9;79.;71.;83.4;21.5
"04.05.09";"15:23:12";144.55;79.2;80.7;79.1;81.4;78.6;70.6;82.9;21.4
"04.05.09";"15:25:12";146.43;78.8;80.2;78.9;81.;78.3;70.4;82.6;21.6
"04.05.09";"15:27:12";144.7;78.5;79.9;78.6;80.6;78.1;70.3;82.2;22.
"04.05.09";"15:29:12";144.49;78.6;80.;78.9;80.9;78.2;70.5;82.2;21.6
"04.05.09";"15:31:12";146.61;78.7;80.1;79.;81.;78.4;70.8;82.3;22.3
```

der innstrålingsdata befinner seg i tredje kolonne.

4.4.3 Temperaturdata

Disse verdiene er målt hver time i perioden fra 1.4.2009 til 30.9.2009, og er på samme form som i seksjon 4.1.

Java ble derfor benyttet til å lese inn dataene med dato og tid vha. kalenderfunksjonen som er innebygget. Kildekoden befinner seg i tillegg C.4. Å samordne temperaturdata, solinnstrålingsdata og atmosfærisk innstrålingsdata var en komplisert oppgave, da de ikke var på samme form eller hadde samme tidsoppløsning. Siden temperaturdata kun har timeverdier, ble disse interpolert og flettet inn i data for atmosfærisk innstråling. Atmosfærisk emittans ved klarvær ble beregnet ut fra likning 2.10, og emittansen for atmosfærisk innstråling ved dette tidspunktet ble funnet ved å skrive om 2.1:

$$\epsilon_{atm} = \frac{R_{atm}}{\sigma T_a^4} \tag{4.2}$$

Klarhetsindeks ble funnet vha. 2.2 der cosinus til innfallsvinkelen ble funnet ved relasjonen 2.19 etter at soltid ble beregnet ut fra lokaltid. Klarhetsindeksen ble kun beregnet for innfalsvinkler mindre enn 60°, for å hindre effekten av den relative responsen til pyranometeret. Data ble slått sammen for de tidspunktene der all data var tilgjengelig.

Kapittel 5

Utviklede modeller for simulering av værparametere

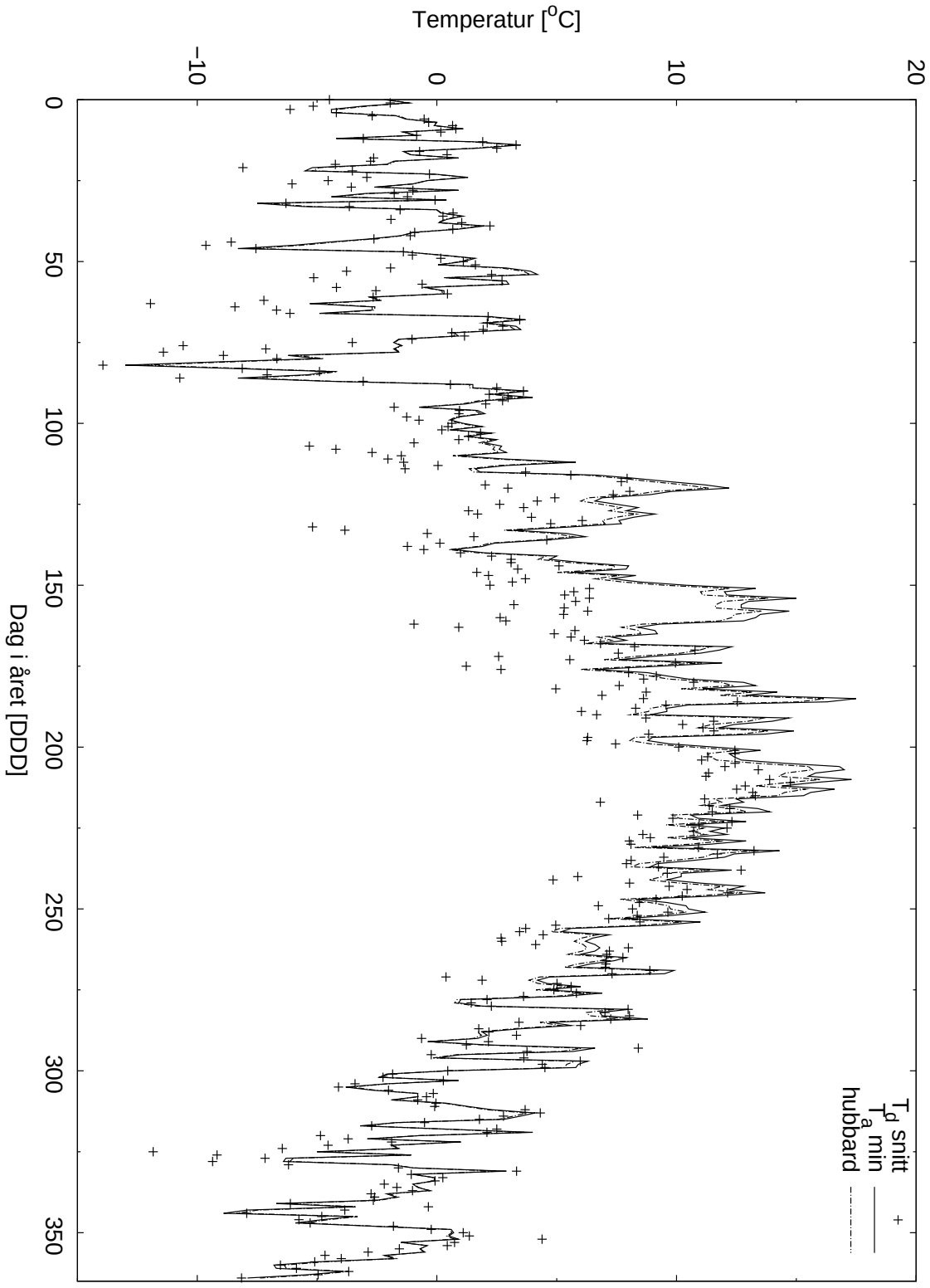
Implementering av kjølepotensiale krever nye modeller og værparametere. Disse finnes ikke i programmet fra før, og det er derfor utviklet nye metoder for å finne disse verdiene. Nedenfor følger en beskrivelse av modellene som er utviklet for automatisk generering av værparametere.

5.1 Duggpunktstemperaturer for Oslo

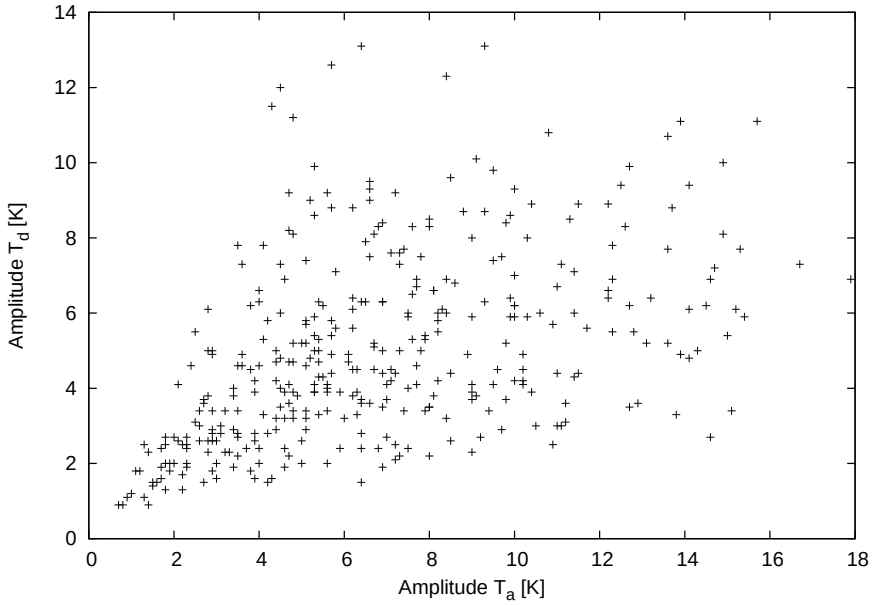
Duggpunktstemperatur bestemmer atmosfærisk klarværsemittans når likn. (2.10) benyttes. Under følger en analyse av hvordan duggpunktstemperaturen oppfører seg gjennom året.

Gjennomsnittlig døgnlig duggpunktstemperatur er plottet for Oslo gjennom et helt år fra og med 1. 1.2008 til og med 31.12.2008. Denne tidsutviklingen er vist i figur 5.1. Sammen med gjennomsnittlig T_d er også to modeller vist. Heltrukken linje viser verdier beregnet med likn. (2.29), mens stiplet linje tilsvarer modellen presentert av Hubbard et al.(Hubbard, Mahmood, and Carlson 2003) vist i likn. (2.31).

Figur 5.1 viser tydelig at ingen av modellene gir gode verdier for gjennomsnittlig T_d mellom dag 100 og 175. Her ligger gjennomsnittlig T_d gjevnt ~5 K under det modellene estimerer. Modell (2.31) ser likevel ut til å gi et bedre estimat. Det ser ut til å være en tendens til at $T_{a,min}$ nærmer seg T_d i større grad for minkende T_a



Figur 5.1: Målte og modellerte verdier for T_d gjennom ett år i Oslo



Figur 5.2: Døgnlig amplitude i T_d mot døgnlig amplitude i T_a

enn for økende. Etter dag 200 gir begge modellene gode verdier for T_d.

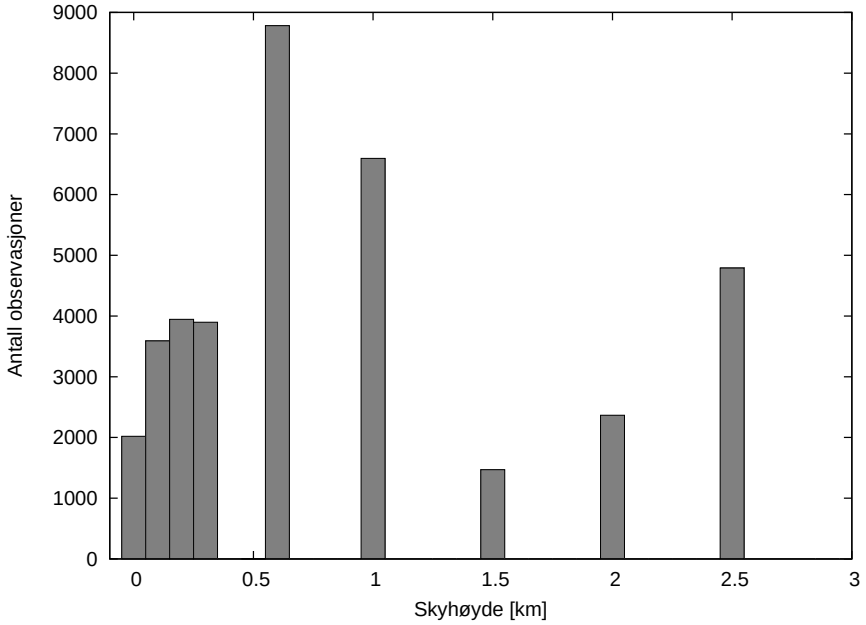
Som figur 5.2 viser er det ingen klar sammenheng mellom døgnlig temperaturspenn for T_d og T_a. Det lar seg dermed ikke gjøre å finne en tidustvikling for T_d gjennom døgnet. I simuleringsprogrammet er T_d satt konstant. Det betyr at klarværemittansen blir lik gjennom hele døgnet.

5.2 Skyhøyder for Oslo

Skyhøydedata for hver time i perioden år 2000 til 2008 er tatt med i analysen. Skyhøydene som blir målt av Meteorologisk Institutt er oppgitt i intervall, 0-50 m, 50-100 m, 100-200 m, 200-300 m, 300-600 m, 600-1000 m, 1000-1500 m, 1500-2000 m, 2000-2500 m og over 2500 m (Meteorologisk Institutt 2007).

Den totale fordelingen av skyhøyder gjennom alle årene i perioden er vist som histogram i figur 5.3. Det er tydelig at observasjonene domineres av lave skyhøyder, mens det befinner seg en topp ved skyhøyde på 2500 m.

Sannsynlighetsfordelinger for skyhøyden den neste timen er vist i tabell 5.2, og er kategorisert etter skøyhøyden den foregående timen. Det er naturlig at skylaget



Figur 5.3: Histogram over observerte skyhøyder i Oslo fra og med år 2000 til og med år 2008.

Tabell 5.1: Sannsynlighetsfordelinger for skyhøyder. Ved en gitt skyhøyde er en diskret sannsynlighetsfordeling for skyhøden den neste timen gitt i %. Data er fra Oslo i årene 2000-2008 og er hentet fra (Meteorologisk Institutt 2009a)

Forrige z_h	Sannsynlighetsfordeling for neste skyhøyde (%)								
z_h (km)	$< 0{,}1$	0,1	0,2	0,3	0,6	1,0	1,5	2,0	$\geq 2{,}5$
$< 0{,}1$	62,38	18,91	7,23	4,06	2,48	1,34	0,54	1,09	1,98
0,1	11,09	58,74	17,86	6,58	2,98	0,92	0,33	0,50	1,00
0,2	3,60	16,58	54,73	15,18	6,84	1,22	0,25	0,51	1,09
0,3	1,36	5,41	14,41	55,48	17,47	2,98	0,59	1,10	1,21
0,6	0,51	1,05	2,64	6,11	76,88	6,93	0,98	2,10	2,79
1,0	0,30	0,74	1,27	1,93	7,69	73,35	3,18	4,81	6,73
1,5	0,48	0,54	1,29	1,50	5,85	16,88	54,46	8,10	10,89
2,0	1,18	0,89	1,69	2,41	5,45	12,60	5,75	54,69	15,34
$\geq 2{,}5$	1,38	1,36	1,29	1,59	4,17	7,91	3,78	7,28	71,24

befinner seg i en stabil høyde, noe som også viser seg ved at sannsynligheten er størst for at skyhøyden den neste timen er den samme som for foregående time.

Beregning av skyhøyder i SolEC benytter seg av fordelingene gitt i tabellen direkte. Det er altså ikke forsøkt å finne sannsynlighetsfordelinger som kan beskrive dataene.

5.3 Værperioder

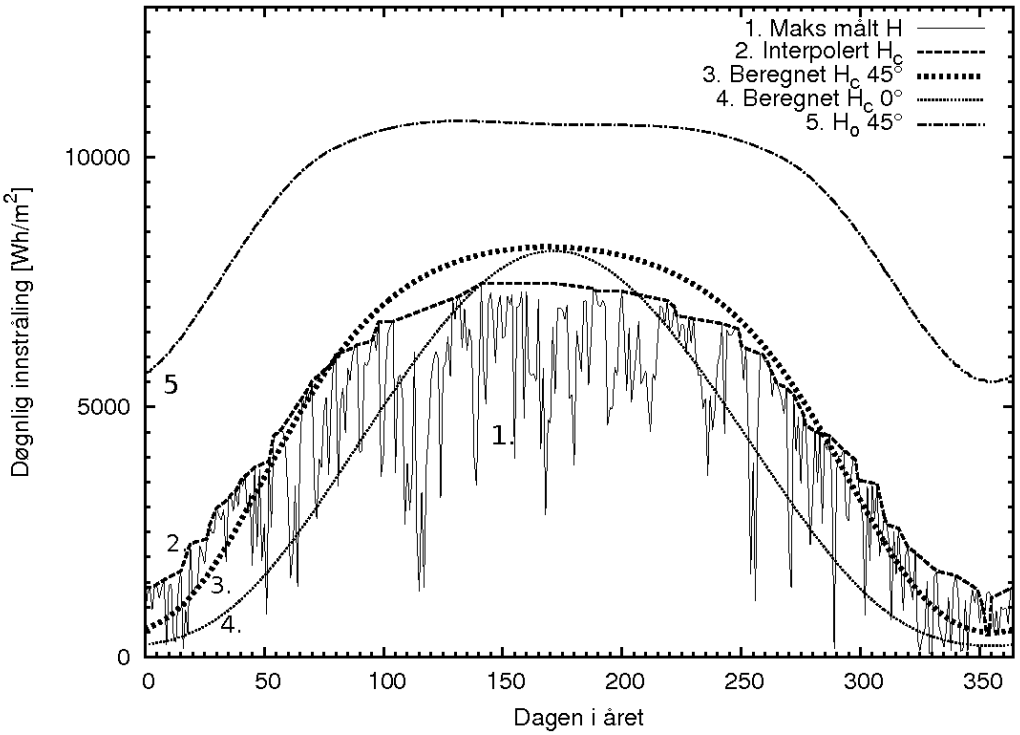
Det er funnet flere interessante resultater av beregninger utført på SOLIS-dataene. For det første er klarværsinnstråling funnet for hver målestasjon. Fem figurer som viser klarværsstråling er tatt med i resultatene, tre er vist nedenfor, men de siste to er med i kapittelet om usikkerhet (seksjon 7.3).

Norge er her delt inn i fem klimatiske soner, og fordelinger for værperiodelengder er funnet for hver sone.

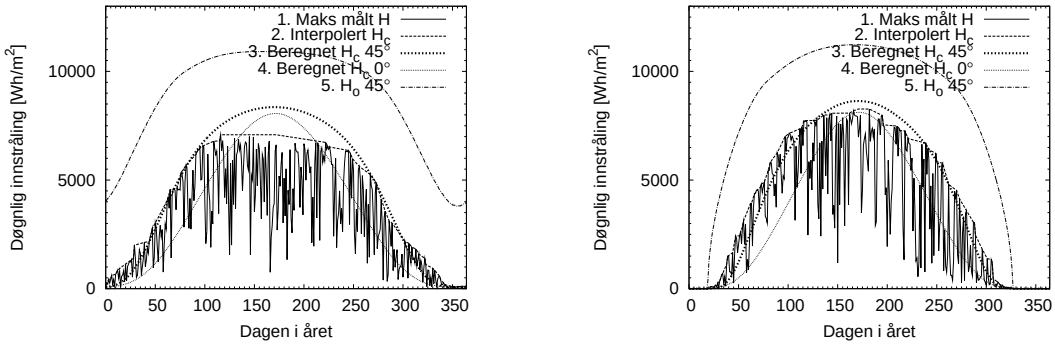
5.3.1 Beregning av klarværsinnstråling

Klarværsinnstråling blir benyttet til å beregne klarværsindeks, slik at periodelengder kan beregnes. Klarværsinnstrålingen er funnet ved de to metodene beskrevet i seksjon 4.3 og resultatene av begge metoder er vist i figur 5.4, der målinger fra Bjertnes videregående skole i Nittedal ($\phi = 60{,}04°$ N og $L = 10.88°$ Ø) er brukt som eksempel. Det er kun de største innstrålingsverdiene for en gitt dag som blir plottet. Klarværsinnstråling beregnet ved HDKR-modellen er markert med 1 (tykk stiplet linje) og interpolerte klarværsverdier er markert med 2. Ekstraterrestriell innstråling (5) mot skråpanel og beregnet klarværsinnstråling mot horisontalflate (4) er tatt med til sammenlikning.

Figur 5.5 viser tydelig hvordan total innstråling forandrer seg jo lenger nord målestasjonen befinner seg. Midnattssolen gjør at maksimal innstråling mot skråflate øker, mens det er et klart skille for mørketid. Levanger ($\phi = 63{,}74°$ N og $L = 11{,}29°$ Ø) ser ut til å være akkurat på grensen til å ikke ha noe solenergipotensiale ved vintersolverv.



Figur 5.4: Klarværsinnstråling for Nittedal i Akershus. Maksimale verdier for innstråling i perioden er vist som heltrukken linje. Målingene er fra Bjertnes videregående skole i perioden 01.01.1995 til 28.0.1999



(a) Levanger **(b)** Troms

Figur 5.5: Klarværsinnstråling for områder lenger mot nord. Målinger utført av (5.5a) Levanger videregående skole 1.12.1994- 28.2.1999 og (5.5b) Breivika tekniske fagskole 1.1.1995 - 28.2.1999

Tabell 5.2: Antall perioder og manglende perioder for gode periodelengder i hvert område

Område	Antall perioder	Antall manglende perioder
Sørlandet	1059	62
Østlandet	1474	56
Vestlandet	218	7
Midt-Norge	1218	55
Nord-Norge	1252	48

Tabell 5.3: Antall perioder og manglende perioder for dårlige periodelengder i hvert område

Område	Antall perioder	Antall manglende perioder
Sørlandet	1063	59
Østlandet	1433	97
Vestlandet	216	15
Midt-Norge	1183	80
Nord-Norge	1221	78

5.4 Periodelengder

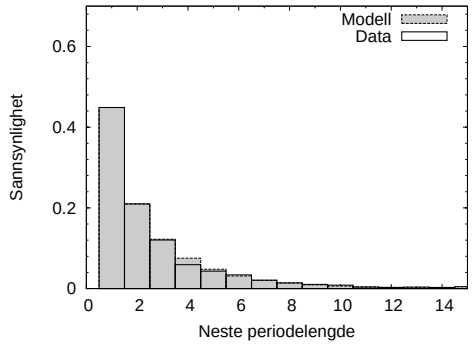
I tabellene 5.2 og 5.3 er det totale antall perioder med godt og dårlig vær, som er funnet for hver landsdel, vist sammen med antallet perioder der data mangler. De viser det totale statistiske grunnlaget for hvert område. Alle områder har data for over 2000 værperioder, bortsett fra Vestlandet. Grunnen til det lille datasette for Vestlandet er at det kun befant seg to videregående skoler som var med på SOLIS-prosjektet i området.

Under er fordelingen av periodelengder for hvert område vist i stolpediagram. I hvert stolpediagram er både weibullmodellen samt målte verdier for periodelengder vist. Innrammede, hvite solper representerer modellen, men fulte, grå stolper representerer målte periodelengder. Ved siden av hvert diagram er det også vist et weibulldiagram, både for modell og data. Modellen er vist med stiplet kurve og dataene med heltrukken linje.

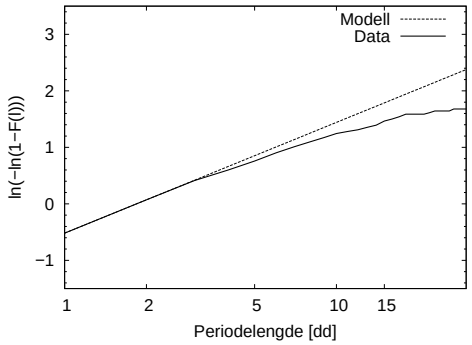
Fordelingene er presentert slik at de viser sannsynligheten for neste periodelengde ved en gitt værtype. Dersom foregående værperiode var dårlig vil neste værperiode være bra. Sannsynligheten for lengden til den neste gode værperioden kan da leses av i fordelingen for gode værperioder. Er foregående værperiode bra vil sannsynligheten for neste dårlige værperiode kunne leses av i fordelingen for dårlige

værperioder.

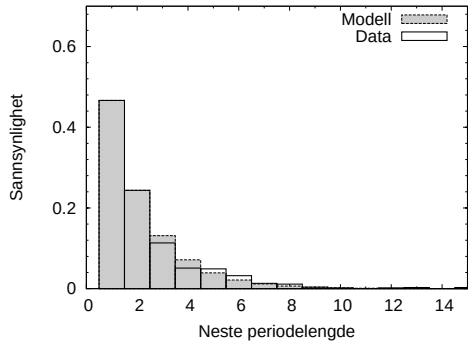
For hvert stolpediagram er det også tatt med et Weibull-diagram. Dersom datakurven forholder seg som en rett linje i Weibull-diagrammet, er værperiodene weibullfordelt. Dersom fordelingen har en avtakende hale, vil paretofordelingen være en bedre modell. Økende $\ln(-\ln(1-F(l)))$ gir økende $F(l)$, som medfører at andelen periodelengder med lengde $L < l$ vokser. At modellen krummer betyr en dermed en økende sannsynlighet for lange værperioder.



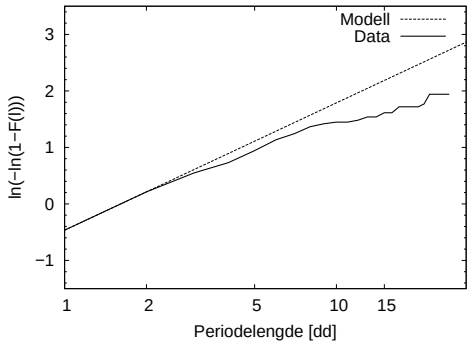
(a) Fordeling av gode periodelengder



(b) Weibulldiagram av 5.6a



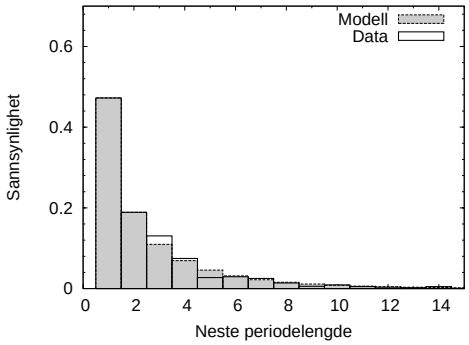
(c) Fordeling av dårlige periodelengder



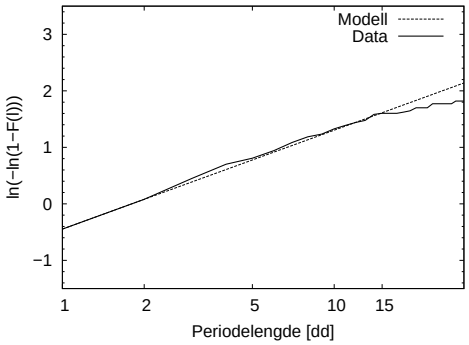
(d) Weibulldiagram av 5.6c

Figur 5.6: Fordelinger og weibulldiagram for periodelengder på Sørlandet

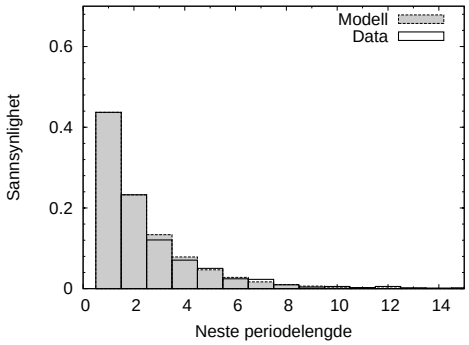
Figur 5.6a og 5.6c viser de totale fordelingene for periodelengder for godt og dårlig vær på Sørlandet. Det ser ut til å være litt større sannsynlighet for dårlige periodelengder på en og to dager enn for gode periodelengder. Ellers er lengdene på periodene omtrent like for både godt og dårlig vær på Sørlandet. Litt under halvparten av periodene er på 1 dag. Weibulldiagrammene 5.6b og 5.6d viser tydelig at weibullmodellen ikke er tilfredsstillende verken for gode eller dårlige perioder. Modellen avviker allerede ved periodelengde på tre dager.



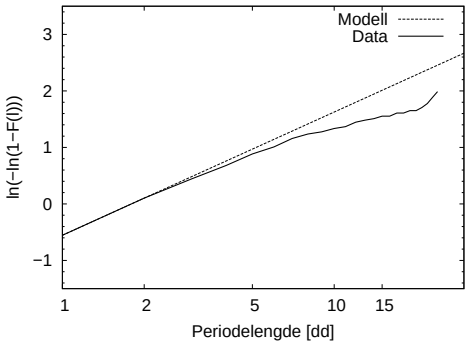
(a) Fordeling av gode periodelengder



(b) Weibulldiagram av 5.7a



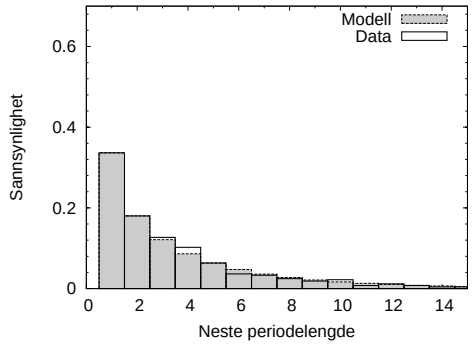
(c) Fordeling av dårlige periodelengder



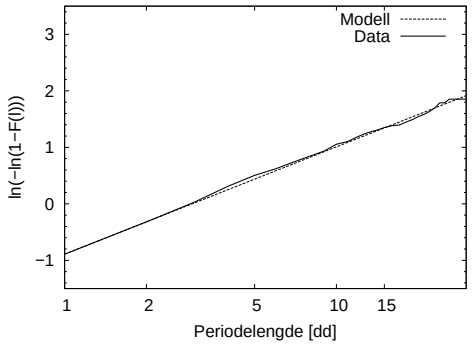
(d) Weibulldiagram av 5.7c

Figur 5.7: Fordelinger og weibulldiagram for periodelengder på Østlandet

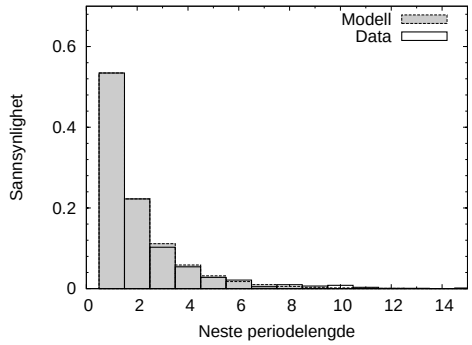
Fordelingen av periodelengder for Østlandet er vist i figur 5.7. Ved å sammenlikne figurene 5.7a med 5.6a og figur 5.7c med 5.6c er det ingen store forskjeller å spore. Periodelengdene ser ut til å være noe kortere på Østlandet, både for godt og dårlig vær. En vesentlig forskjell er at fordelingen for Østlandet ser ut til å følge weibullmodellen bedre. Spesielt fordelingen for gode periodelengder ser ut til å følge modellen helt til periodelengder på 14 dager. Fordelingen for dårlige periodelengder ser ut til avvike allerede ved 4 dager.



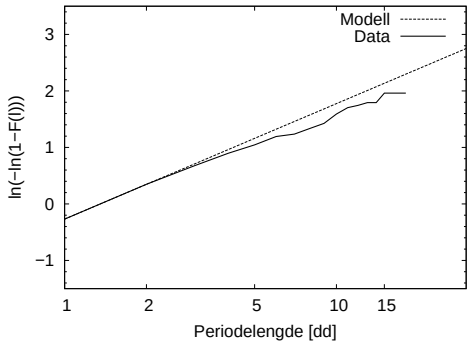
(a) Fordeling av gode periodelengder



(b) WPP-plott av 5.8a



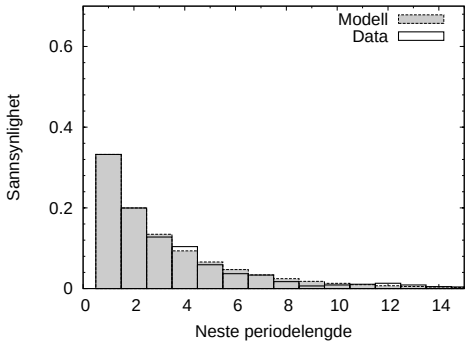
(c) Fordeling av dårlige periodelengder



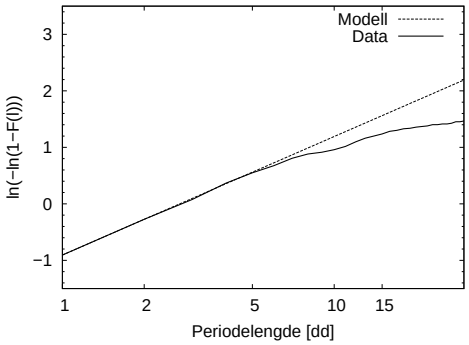
(d) WPP-plott av 5.8c

Figur 5.8: Fordelinger og weibulldiagram for periodelengder i Midt-Norge

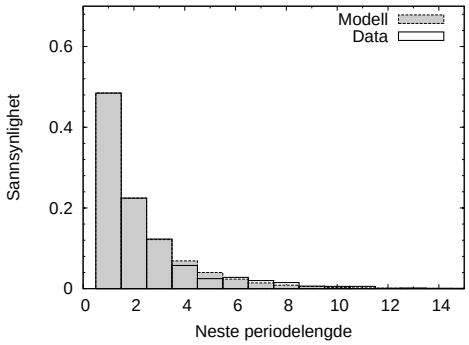
Periodelengdefordelingene for Midt-Norge er vist i figur 5.8. Her er det en klar forskjell mellom lengdene for godt og dårlig vær. Som vist i figur 5.8a er sannsynligheten relativt liten for korte gode periodelengder. Det ser ut til at området er preget av lange perioder med godvær. Stikk motsatt er det med fordelingen for dårlige periodelengder vist i figur 5.8c. Sannsynligheten for kun en dårlig dag er over 50%. Samtidig er det få lange perioder med dårlig vær. Som en kan se i figur 5.8b stemmer weibullfordelingen helt overens med data for gode periodelengder. Weibullmodellen stemmer også for dårlige perioder på opptil fem dager, men faller for lengre periodelengder, som vist i figur 5.8d.



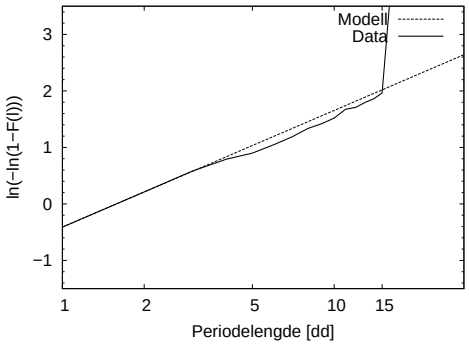
(a) Fordeling av gode periodelengder



(b) WPP-plott av 5.9a



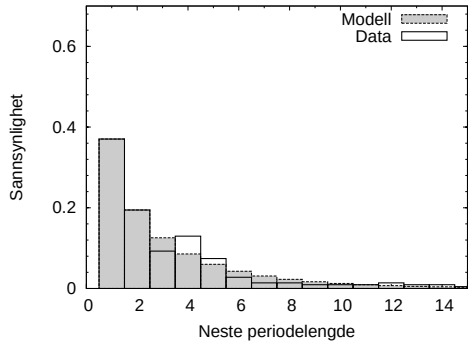
(c) Fordeling av dårlige periodelengder



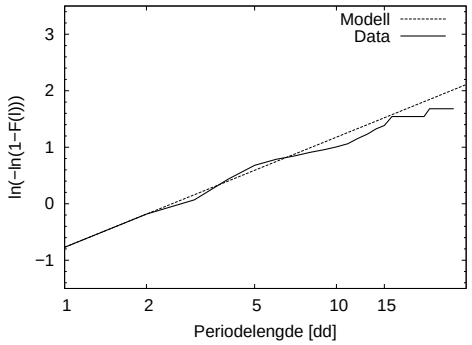
(d) WPP-plott av 5.9c

Figur 5.9: Fordelinger og weibulldiagram for periodelengder i Nordnorge

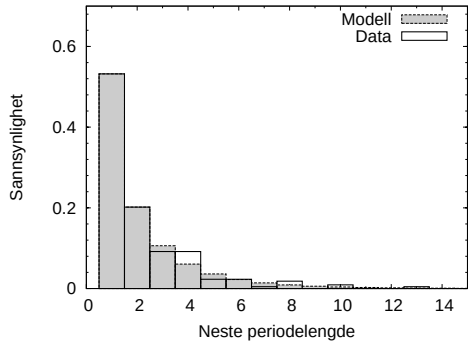
Fordelingene for Nord-Norge er vist i figur 5.9. Også dette området er preget av lange godværsperioder. Ved å sammenlikne weibulldiagrammene 5.9b og 5.9d med de tilsvarende weibullfordelingene for Østlandet, kan man se at modellen som passer best er stikk motsatt. Gode periodelengder er nå paretofordelt, mens dårlige periodelengder er weibullfordelt.



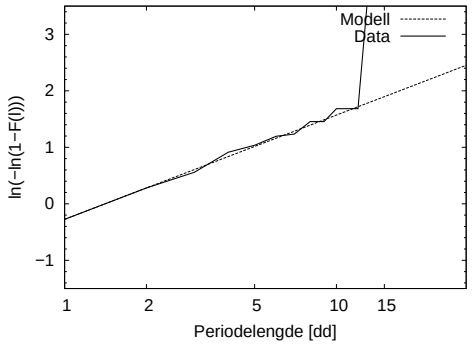
(a) Fordeling av gode periodelengder



(b) WPP-plott av 5.10a



(c) Fordeling av dårlige periodelengder



(d) WPP-plott av 5.10c

Figur 5.10: Fordelinger og weibulldiagram for periodelengder på Vestlandet

Fordelingene for periodelengder på Vestlandet vist i figur 5.10 er tatt med for fullstendighetens skyld. Datasettet er kun på 216 perioder for dårlig vær og 218 perioder for godt vær. At datasettet er for lite kan man tydelig se i histogrammene 5.10a og 5.10c. Modellen overestimer sannsynligheten for periodelengder på 3 dager og underestimer sannsynligheten for periodelengder på 4 fire dager for begge værtyper. Det er tydelig at dette skyldes varians.

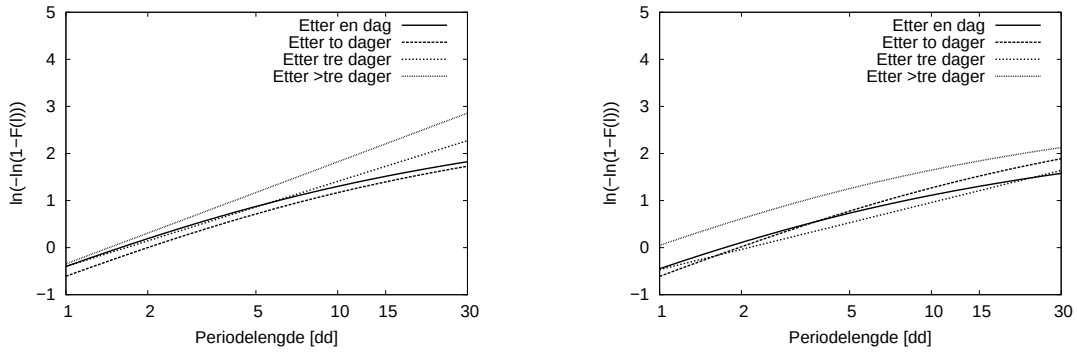
5.5 Kategorisering av periodelengder

Fordelingen for periodelengder er kategorisert etter antall dager i foregående periode. Det er funnet godt nok statistisk grunnlag for å skille mellom datasett for hver periodelengde opp til tre dager, men for fire dager er datasettet for tynt. Siden sannsynligheten for en periodlengde faller eksponensielt er det lite data for hver enkelt periodelengde, og derfor er data slått sammen for alle periodelengder over tre dager.

Tabell 5.4: Parametere for fordeling av værperioders varighet etter landsdel og antall dager i foregående periode. Ved en gitt værtype og periodelengde kan fordelingen av lengder for neste periode finnes ved å benytte disse parameterne i sin respektive fordeling. Kun parametere til den modellen som passer best er vist

			Weibull		Generalisert pareto	
Landsdel	Værtype	Dager	q	β_w	c	l_0
Sørlandet	bra	1			0,3552	1,322
		2			0,1502	1,665)
		3	0,512	0,786		
		> 3	0,492	0,941		
	dårlig	1			0,5317	1,314
		2			0,2586	1,714
		3	0,534	0,619		
		> 3			0,3952	1,089
Østlandet	bra	1	0,555	0,865		
		2			0,4569	1,394
		3			0,3525	1,812
		> 3	0,553	1,033		
	dårlig	1	0,527	0,625		
		2			0,2264	1,775
		3	0,508	0,807		
		> 3	0,443	0,751		
Vestlandet	bra	1	0,476	0,628		
		2	0,595	1,082		
		3	0,474	1,060		
		> 3	0,387	0,821		
	dårlig	1	0,664	0,686		
		2			0,6181	1,319
		3	0,700	1,163		
		> 3	0,622	1,351		
Midt-Norge	bra	1			0,1302	1,244
		2	0,401	0,804		
		3			0,2522	1,327
		> 3	0,468	0,800		
	dårlig	1	0,685	0,884		
		2			0,2505	2,631
		3			0,2522	2,594
		> 3	0,569	0,624		
Nord-Norge	bra	1	0,555	0,865		
		2	0,564	1,074		
		3	0,608	0,916		
		> 3			0,1706	1,313
	dårlig	1			0,4328	1,977
		2			0,5724	1,897
		3			0,295	2,575
		> 3			0,35	2,3 *

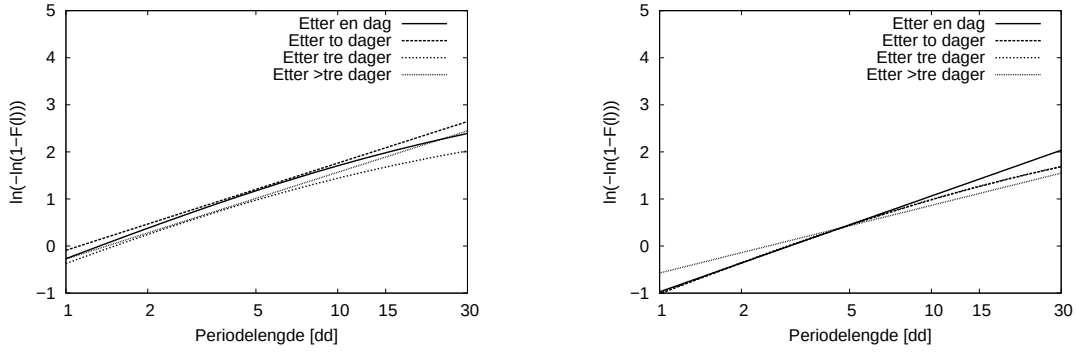
I hvert enkelt tilfelle er det ut ifra et weibulldiagram bestemt om fordelingen av periodelengder er paretofordelt eller weibullfordelt. Tabell 5.4 viser en oversikt over parameterne for hver fordeling. Kun parameterne til den modellen som passer best er tatt med.



(a) Gode værperioder **(b)** Dårlige værperioder

Figur 5.11: Weibulldiagram for kategoriserte periodelengder etter lengden på foregående værperiode på Sørlandet

Som et eksempel på bruk av parameterene gitt i tabell 5.4, er kategoriserte fordelinger for periodelengder på Sørlandet fremstilt i figur 5.11. Figurene 5.11a og 5.11b viser for både godt og dårlig vær en tendens til at periodelengden blir kort etter en lengde på foregående periode over tre dager. I dette tilfellet vil det være en større andel lange dårlige værperioder i forhold til gode. Det virker som neste periodelengde blir kortere ved lengre foregående periodelengde.



(a) Gode værperioder **(b)** Dårlige værperioder

Figur 5.12: Weibulldiagram for kategoriserte periodelengder etter lengden på foregående værperiode i Midt-Norge

Et eksempel for Midt-Norge er vist i figur 5.12. Her er tendensen motsatt. Økende lengde på foregående periode medfører økende lengde på neste periode. Med andre

ord vil skiftende vær gi økt sannsynlighet for været forblir skiftende. Stabilt, godt vær gir økt sannsynlighet for stabilt, dårlig vær. Merk likevel at sannsynligheten for $L_p \leq 3$ er overveldene.

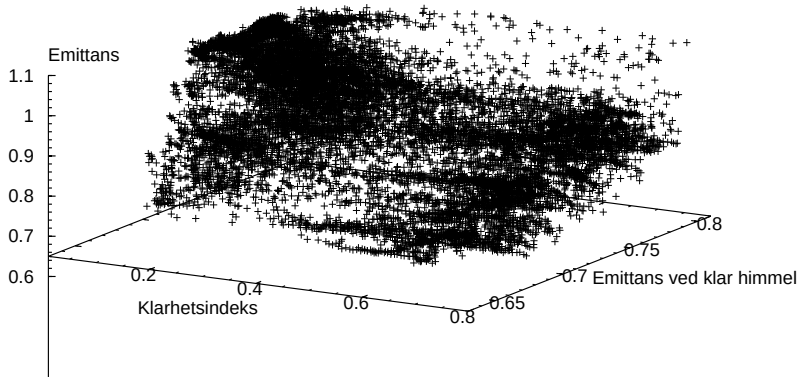
I programmet blir simulering av klarværsindeks utført med fordelingsparameterne i tabell 5.4. Brukeren trenger da kun å huke av for hvilken landsdel det ønskes å simulere for.

5.6 Atmosfærisk emittans

Modellen det er forsøkt å finne for atmosfærisk emittans ϵ_{atm}, er en funksjon av både klarhetsindeksen, k, og klarværsemittansen, ϵ_0, beregnet ved likn. (2.10). Totalt var det 14623 målepunkter av atmosfærisk innstråling som samsvarte tidsmessig med innstrålingsmålingene som var foretatt hvert andre minutt. Dette medførte en tidsforskjell på opptil 16 sekunder mellom de to målingene. I figur 5.16 er $\epsilon_{atm} = \frac{R_{atm}}{\sigma T_a^4}$ plottet i tredimensjonale grafer som funksjon av ϵ_0 og k. I disse grafene er alle punktene tatt med. I hele tidsperioden manglet det målinger for 31 juli til 6 august og 14. - 15 august. T_d har i måleperioden stort sett hatt verdier $T_d \geq 0$ °C. Dette har medført at alle målingene her er gitt ved $\epsilon_0 \geq 0.65$.

Figur 5.13 og 5.14 viser ϵ_{atm} ved to forskjellige gjennomsnitt av klarhetsindeksen. I figur 5.13 er det benyttet momentanverdier av k. Sommeren har vært preget av klart vær i slutten av juni, mens det stort sett har vært overskyet og regn utover juli. Det er tydelig at de fleste verdiene er målt ved veldig klart og veldig skyet vær. For verdier for k mellom 0,3 og 0,6 finnes det færre målepunkter, og i dette intervallet har ϵ_{atm} en mer eller mindre konstant verdi. Været i måleperioden har med andre ord vært spesielt stabilt, og det foreligger lite data for skiftende værtyper. Figur 5.15 viser gjennomsnitt av k over en time. Gjennomsnittet er finnet for en halv time frem og tilbake i tid. Det er tydelig at verdiene dette plottet i større grad fordeler seg jevnt i k_{av}-retningen.

Ved første øyekast kan det se ut til at emittansen går som en eksponensialfunksjon av k, der ϵ_{atm} minker for økende klarhetsindeks. Det blir tydeligere når gjennomsnittlig k_{av} blir brukt. Som forklart i seksjon 3.2 gir gjennomsnittlig klarhetsindeks et bedre estimat på skydekket. Dette er en naturlig konsekvens av at skyene ikke alltid er isotropt fordelt over himmelkulen. I ekstreme tilfeller kan det hende at den eneste skyen på himmelen befinner seg foran Sola, og motsatt, det eneste

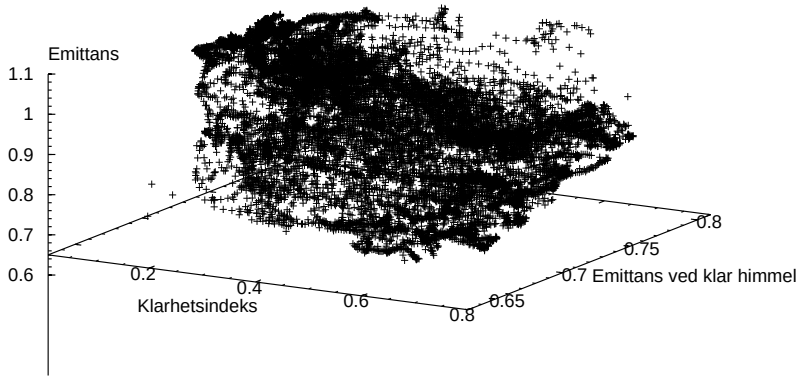


Figur 5.13: Målte verdier av ϵ_{atm} som funksjon av momentan k og ϵ_0

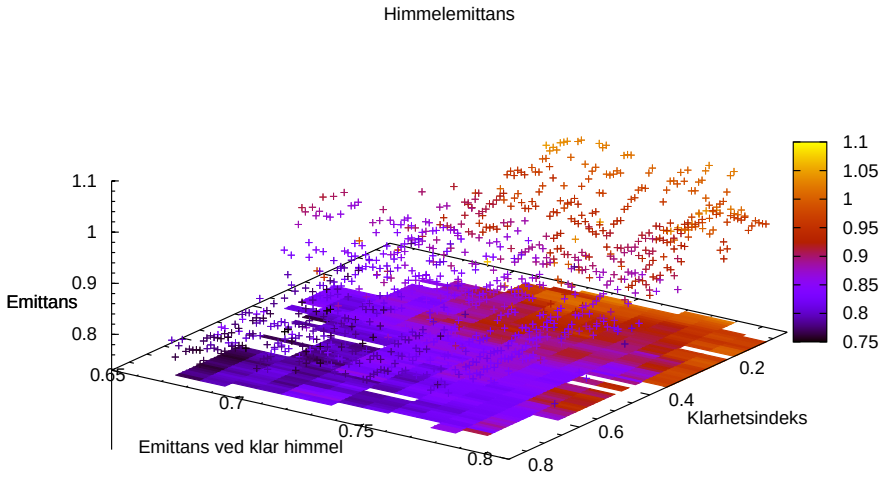
gløttet i skydekke kan være nok til å slippe hele solstrålen gjennom.

Det er valgt å benytte timeverdier for gjennomsnittlig klarhetsindeks, k_{av}, som representant for skydekket. Dette er et naturlig valg, siden timevise k_{av} automatisk genereres i programmet. For å finne en empirisk relasjon for $\epsilon_{atm}(k_{av}, \epsilon_0)$ er k_{av}, ϵ_0-planet del opp i et gitter med $\Delta\epsilon_0 = 0,01$ og $\Delta k_{av} = 0{,}02$, noe som gitt totalt 40 x 17 soner. Denne inndelingen er illustrert i figur 5.15. I hver sone er gjennomsnittsverdien av alle målepunkter funnet og plottet. Disse verdiene er også projektert ned på (k_{av}, ϵ_0)-planet som en fargenyanse mellom mørkeblå og gul. Mørkeblå farge tilsvarer en kjølig, klar atmosfære, mens gul farge tilsvarer en varm, og skyet atmosfære. Det er tydelig ut fra figuren at maksimalt kjølepotensiale oppnås ved høye k_{av} og ϵ_0. Ved ekstremt overskyet og fuktig vær vil emittansen stige til over 1 og kjølepotensialet forsvinner.

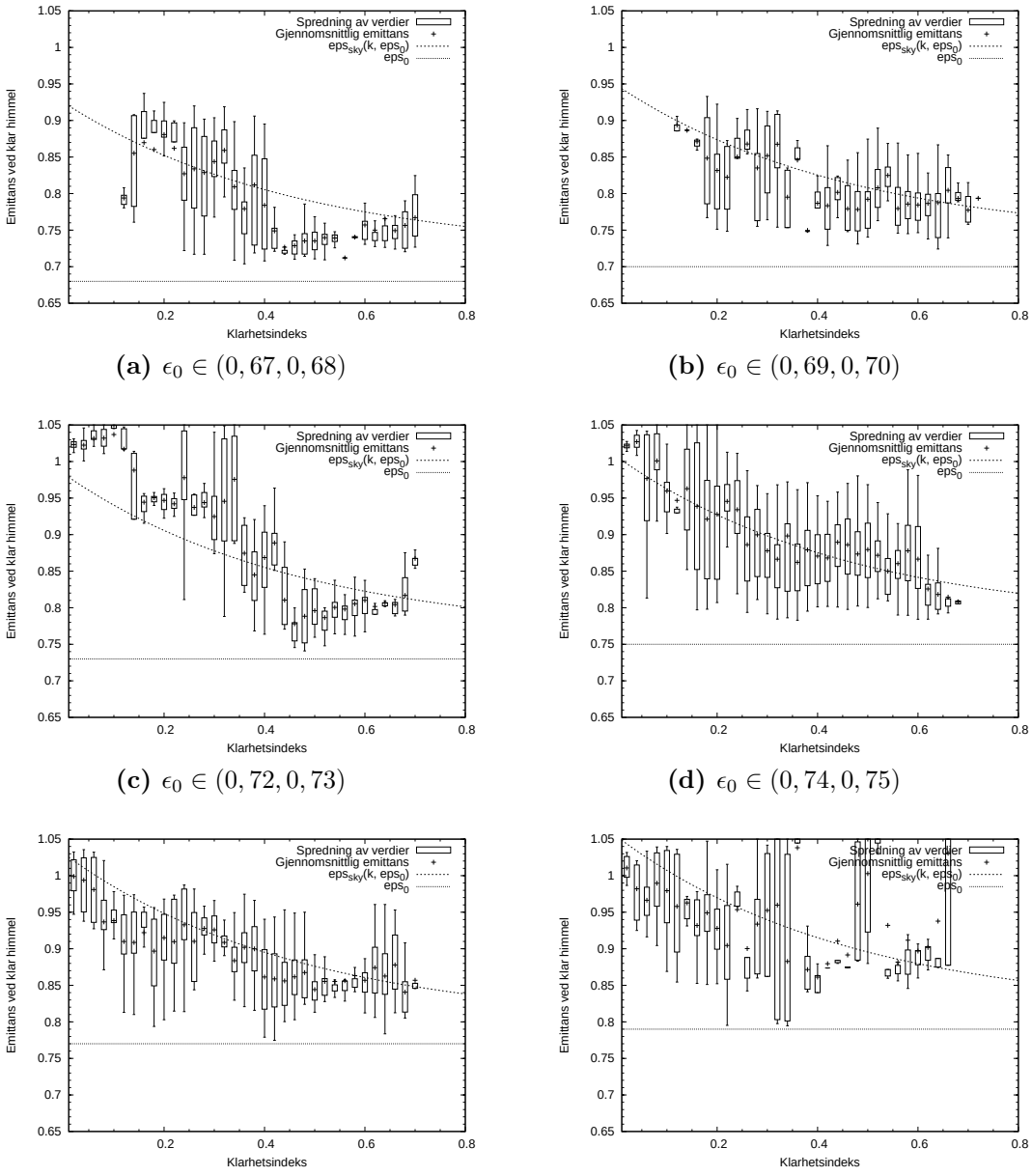
Et utvalg av plan med verdier av ϵ_0 i intervall på $\Delta\epsilon_0 = 0,01$ er plottet i figur 5.16. Dette er box-whisker-plots (Montgomery and Runger 2003), hvor 50% av punktene i fordelingene for hvert punkt er avgrenset av boksen. Med andre ord strekker boksen seg fra første til tredje fjerdedelsverdi. Whiskeren over og under boksen strekker seg opptil halvannen av høyden til boksen i begge retninger og beskriver på en god måte spredingen av verdiene. Kurvetilpasningen er utført ved hjelp av fit-funksjonen i Gnuplot (Williams and Kelley 2007). Flere eksponensialfunksjoner



Figur 5.14: Målte verdier av atmosfærisk emittans som funksjon av gjennomsnittlig klarhetsindeks over en time og klarværsemittans



Figur 5.15: (k_{av}, ϵ_0)-planet inndelt i et gitter på 40x17 soner. I hver sone er snittet av ϵ_{atm} funnet og plottet. Mørkeblå soner tilsvarer en klar himmel uten skyer mens røde soner betyr overskyet vær og en høy temperatur i atmosfæren

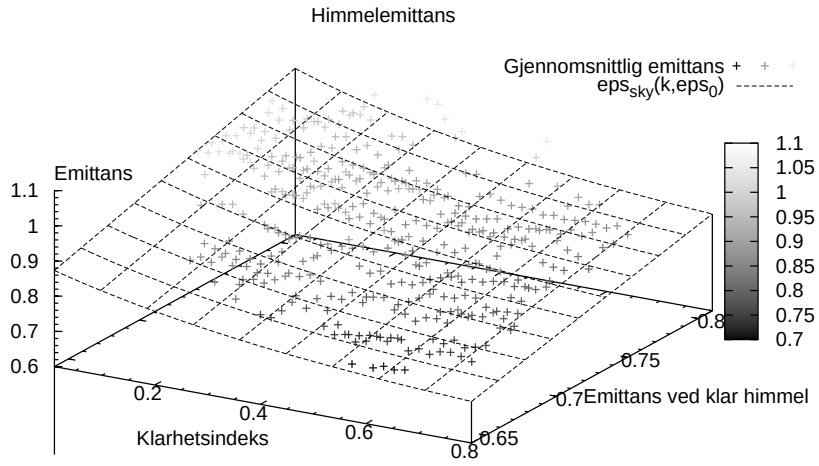


(a) $\epsilon_0 \in (0,67, 0,68)$ **(b)** $\epsilon_0 \in (0,69, 0,70)$

(c) $\epsilon_0 \in (0,72, 0,73)$ **(d)** $\epsilon_0 \in (0,74, 0,75)$

(e) $\epsilon_0 \in (0,76, 0,77)$ **(f)** $\epsilon_0 \in (0,78, 0,79)$

Figur 5.16: Box-whisker-plott med gjennomsnittlige verdier av ϵ_{atm} er plottet i planet med ϵ_0 som ligger i intervall på $\Delta\epsilon_0 = 0,01$ likn. (5.1).



Figur 5.17: Gjennomsnittlige målte verdier av atmosfærisk emittans er plottet sammen med den empiriske funksjonen (5.1)

ble forsøkt tilpasset dataene, og valget falt til slutt på

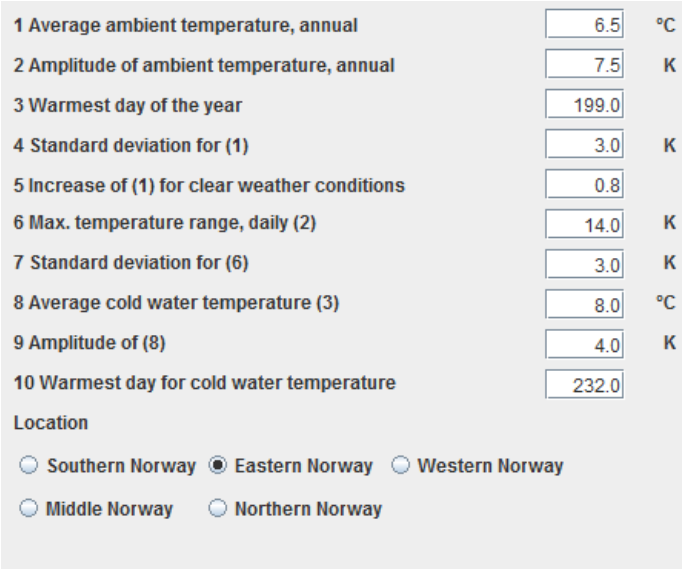
$$\epsilon_{atm}(\epsilon_0, k_{av}) = \epsilon_0(c_1 e^{-c_2 k_{av}} + c_3) + c_4 \tag{5.1}$$

der parameterene har verdiene $c_1 = 0{,}30 \pm 0{,}02$, $c_2 = 2{,}1 \pm 0{,}4$, $c_3 = 0{,}87 \pm 0{,}05$ og $c_4 = 0{,}127 \pm 0{,}03$ med 393 frihetsgrader. At en parameter har stort standardavvik betyr at den kan varieres uten at funksjonen får en vesentlig dårligere tilpasning. For små standardavvik kan selv en liten forandring i parameteren medføre at funksjonen gir et høyt avvik. Likn. (5.1) er plottet i figur 5.17 sammen med gjennomsnittet av de målte verdiene av ϵ_{atm}.

Kapittel 6

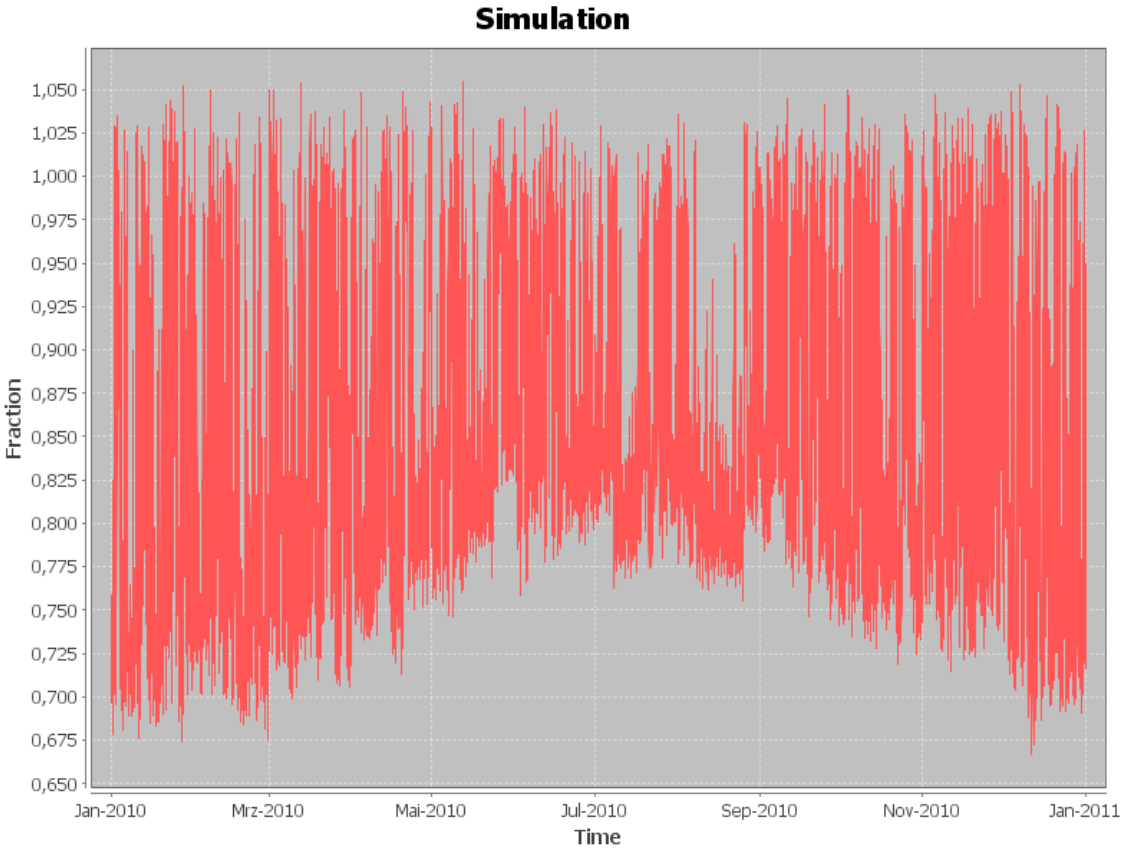
Simuleringsresultater

6.1 Simulering av atmosfærisk emittans



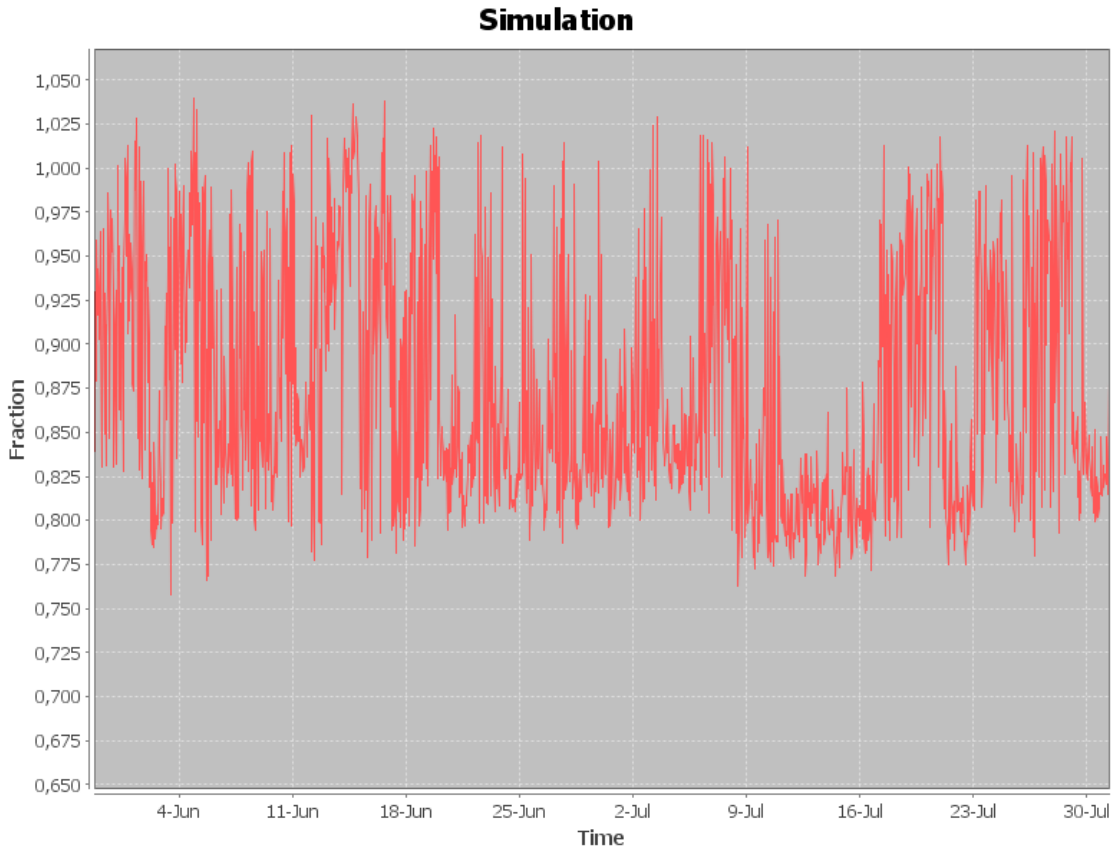
Figur 6.1: Parameterpanel for vær.

Ved hjelp av simuleringsprogrammet er det utført en simulering av atmosfærisk emittans for å vise hvordan variabelen endrer seg gjennom året. Simuleringen er for Østlandet og det er benyttet syntetisk vær. Som input-parametere er standardverdiene benyttet. Gjennomsnittstemperaturen gjennom året er satt til 6,5 °C med standardavvik på 3,0 °C og årlig amplitude på 7,5 °C. Værparameterpanelet som er utbiklet til programmet er vist i figur 6.1.



Figur 6.2: Hemisfærisk emittans simulert med syntetisk vær for et helt år. Simuleringen er utført for Oslo

Figur 6.2 viser tidsutviklingen av atmosfærisk emittans ved bruk av syntetisk vær på Østlandet. Det er tydelig at emittansen øker om sommeren, noe som er naturlig siden klarværsemittansen følger duggpunktstemperaturen. Variasjonen i atmosfærisk emittans ϵ_{atm} er høyest om vinteren siden $\epsilon_{atm} \rightarrow 1$ når andel skydekke $n \rightarrow 1$ uavhengig av verdien på klarværsemittansen. Kjølepotensialet er på sitt høyeste når atmosfærisk emittans er lavest.



Figur 6.3: Utsnitt av figur 6.2 for månedene Juni-Juli

Et utsnitt for sommermånedene juni og juli er vist i figur 6.3. Det kan leses av grafen at emittansen er ustabil. Det skyldes at relasjonen følger klarværsindeksen, men det gjevner seg uto og skal ha liten betydning for simuleringsresultatene dersom korte tidssteg, $\Delta t \leq 60$ min, benyttes.

6.2 Simulering av energibehov

Det er utført behovssimuleringer for de ulike landsdelene. Romoppvarmingsbehovet og kjølebehovet er fremstilt i tabell 6.1. Det er benyttet identisk frø for alle de klimatiske sonene, slik at rekken med tilfeldige tall er lik. Temperaturparameterne er satt som ved simuleringen av atmosfærisk emittans. En eventuell varisjon i utetemperaturen skyldes dermed kun påvirkningen av klarværsindeks. Simuleringene er foretatt med standard behovsparametere vist i figur 3.2.

Merk at Vestlandet i følge disse simuleringene vil ha det største varmebehovet og kjølebehovet.

Tabell 6.1: Behovsimuleringer med identisk tallgeneratorfrø.

Område	Varmebehov (kWh)	Kjølebehov (kWh)
Sørlandet	19165	493
Østlandet	19280	465
Vestlandet	19813	508
Midt-Norge	19278	446
Nord-Norge	19032	244

Kapittel 7

Usikkerhetsanalyse

Det er gjort et forsøk på å validere modellene i oppgaven. Usikkerheten i datagrunnlaget er funnet, og statistiske beregninger for den utviklede emittansmodellen samt modellen for duggpunktstemperatur er utført.

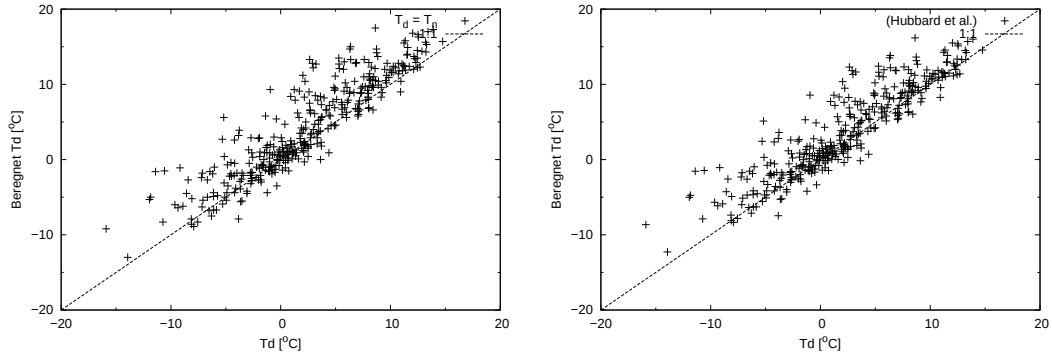
7.1 Meteorologiske data

Både ved utvikling av den atmosfæriske emittansmodellen (likn. (5.1)) og testing av modeller for duggpunktstemperatur, er data for både duggpunktstemperatur og utetemparatur hentet inn. Temperaturmålingene til Meteorologisk Institutt er oppgitt å ha en usikkerhet på ±0,1 °C (Meteorologisk Institutt 2009b).

Usikkerheten i skøyhøyde er irrelevant, siden dataene kun er benyttet til å utvikle sannsynlighetsfordelinger. Likevel er det interessant å observere at to skyhøydeobservasjoner har verdien 500 m. Disse er observert 13:00 17.3.2006 og 10:00 17.3.2006. Det er ukjent hvordan disse verdien har havnet i datasettet, siden 500 m ikke inngår i intervallmålingene (Meteorologisk Institutt 2007). Det er ikke mulig å beregne en statistisk fordeling for neste skyhøyde med kun to punkter, så begge verdiene er fjernet.

7.2 Duggpunktstemperatur

De statistiske metodene beskrevet i seksjon 2.8 ble benyttet for å finne ut hvor godt modellene som beregner T_d yter. Beregnet mot observert T_d er fremstilt gra-



(a) (2.29) **(b)** Likn (5.1)

Figur 7.1: Modellert mot målt T_d. (7.1a) T_d tilnærmet med $T_{a,min}$, (7.1b) modell utviklet av Hubbard et al. (2003)

fisk i spredningsdiagram i figur 7.1 for begge modellene gjennom hele året totalt sett. Modellen til. (2.31) vist i figur 7.1b synes å vise mindre spredning enn modellen som kun bruker $T_{d,m} = T_{a,n}$ vist i figur 7.1a.

Som poengtert i (Legates and McCabe 1999) gir ikke metodene for treffsikkerhetsberegninger god informasjon for store datasett med stor variasjon i observerte verdier. Spesielt gjelder det for data som er målt over lange tidsperioder. Beregningene er derfor gjort månedsvis. Resultatene ved bruk av $T_d = T_{a,n}$ for hver måned er vist i tabell 7.1. I tabell 7.2 er månedsvise statistiske beregninger for bruk av modell (2.29) vist.

Resultatet av effektivitetskoeffisenten viser at begge modellene gir negativ verdi fra og med måned 4 til og med 6 i Oslo. I denne perioden ville gjennomsnittlig månedlig T_d gi et bedre resultat. Fra og med måned 7 stemmer modellen overens med målt verdi, da utetemperaturen begynner å synke igjen.

7.3 Periodelengder

Usikkerheten i periodelengdene avhenger av usikkerheten i klarværsindeksen beregnet ut fra klarværsinnstrålingen. Det er vanskelig å gi noe overslag over usikkerheten i klarværinnstrålingen når detaljer angående målingene og omgivelsene ikke er kjent. Det er likevel ikke så viktig å kjenne til absoluttverdien av innstrålingen siden kun forholdet mellom innstråling og klarværinnstråling er beregnet.

Tabell 7.1: Statistiske beregninger for bruk av $T_d = T_{a,n}$ som duggpunktstemperaturen. Verdiene er funnet for hver måned i 2008.

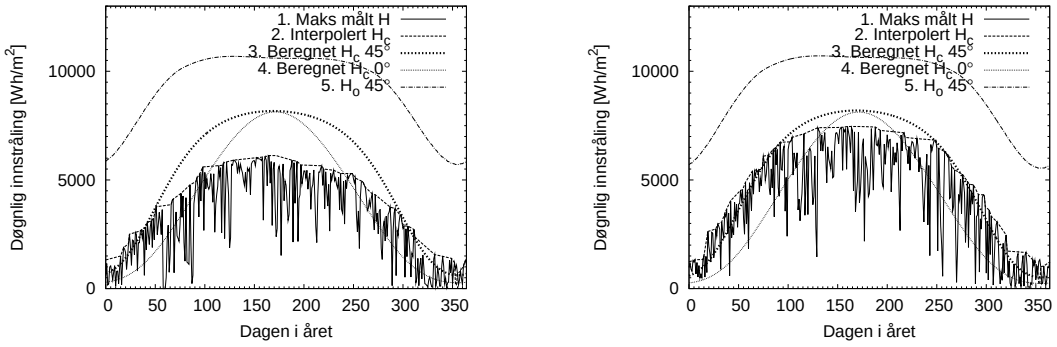
Måned	MAE[1](K)	RMSE[2](K)	r[3]	R^2 [4]	E[5]	d[6]
1	1,58	2,04	0,69	0,47	0,39	0,80
2	1,72	2,56	0,73	0,54	0,33	0,81
3	2,34	3,59	0,86	0,74	0,58	0,86
4	2,61	3,78	0,54	0,29	-0,69	0,54
5	3,97	4,69	0,60	0,36	-1,95	0,16
6	4,75	5,86	0,47	0,22	-2,52	0,07
7	2,89	3,50	0,82	0,68	-0,36	0,70
8	1,96	2,49	0,79	0,62	0,20	0,78
9	1,72	2,11	0,85	0,73	0,52	0,87
10	1,06	1,34	0,91	0,83	0,81	0,95
11	1,73	2,63	0,82	0,67	0,56	0,83
12	1,11	1,52	0,86	0,75	0,73	0,93

[1] Gjennomsnittlig absoluttavvik
[2] Roten av gjennomsnittlig kvadratavvik
[3] Korrelasjonskoeffisienten,
[4] Bestemmelseskoeffisienten
[5] Effektivitetskoeffisienten
[6] Enighetskoeffisienten

Tabell 7.2: Statistiske beregninger for bruk av modell (2.31) som benytter $T_{a,m}$, $T_{a,x}$ og $T_{a,n}$ for å beregne duggpunktstemperaturen. Verdiene er funnet for hver måned i 2008.

Måned	MAE[1](K)	RMSE[2](K)	r[3]	R^2 [4]	E[5]	d[6]
1	1,57	2,05	0,69	0,47	0,39	0,79
2	1,64	2,49	0,74	0,54	0,36	0,81
3	2,42	3,70	0,86	0,74	0,56	0,84
4	2,39	3,51	0,55	0,30	-0,45	0,58
5	3,44	4,17	0,61	0,37	-1,33	0,29
6	4,02	5,13	0,48	0,23	-1,70	0,23
7	2,00	2,66	0,82	0,68	0,21	0,81
8	1,46	1,95	0,79	0,62	0,51	0,86
9	1,44	1,79	0,85	0,72	0,66	0,90
10	1,05	1,32	0,91	0,82	0,82	0,95
11	1,74	2,67	0,82	0,68	0,55	0,81
12	1,08	1,50	0,86	0,74	0,74	0,93

[1] Gjennomsnittlig absoluttavvik
[2] Roten av gjennomsnittlig kvadratavvik
[3] Korrelasjonskoeffisienten,
[4] Bestemmelseskoeffisienten
[5] Effektivitetskoeffisienten
[6] Enighetskoeffisienten



(a) Moss **(b)** Oslo

Figur 7.2: Klarværsinnstråling for to områder rundt Oslofjorden. Målinger utført av (7.2a) Kirkeparken videregående skole i Moss i perioden 1.4.1994 - 27.2.1999 og (7.2b) Fysisk institutt, Universitetet i Oslo i perioden 1.1.1995 - 28.2.1999

Dager med innstrålingsverdier som er høyere enn 70% av ekstraterrestriell innstråling mot samme flate er erstattet med verdien produsert av modellen som beregner komponentene av solinnstrålingen (HDKR-modllen seksjon 2.6.1).

Figur 7.2 viser to sett med målinger, hvor målestasjonene ligger relativt nær hverandre (Kirkeparken videregående skole i Moss ligger ca. 6 mil fra Oslo). Det vil derfor være nærliggende å tro at været på de to stedene skulle være rimelig likt i samme periode. Ved sammenlikning av de to figurene er det mulig å gjenkjenne de samme toppene for klarværsinnstråling. Den eneste forskjellen er litt variasjon i været og forskjellig verdi for klarværsinnstråling sommerstid. Dette bekrefter at forskjellen på vær og klima er lite over korte avstander. Likevel viser figuren tydelig at det ikke er målt samme verdier for innstrålingen på de to stedene. Døgnverdien for målingene utført på Fysisk Institutt følger HDKR-modellen så nøyaktig som en kan forvente av en empirisk modell laget ut ifra innstrålingsdata fra et annet sted på kloden.

Målingene fra Moss viser en mye flatere kurve og modellen gir et avvik $> 25\%$ sommerstid. Vinterstid er avvikene omtrent like store for begge steder. Om denne variasjonen kommer av usikkerhet i måleutstyret, mangler i HDKR-modellen eller skyggende objekter vites ikke. Likevel er innstrålingsmålinger fra alle områder viktige og det er tydelig at det eneste som skiller målingene for disse to områdene er en skaleringskonstant. HDKR-modellen vil i dette tilfellet for Moss underestimere klarværsindeksen sommerstid ganske kraftig. For de aller fleste målestasjoner vil HDKR-modellen overestimere klarværsinnstrålingen om vinteren. Interpole-

ringsmetoden eliminerer dette problemet og beregner klarværsindeksen relativt til målte klarværsverdier på området.

7.4 Pyranometermålinger

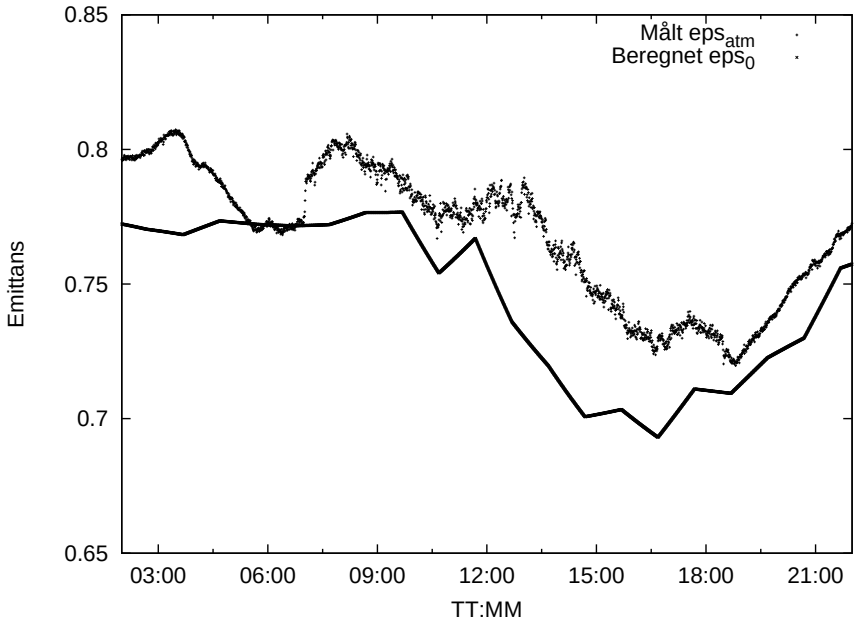
Pyranometerdata ble benyttet for å finne klarhetsindeksen ved utvikling av likn. (5.1). Da pyranometeret ble kalibrert sist i 2006 (Gjessing 2006), ble det funnet å ha en usikkerhet på ±3%. Pyranometeret hadde en hellingsvinkel $\beta = 32$ °C. Dette gir et synsfelt som er 92% av det som er tilfelle ved horisontal plassering. Siden pyranometeret har en lav respons på store innfallsvinkler vil feilen bli mindre. Kun verdier der innfallsvinkelen var mindre enn 60° ble brukt i beregningene. Klarhetsindeksen kan dermed få et avvik på opptil 4-6% på grunn av at det ikke er stilt horisontalt.

7.5 Pyrgeometermålinger

Pyrgeometermålingene hentet fra (Degnes-Ødemark 2009) er tilknyttet en del usikkerheter. For det første ble dagverdier benyttet, noe som har ført til et avvik grunnet oppvarming direkte fra solinnstråling. For det andre hadde ikke pyrgeometeret fullt synsefelt mot atmosfæren. Fysikkbygningen blokkerte en del av horisonten.

Maksimal innstråling inntreffer på dag nr 172, som er midt i måleperioden. Denne dagen kommer $\cos\theta_z$ opp i 0.8. Hvis det antas en transmisjon i atmosfæren på $\tau_c = 0{,}7$ vil pyrgeometeret kunne bli utsatt for en solinnstråling på $G_T = G_{sk}\tau_c \cos\theta \approx 750$ Wm^{-2}. Dette tilsvarer et avvik på omkring 19 Wm^{-2} når det antas en lineær sammenheng mellom solinnstråling og oppvarming av glasset. I perioden har pyrgeometeret dermed vært utsatt for en innstråling på maksimalt 750 Wm^{-2}. Ved en utetemperatur på 20 °C vil den atmosfæriske innstrålingen typisk være $R_{atm} = 300$ Wm^{-2} dersom det er klart vær og lav duggpunktstemperatur, slik at $\epsilon_{atm} \approx 0{,}7$. Avviket i ϵ_{atm} vil i dette tilfellet, med en solinnstråling på $G = 750$ Wm^{-2} være maksimalt $\delta\epsilon_{atm} = 19/300 = 0{,}063$. dette er det korrigert for ved innlesing av data.

I figur 7.3 er tidustviklingen for målt ϵ_{atm} plottet sammen med ϵ_0 beregnet ved likn (2.10) for 23. juni 2009. At kurven for ϵ_0 ikke er jevn skyldes at det er be-



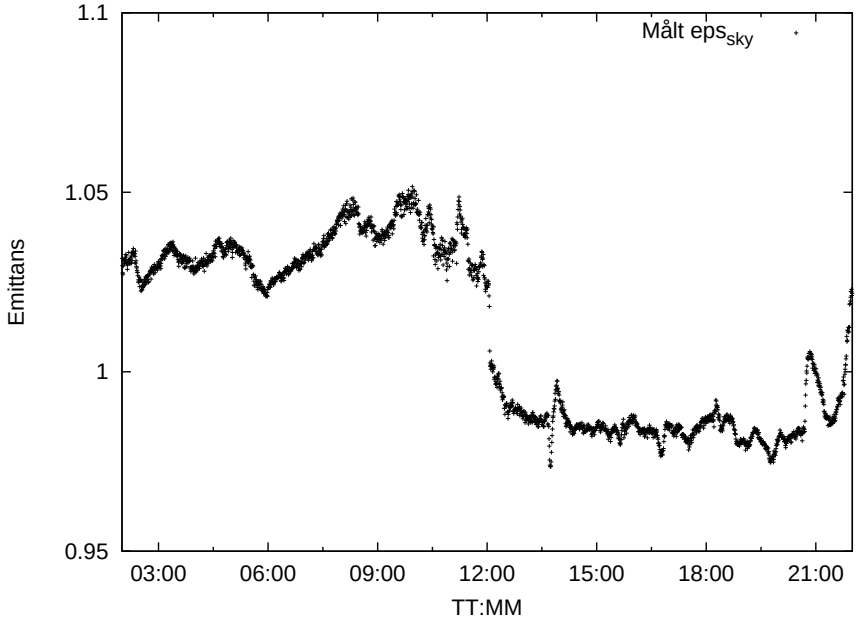
Figur 7.3: Målt og beregnet atmosfærisk emittans ved klart vær 23. juni 2009

nyttet timeverdier for utetemperatur og duggpunktstemperatur. disse er hentet fra Meteorologisk Institutt som kun registrer timeverdier. Det er derfor gjort en lineærinterpolasjon for hver timeverdi, og resultatet blir en kurve med diskontinuerlige deriverte. Det er tydelig at det blir en høyere spredning i datapunktene idet sola treffer dekkglasset til pyrgeometeret ca 07:30. Det er i dette tilfellet ikke korrigert for avviket som skyldes tempertaurøkning i glasset. Ut fra figuren ser det ut il å være et avvik på omtrent $\delta\epsilon_{atm} = +\ 0{,}03$.

Til sammenlikning er det i figur 7.4 plottet tidsutviklingen for ϵ_{atm} for en overskyet dag, 6. juli 2009. Her er det ingen tydelig skille ved soloppgang. Avviket som følge av oppvarming er her trolig rundt 2 - 3 W og er neglisjerbart.

Fysikkbygningen skygger for omtrent 15% av synsfeltet til pyrgeometeret. Dette medfører enda et positivt avvik i ϵ_{atm} siden fysikkbygningen ved klarvær vil ha en mye høyere temperatur enn atmosfæren. 23. juni 2009 var det en forskjell mellom utetemperatur og himmeltempertur på $\Delta T_{atm} = 30$ K. Denne temperaturforskjellen gir et avvik i T_{atm} på $\delta T_{atm} = 3\%$ når det antas at fysikkbygningen har en temperatur på 25 °C (Degnes-Ødemark 2009). Er atmofæretemperaturen kjent kan emittansen uttrykkes ved $\epsilon_{atm} = \frac{T_{atm}^4}{T_a^4}$. Det gir en relativ feil i atmosfærisk emittans på

$$\frac{\delta\epsilon_{atm}}{\epsilon_{atm}} = 4\frac{\delta T_{atm}}{T_{atm}} = 12\%$$



Figur 7.4: Målt emittans ved helt overskyet vær 6. juli 2009.

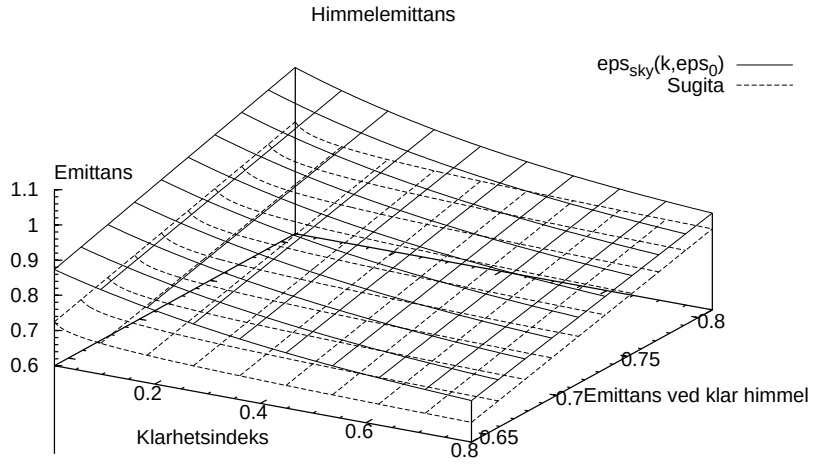
når usikkerheten i utetemperaturen neglisjeres.

7.6 Validering av emittansmodell

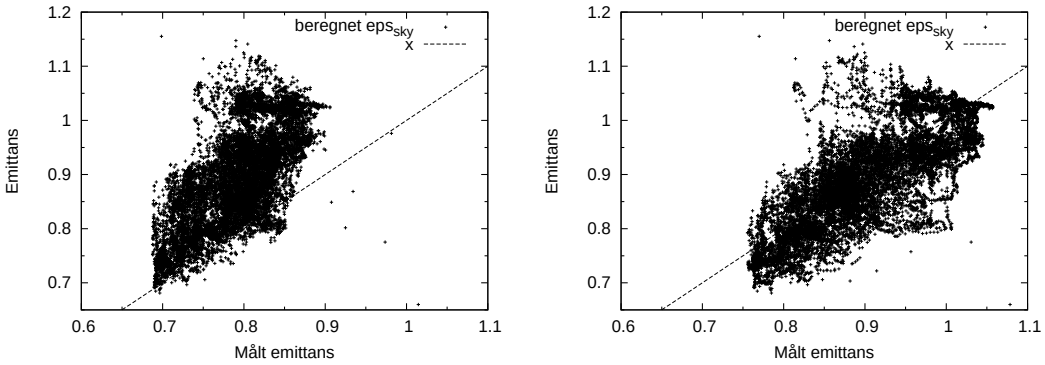
Sugita and Brutsaert (1993) har funnet en modell (likn. (2.13)) for R_{atm} som funksjon av momentan k som har vist seg å stemme med en estimert standardfeil på 15 Wm^{-2}. Ved tilpasningen av denne modellen var det benyttet 325 datapunkter der irradiansen var høyere enn 30 Wm^{-2}.

Likn 2.13 er i figur 7.5 plottet sammen med likn. (5.1). Figuren viser et systematisk avvik i ϵ_{atm} på ca. -0,06 i forhold til likn. (5.1) for høye k. Figuren viser også tydelig en større avhenginghet av ϵ_0 ved lav k (eller overskyet vær) for modellen til Sugita and Brutsaert (1993), mens likn. (5.1) har en konstant verdi på $1,05$ ved $k = 0$. At ϵ_{atm} har en relativt konstant verdi ved lav k er også tydelig ut fra figur 5.13 til 5.14. Modellen funnet i denne oppgaven viser en klart større avhengighet av klarhetsindeksen. Dette kan komme av at denne modellen benytter gjennomsnittlig klarhetsindeks over en time.

Det virker som modellen funnet i denne oppgaven vil overestimere atmosfærisk emittans med ca 0,06 i forhold til likn. 2.13. Som vist i figur 5.16 faller ikke ϵ_{sky}



Figur 7.5: To modeller for atmosfærisk emittans, ϵ_{atm} er funnet her sammenlignet med modellen funnet av (Sugita and Brutsaert 1993)



(a) (Sugita and Brutsaert 1993) **(b)** Likn (5.1)

Figur 7.6: Modellert mot målt ϵ_{atm}. I figur (7.6a) er det benyttet momentane klarhetsindekser, og i figur (7.6b) er det benyttet timevise klarhetsindekser

Tabell 7.3: Statistiske beregninger for likn. (5.1)

Type	Verdi
MAE	0,0435
$RMSE$	0,0581
R^2	0,773
E	0,597
d	0,593

ned mot ϵ_0 ved høy gjennomsnittlig k_{av}, men har et avvik på mellom 0,06 og 0,08. I teorien bør $\epsilon_{sky}(k_{av} = 0{,}7, \epsilon_0) = \epsilon_0$

Det er gjort statistiske beregninger for atmosfærisk emittans gitt ved likn. (5.1), og verdiene er vist i tabell 7.6.

Kapittel 8

Diskusjon

I denne oppgaven har jeg videre bearbeidet simuleringsprogrammet fra (Haugen 2000). Det er foretatt en omorganisering og struktorering for å gjøre programkoden mer oversiktlig og dermed enklere å utvikle. Den første oppgaven var dermed å omorganisere klassene på en naturlig måte. Mye tid ble lagt ned til å forstå virkemåten og sammensetningene og rette på feil som har sneket seg inn i kildekoden. Håpet er at den nye strukturen vil gjøre programmet enklere å utvikle i fremtiden.

8.1 Duggpunktstemperatur

Duggpunktstemperaturen er satt konstant gjennom døgnet, men det er en grov tilnærming. Resultatene viser at duggpunktstemperaturen varierer gjennom døgnet, dog med en mindre amplitude enn utetemperaturen. Døgnvariasjonen holder seg stort sett under 6 K.

Det er ikke funnet noen sammenheng for hvordan duggpunktstemperaturen utvikler seg. Variasjonen kan skyldes flere andre faktorer enn det er tatt hensyn til her. Den kan for eksempel avhenge av nedbør, vind og trykk, men en simulering av disse verdiene vil bli for omfattende og komplisert, og ville ført til at programmet simulerer tregere enn nødvendig.

Treffsikkerhetsberegningene viser at modellen som er valgt for simulering av duggpunktstemperatur gir bedre verdier for vinteren og for synkende dagtemperatur. Resultatene viser at det er en liten forbedring fra å kun benytte minimumstemperatur. Modellen gir størst avvik for tørre og varme dager, når også amplituden i

utetemperatur er stor.

En mulig forklaring på at modellen svikter for økende døgnlig utetemperatur kan være at det finnes en forsinkelse i evapotransprirasjonen (summen av evaporasjon fra objekter og transpirasjon fra levende planter), slik at det tar flere døgn før lufta blir mettet med vanndamp. En slik forsinkelse kan også forklare at modellen stemmer godt for avtakende døgnlig utetemperatur; Innholdet av vanndamp vil ikke minke før utetemperaturen synker til duggpunktstemperaturen og tvinger frem duggformasjon. Tidligere forsøk utført i Europa av Andersson-Sköld, Simpson, and Ødegaard (2008) viser at den enkle relasjonen (2.29) er en god modell for døgnlig duggpunktstemperatur. Forsøket ble utført for 32 værstasjoner over hele Europa, og modellen har vist seg å være en god tilnærming de fleste steder. Det månedlige avviket mellom T_d kl 12:00 og T_n lå stort sett under 2 K. For målestasjonen i Narvik ga modellen store feil. Avviket lå her på mellom -2 K og -8 K.

Modellen ser ut til å fungere bra de fleste steder i Europa, men kan være ustabil for steder som ligger nær kysten.

8.2 Skyhøyder

Skyhøyder blir simulert fra en diskret fordeling direkte fra skyhøydedata innsamlet fra Meteorologigsk Institutt (Meteorologisk Institutt 2009a). Dette datasettet innholder ingen verdier for skyhøyden større enn 2500 m, og forekommer i intervall på 500 m for de største verdiene og 100 m for de laveste. Likevel har skyhøydedata fra Meteorologisk Institutt vist seg å gi gode verdier for hemisfærisk emittans (Meir, Rekstad, and Løvvik 2002) og (Degnes-Ødemark 2009) ved forsøk i Oslo.

Det er valgt en ganske enkel modell for utviklingen av skyhøyder. Det er ikke sett på årlige variasjoner i skyhøyden, kun sannsynligheten for neste skyhøyde ved en gitt skyhøyde. Årlige variasjoner kan ha en betydning, da det er den varmeste delen av året kjølebehovet er høyest.

Det er ukjent hvorvidt skyhøydemodellen for Oslo vil gjelde for andre steder i landet, og ikke minst verden. Den lokale topografien vil i stor grad avgjøre i hvilket lag i atmosfæren skyene vil befinne seg. Området rundt Oslo er preget av små topper på omkring 500 m, og befinner seg innerst i en fjord. Oslo befinner seg i et

vestavindsbelte slik at den vanligste vindretningen er fra sørøst, og skyene driver inn Oslofjorden.

Videre forskning anbefales for å utvikle mer generelle modeller for skyhøyden.

8.3 Periodelengder

Innstrålingsmålingene i SOLIS-prosjektet er foretatt med hellingsvinkel på 45°. Dette er ikke ideelt da klarværsindekser er definert mot en horisontal flate, men samme hellingsvinkel er brukt i datasettet i tidligere oppgave. Det er heller ikke krav til stor nøyaktighet i klarværsindeksen, siden kun varigheten av værperiodene blir regnet ut. Klarværsindeksen er kun brukt for skille mellom gode og dårlige værperioder.

Klarværsinnstrålingen er forsøkt funnet vha. HDKR-modellen. Resultatene for klarværsinnstrålingen fra de forskjellige målestasjonene viser at det ikke er en god metode, siden den overestimerer innstrålingen om sommeren og underestimerer den om vinteren. Det kan være en svakhet ved modellen, men det kan også skyldes at omgivelsene ved målestasjonene ikke er kjent. Det kan være bygninger, fjell eller andre objekter som skygger for pyranometeret bestemte tider i døgnet. De forskjellige målestasjonene vil ha ulike omgivelser slik at resultatet i hvert enkelt tilfelle blir forskjellig. Albedoen er satt til konstant $a = 0.15$, men det kan forekomme store årlige variasjoner, avhengig av snøfall, blader på trær etc. En mulig forklaring på at innstrålingen blir underestimert vinterstid er at snødekket kan reflektere en stor del av lyset, som igjen vil treffe pyranometere på grunn av den store hellingsvinkelen. Befinner målestasjonen seg i et område med mye gatebelysning vil en del lys bli reflektert mot bakken ved overskyet vær, noe som kan gi et ekstra bidrag og medvirke til høyere innstrålingsmålinger om vinteren. Jeg har ikke noe mål på dette bidraget, men det vil antakelig være lite ($< 10\%$).

Interpolering mellom toppunkter ser ut til å være en god måte å bestemme klarværsinnstråling på. Det er benyttet lineær interpolering og metoden virker utmerket. I tidligere oppgave er klarværsinnstrålingen funnet ved å se på klare dager gjennom ett år. Innstrålingen ble da sett på uke for uke og det ble satt konstante ukeverdier. Argumentet for å gjøre det slik var at klarhetsindeksen ellers ville bli liggende høyere for dager med lav potensiell innstråling. Dette er løst i denne oppgaven ved å gå ut ifra den dagen som i teorien skulle ha lavest innstråling. Klarværsinnstrålingen for denne dagen ble funnet ved å gå fem dager frem og tilbake i

tid og finne maksimalverdien for de ti dagene.

Det ideelle hadde vært komplette datasett over flere år, uten dager og perioder med manglende målinger. Det finnes innimellom mange og/eller lange perioder uten målinger. Som vist i resultatkapittelet mangler det i snitt data for 5% av periodene. Et poeng er at det er de lange periodelengdene som er skadelidende av hull i datasett. Lange periodelengder strekker seg over lengre tidsrom enn korte, og periodlengden blir dermed lettere avbrutt av manglende måledata.

Kvaliteten på datasettene fra skolene er varierende. Enkelte datasett inneholder målinger med avbrudd på et år, og det er av den grunn valgt å slå sammen datasettende som er produsert av målestasjoner innenfor bestemte geografiske områder. Tanken er at klimaet vil være omtrent likt i hvert område. Noen datasett inneholder urealistiske verdier med tanke på årstiden. Det gjelder ikke for mer enn et titalls punkter og vil ikke få konsekvenser for resultatet.

I tidligere oppgave er det funnet at de dårlige værperiodene er weibullfordelt, mens for de gode værperiodene er den generaliserte paretofordelingen brukt. Resultatene presentert nå forteller noe annet. Det er ikke noe fysisk fenomen som tilsier at fordelingene skal være på den ene eller andre formen. Det virker som det er tilfeldigheter som fører til hvilken modell som passer best. Spesielt bør det merkes at for Østlandet total sett, inkludert Oslo, er det de gode periodelengdene som er weibullfordelt, mens paretomodellen passer bedre for de dårlige periodelendene.

Tidsutviklingen av klarværsindekser har en sammenheng med størrelsen på lavtrykk- og høytrykksentere som driver innover land. Denne informasjonen kommer tydeligere frem ved kategorisering etter lengden på foregående periode. Ved å inndele Norge i klimasoner, vil det være mulig å bestemme potensialet for solenergi og nattekjøling mer presist.

8.4 Valg av modell for hemisfærisk emittans

Det ble forsøkt å finne en modell for hemisfærisk emittans som funksjon av klarværsindeks direkte, og modellen som er funnet er gitt ved likn. (5.1). Som et første forsøk ble klarhetsindeks benyttet siden den er enklere å beregne enn klarværsindeks. Datagrunnlaget ble hentet fra tre forskjellig kilder, og siden temperaturdata er hentet fra (Meteorologisk Institutt 2009a) og ikke fra lokale målinger er det usikkert hvor relevante verdiene er.

De statistiske beregningene som er utført og gitt i tabell 7.6, viser at modellen følger de observerte verdiene. Som vist i figur 5.16 er det likevel stor variasjon i datasettet. En $RMSE$-verdi på 0,0581 er for høy for en modell som skal benyttes for å beregne kjølepotensial.

Modell 2.13 som er funnet i (Sugita and Brutsaert 1993) er en potensfunksjon som minker proporsjonalt med $k^{-0,0227}$. Den lave eksponenten i modellen fører til at emittansen får en bratt øknnig ved lave klarhetsindekser ($k < 0,2$). Den avhenger for lite av klarhetsindeksen til å kunne beregne atmosfærisk emittans tilfredsstillende, noe som figur 7.6a viser. Modellen underestimer verdier for datasettet benyttet i denne oppgaven. I og med at kun 325 datapunkter ble benyttet til å utvikle modellen er det tvilsomt hvor nyttig den er.

Likn. (2.12) ble valgt for å beregne atmosfærisk emittans, da den har vist seg å beskrive kjølepotensialet godt for Oslo tidligere (Meir, Rekstad, and Løvvik 2002). Den er også forholdsvis enkel, og krevde kun modeller for skyhøyde og duggpunktstemperatur, siden klarværsindeksen ble benyttet som en modell for skydekket.

8.5 Simuleringsresultater

Simuleringen av atmofærisk emittans ϵ_{atm} viser viser modellen som er implementert virker utmerket. Den årlige variasjonen av klarværsemittansen ϵ_0 kommer tydelig frem. Det faktum at $\epsilon_{atm} \to 1$ uavhengig av verdien til ϵ_0. En radiator med emittans $\epsilon_{rad} \geq 0,9$ vil ha et betydelig utstrålingspotensiale gjennom hele året, også på sommeren.

Simuleringen av behov for hver sone viser at det ikke er store klimatiske avvik når temperaturparameterne er like. Siden modellene for behov kun er temperaturbasert, kan det enten bety at modellen for utetemperatur er for lite avhengig av klarværsindeksen eller at den klimatiske forskjellen ikke er så stor som først antatt.

8.6 Forslag til forbedringer

Målet med oppgaven var å videreutvikle et simuleringsprogram for oppvarming basert på en enkelt solfangersystem til også kunne beregne kjøling ved bruk av flatplateradiatorer, samt videreutvikle noen av modellene som styrer værgenere-

ringen i programmet. Implementering av modeller for behov har vært vellykket. Fullstendig implementering av kjølepotensial har ikke latt seg gjøre før tidsfristen. Under utviklingen av modellene er det kun benyttet data fra Norge. Likevel kan metodene som er benyttet la seg enkelt bruke til å utvikle modeller som gjelder andre steder. Noen svakheter som fremdeles eksisterer er diskutert under.

8.6.1 Vindhastighet og konveksjon

I oppgaven er det utviklet metoder og modeller for beregning av kjølepotensial ved utstråling til nattehimmelen. Det er dermed kun strålingstapet det er tatt hensyn til når kjølepotensialet beregnes. Varmetap bestemmes ved tre forskjellig prosesser. Stråling er en av prosessene og de andre to andre er konveksjon og varmeledning. Dersom det brukes vindskjerm og tynn polymerfolie, vil konveksjon bli undertrykt og størsteparten av varmetapet vil foregå ved utstråling.

Konveksjonen er undertrykt for absorbatorer med dekkglass. For simulering av oppvarmingspotensiale har vindhastigheten trolig en neglisjerbar innvirking. Gamle SolDat fungerer fremdeles utmerket til dette formålet.

Dersom det ikke benyttes dekkglass er radiatoreffekten i stor grad avhengig av konveksjon. Den gjennomsnittlige vindhastigheten på jorda er målt til 5 ms^{-1} Det hadde dermed vært interessant å finne en modell for vindhastighet og implementere denne i programmet.

8.6.2 Modeller for klarværsindeks

Modellene for klarværsindekser er de samme som er benyttet i tidligere oppgave (Haugen 2000). Det er benyttet tre modeller for: døgnvis, timevis og momentan klarværsindeks. Ved studering av grafene som produseres for tidsserien av klarværsindekser ser ikke modellene ut til å gi tilfredsstillende resultater. Den momentane modellen ser ut til å gi gode verdier for korte tidsrom ved små tidssteg på 10 min eller mindre, mens det er klare tendenser til at været skifter brått omtrent en gang i timen. Den timevise klarværsindeksen ser med andre ord ut til å mangle autokorrelasjon.

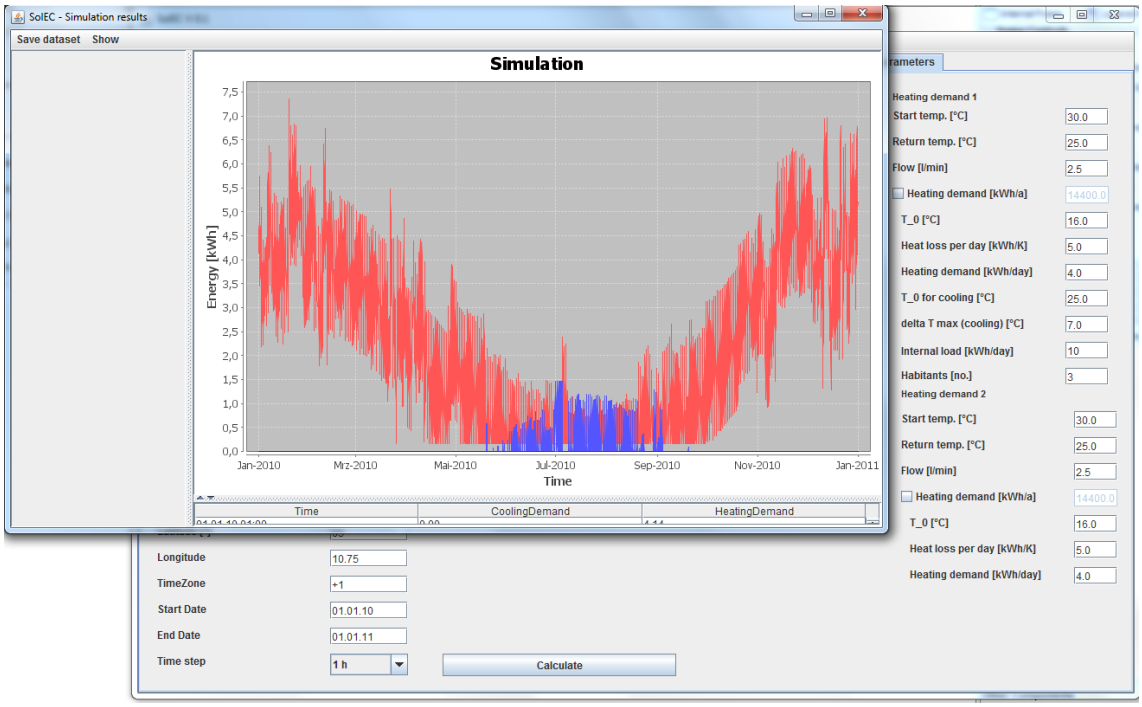
Kapittel 9

Konklusjon

Programmet som er blitt videreutviklet i denne oppgaven har den fordel i forhold til andre modeller at det simulerer været og varigheten til værperiodene, uten å kreve detaljerte, historiske værdata. Dette blir gjort ved Monte Carlo-metoden, der været blir generert ved å plukke ut en rekke tilfeldige tall som benyttes for å plukke ut periodelengder, klarværsindekser, skydekke og skyhøyder ut fra sine respektive fordelinger.

I SolEC er modeller for simulering av varmebehov og kjølebehov implementert. Et skjermbilde av programmet, der varme- og kjølebehov er simulert ut fra standardparameterne er vist i figur 9.1.

Værmodellene utvilket i denne oppgaven er implementert. Modellene gir variable som kan benyttes til simulering av utstrålingspotensial: skyhøyde, andel skydekke og duggpunktstemperatur. Modellen som beregner klarværsindeks er forbedret ved å inkludere innstrålingsdata utført ved videregående skoler over hele landet. En modell for hemisfærisk emittans direkte når klarværsindeksen og duggpunktstemperturen er kjent er utviklet. Denne modellen har vist seg å overestimere emittansen med omtrent 0,05 i forhold til andre modeller. Den er derfor ikke ansett for å være pålitelig og det er valgt å implementere en godt etablert modell. Selv om modellene som gir værdata er implentert, er det ikke tukket å implentere kjølepotensialet fullstendig.



Figur 9.1: SolEC, slik programmet fremstår i dag. Her beregnes timevis kjølebehov og oppvarmingsbehov med standard behovs- og værparametere og for Østlandet.

Symbolliste

a_k Kjøleandel, side 29

A_{rad} Det totale arealet til en radiator (m^2)., side 12

a_v Solenergiandel, side 29

C_i bidraget til skydekke i., side 13

c_w Spesifikk varmekapasitet til vann, side 27

d Diffusandel., side 17

$\frac{dQ}{dT}$ Boligens varmetap (kWh(K døgn)$^{-1}$), side 26

ΔQ_{ak} Kjølebidrag, side 30

ΔQ_{as} Solenergibidrag, side 30

ΔQ_{RK} Kjølebehov for et tidssteg, side 27

ΔQ_{RV} Romvarmebehov for et tidssteg, side 27

$\Delta Q_{eks,k}$ Ekstra energi krevet til kjøling, side 29

$\Delta Q_{eks,v}$ Ekstra energi krevet til oppvarming, side 29

ΔQ_{VV} Varmtvannsbehov, side 27

Δt Simuleringens tidssteg (min), side 23

E Tidslikningen (min)., side 15

e_0 Vanndamptrykk ved bakkenivå (hPa)., side 12

ϵ_0 Atmosfærisk klarværsemittans., side 12

ϵ_{atm} Atmosfærisk emittans., side 11

ϵ_{rad} Emittans til en radiator., side 12

F_g Synsfeltet mot jordoverflaten., side 19

F_s Synsfeltet mot himmelen., side 18

$F_{s,c}$ Sysnfelt mot klar himmel, side 24

G Global innstråling, mot horisontal flate (Wm^{-2})., side 9

G_c Global solinnstråling ved klarvær (Wm^{-2})., side 10

G_o Global ekstraterrestriell innstråling mot horisontal flate (Wm^{-2})., side 9

G_{on} Global ekstraterrestriell innstråling normalt på innstrålingsretningen (Wm^{-2})., side 9

G_{sk} Solarkonstanten $G_{sk} = 1367$ Wm^{-2}., side 16

G_T Global innstråling mot skråflate, side 16

$G_{b,T}$ Den direkte komponenten av global innstråling mot skråflate, side 16

$G_{d,T}$ Den diffuse komponenten av global innstråling mot skråflate, side 16

$G_{r,T}$ Den reflekterte komponenten av global innstråling mot skråflate, side 16

H Total innstråling gjennom et døgn (Whm^{-2})., side 37

H_c Total klarværsinnstråling gjennom et døgn (Wh m^{-2})., side 39

k klarhetsindeks., side 9

k_{av} Gjennomsnittlig klarhetsindeks, side 61

k_c Klarværsindeks., side 10

$l_{P,ann}$ Total årlig nedbør (mm)., side 20

$l_{Ep,dag}$ Potensiell døgnlig evapotranspirasjon (mm)., side 20

L_{lok} Lokalmeridianden, side 15

L_p Stokastisk periodelengde (dager), side 32

L_{std} Standardmeridianden for et geografisk område., side 15

N Antall målepunkter., side 20

n Andel skydekke på hemisfæren, side 13

n Skydekket, side 24

N Dagen i året., side 15

n_i Andel skydekke av skytype i., side 13

p Lokalt atmosfærisk trykk (hPa)., side 12

P_{rad} Det totale varmetapet ved terminsk utstråling for en radiator (W)., side 12

R_{atm} Atmosfærisk innstråling (Wm^{-2})., side 11

σ Stefan-Boltzmanns konstant $\sigma = 5{,}670400 \times 10^{-8}\ \mathrm{Wm}^{-2}\mathrm{K}^{-4}$, side 8

E Effektivitetskoeffisienten., side 21

d Enighetskoeffisienten., side 21

MAE Gjennomsnittlig absoluttavvik., side 20

O Observert verdi., side 20

P Beregnet verdi., side 20

R^2 Bestemmelseskoeffisienten., side 21

r Korrelasjonskoeffisienten., side 21

$RMSE$ Roten av gjennomsnittlig kvadratavvik., side 20

T_a Utetemperatur(K eller °C)., side 11

t_s Antall timer fra midnatt i soltid., side 12

τ_c Atmosfærisk transmittans ved klarvær., side 10

$T_{a,m}$ Døgnets middeltemperatur (°C)., side 20

$T_{a,n}$ Døgnlig minimumstemperatur (°C)., side 19

T_{atm} Himmeltemperatur (K)., side 11

$T_{a,x}$ Døgnets maksimumstemperatur (°C)., side 20

T_b Grensetemperatur for varmebehov, side 27

$T_{b,k}$ Balansetemperatur for kjølebehov., side 27

T_d Duggpunktstemperatur (°C)., side 12

ΔT_k Temperaturforandring i kuldelageret etter et tidssteg, side 29

T_{kv} Kaldtvannstemperatur (°C), side 27

t_{std} Lokaltid i timer etter midnatt., side 15

$T_{t,k}$ Termostattemperaturen til kuldelageret, side 29

$T_{t,v}$ Termostattemperaturen til varmelageret, side 29

ΔT_v Temperaturforandring i varmelageret etter et tidssteg, side 29

T_{VV} Ønskelige varmtvannstemperatur (°C), side 27

ϕ Breddegrad, nord positiv og sør negativ. $\phi \in \{-90°, 90°\}$, side 14

B En vinkel som representerer dagen i året, side 15

L Lengdegrad, vest positiv. $L \in \{0°, 360°\}$, side 14

β Planets hellingsvinkel (°), side 14

γ Planets asimutvinkel (°), side 14

δ Deklinasjon (°)., side 14

ω Timevinkelen (°), side 16

θ Innfallsvinkel mellom solinnstråling og en flate (°), side 16

θ_z Senitvinkelen(°), side 16

α_s Solhøyden (°), side 16

V_k Volumet til kuldelageret, side 29

V_v Volumet til varmelageret, side 29

V_{VV} Døgnlig forbruk av varmtvann ($l\text{dag}^{-1}$), side 27

z_h Høyden til skydekkets basisnivå (km)., side 13

Referanser

ACE (2007, August). http://www.aceee.org/consumerguide/cooling.htm.

Agency, E. P. (2007). Atmosphere changes. http://www.epa.gov/climatechange/science/recentac.html.

Andersson-Sköld, Y., D. Simpson, and V. Ødegaard (2008). Humidity parameters from temperature: test of a simple methodology for european conditions. *International Journal of Climatology 28*(7), 961–972.

Argiriou, A., M. Santamouris, C. Balaras, and S. Jeter (1993). Potential of radiative cooling in southern europe. *International journal of solar energy 13*, 189 – 203.

Berdahl, P. and R. Fromberg (1982). The thermal radiance of clear skies. *Solar Energy 29*(4), 299 – 314.

Berdahl, P. and M. Martin (1984). Emissivity of clear skies. *Solar Energy 32*(5), 663 – 664.

Butler, D. (1992). Daily patterns of dew-point temperature in a semiarid climate. *Agricultural and Forest Meteorology 60*(3-4), 267 – 278.

CICERO (2009). Hvorfor endrer klimaet seg? http://www.cicero.uio.no/abc/klimaendringer.aspx.

Cook, J. (1989). *Passive Cooling*, Chapter 4. MIT Press.

Crawley, D. B., L. K. Lawrie, C. O. Pedersen, and F. C. Winkelmann (2000). EnergyPlus: energy simulation program. *ASHRAE 42*(4), 49 – 56.

D. N. Prabhakar Murthy, M. X. and R. Jiang (2004). *Weibull Models*. John Wiley & Sons.

Degnes-Ødemark, H. (2009). A study of night sky radiation, and heating and cooling of buildings with thermal solar collectors. Master's thesis, Universitet i Oslo.

Duffie, J. A. and W. A. Beckman (2006a). *Solar Engineering of Thermal Processes*. John Wiley & Sons.

Duffie, J. A. and W. A. Beckman (2006b). *Solar Engineering of Thermal Processes*, Chapter 2.16. In Duffie and Beckman (2006a).

Dyer, J. and D. Brown (1977). A climatic simulator for field-drying hay. *Agricultural Meteorology 18*(1), 37 – 48.

Gjessing, J. (2006). Ventilering som metode for å redusere stagnasjonstemperatur i solfangere. Master's thesis, Universitetet i Oslo.

Haugen, G. M. (2000). Måling og simulering av værperioders varighet. Master's thesis, Universitetet i Oslo.

Hetland, K. T. and E. Oterholm (2001). Solis-prosjektet. `http://fysikk.hfk.vgs.no/solar.htm`.

Hubbard, K. G., R. Mahmood, and C. Carlson (2003). Estimating daily dew point temperature for the northern great plains using maximum and minimum temperature. *Agron J 95*(2), 323–328.

Ingebretsen, F. (1991). Solnor, et program for beregning av solbidrag til varmtvann og romvarme. Technical report, Fysisk Institutt.

Khan, M., A. Khalique, and A. Abouammoh (1989, August). On estimating parameters in a discrete weibull distribution. *Reliability, IEEE Transactions on 38*(3), 348–350.

Kimball, J. S., S. W. Running, and R. Nemani (1997). An improved method for estimating surface humidity from daily minimum temperature. *Agricultural and Forest Meteorology 85*(1-2), 87 – 98.

Kipp & Zonen (2009). *INSTRUCTION MANUAL, CG1 Pyrgeometer, CG2 Net Pyrgeometer*. Kipp & Zonen B.V. Rontgenweg 1, 2624 BD.

Legates, D. R. and J. McCabe, Gregory J. (1999). Evaluating the use of goodness-of-fit measures in hydrologic and hydroclimatic model validation. *Water Resour. Res. 35*.

Martin, M. and P. Berdahl (1984). Characteristics of infrared sky radiation in the united states. *Solar Energy 33*(3-4), 321 – 336.

Meir, M. G., J. B. Rekstad, and O. M. Løvvik (2002). A study of a polymer-based radiative cooling system. *Solar Energy 73*(6), 403 – 417.

Meteorologisk Institutt (2007). Skyer. http://met.no/Meteorologi/A_male_varet/Observasjoner_fra_land/Dette_blir_malt/Visuelle_observasjoner/Skyer/.

Meteorologisk Institutt (2009a). eKlima tilgang til meteorologisk institutts vær- og klimadata. http://www.eklima.no/.

Meteorologisk Institutt (2009b). Kvaliteten på observasjonene. http://met.no/Meteorologi/A_male_varet/Observasjoner_fra_land/Dette_blir_malt/Kvaliteten_pa_observasjonene/.

Montgomery, D. C. and G. C. Runger (2003). *Applied Statistics and Probability for Engineers* (3 ed.)., Chapter 6.5, pp. 207–208. John Wiley & Sons.

Naturfagsenteret.

Olseth, J. A. and A. Skartveit (1994). Review and test of parameteterizations of atmospheric radiation. *IEA-SHCP-17F-2*.

Rekstad, J. and M. Meir (2009, November). ENERGY AND PHYSICS compendium for fys4540 energifysikk, solenergi. Department of Physics, University of Oslo, Norway.

Rhode, R. A. (2007). Radiation transmitted by the atmosphere. http://globalwarmingart.com/wiki/File:Atmospheric_Transmission_png.

Storås, H. (1997). Om kjøling av bygninger ved utnyttelse av infrarød stråling mot nattehimmelen. Master's thesis, Universitetet i Oslo.

Sugita, M. and W. Brutsaert (1993). Cloud effect in the estimation of instantaneous downward longwave radiation. *Water Resources Research 29*, 599–606.

Sun Microsystems (2009). Java platform, standard edition 6 api specification. http://java.sun.com/javase/6/docs/api/.

T. E. S. S. (1975). TRNSYS, the transient energy system simulation tool. http://www.trnsys.com/.

Williams, T. and C. Kelley (2007). gnuplot an interactive plotting program. http://www.gnuplot.info/docs/gnuplot.html.

Winn, C. B., G. R. Johnson, and T. E. Corder (1974, Aug). SIMSHAC - a simulation program for solar heating and cooling of buildings. *SIMULATION 23*(6), 165–170.

Øystein Godøy (2004). Tuning and validation of a downward longwave irradiance algoritm. *Remote Sensing* (2).

Tillegg A

Eksperiment: vinkelavhengighet for pyranometer

Pyranometeret som ble brukt i denne oppgaven var av typen SolData 489SPC. Hensikten har vært å hente innstrålingsdata for å beregne klarværsindekser, og siden klarværsindekser er definert mot horisontale flater ble pyranometeret opprinnelig plassert horisontalt. Ideen til disse eksperimentene kom relativt sent, i slutten av september 2009. Dermed er solinnstrålingsmålingene utført ved store senitvinkler ($> 70°$). Dette ga utslag i grafene for skyfrie dager ved at klarværsindeksene hadde en buet form i den perioden sola skinte direkte på pyranometeret. Overflaten til dette pyranometeret er svakt buet, tilnærmet plant. Det ble dermed mistenkt for å ha en større relativ respons som funksjon av innfallsvinkel enn tidligere antatt, og dermed ble det satt igang et forsøk for å finne ut om det var måleinstrumentet eller atmosfæreeffekter som var skyld i krummingen.

Utstyr

- Pyranometer: SolData 489SPC med kalibreringskonstant 172 $\mathrm{mV}\left(\mathrm{kWm}^{-2}\right)^{-1}$
- Gradeskive
- Loggeprogram i LabView

Utføringen av eksperimentet var relativt enkelt. 27.10.2009 var en skyfri og klar dag. I tiden mellom 12:00 og 14:00 var variasjonen i solhøyde/senitvinkel liten nok til at pyranometeret kunne tiltes i flere vinkler mens innstrålingen holdt seg konstant. Det ble benyttet en enkel gradeskive med oppløsning på 1°, og for å kunne

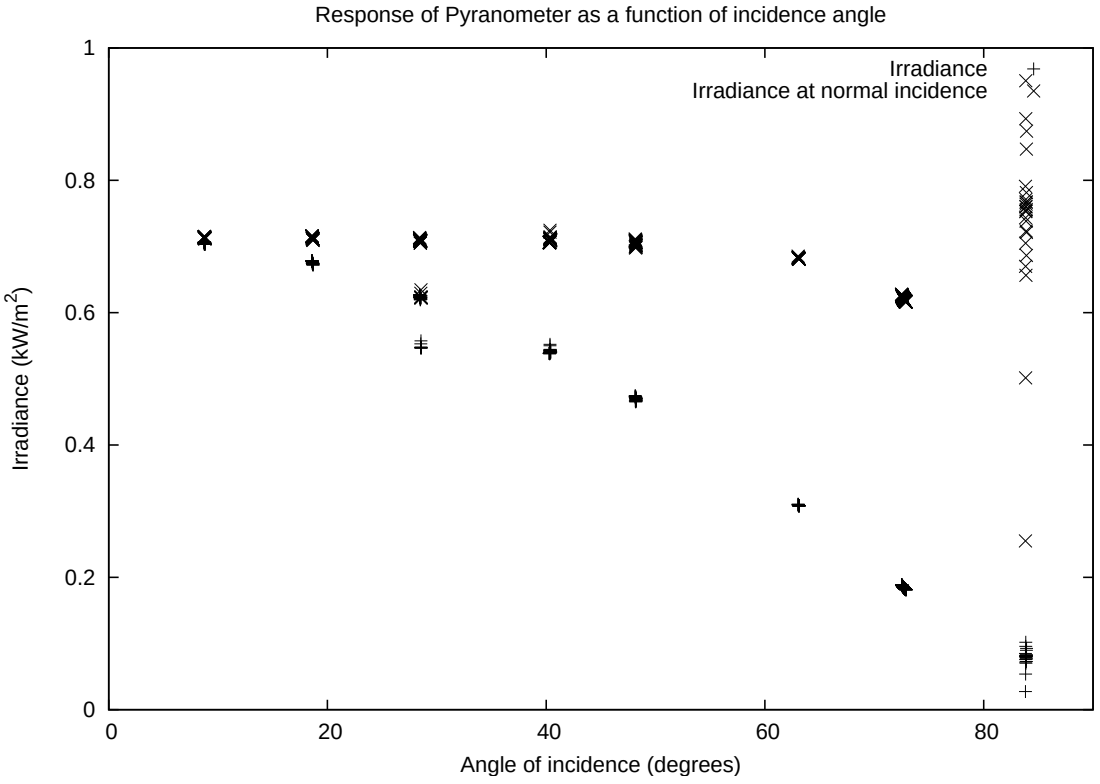
Tabell A.1: Hellingsvinkel og sluttidspunkt for målingene

Hellingsvinkel (°)	Tidspunkt
0	12:55
10	13:13
25	13:16
33	13:20
45	13:23
55	13:26
65	13:29
-10	13:31

holde pyranometeret fast i lengre tid i samme vinkel ble gradeskiva skrudd fast til en planke. Målingene ble utført i retning direkte mot Sola, og dermed kunne innfallsvinkelens avhengighet av asimutvinklen elimineres. Loggeprogrammet benyttet et tidssteg på 6 s, og for å få rimelige resultater med lite statistisk fluktuasjon ble pyranometeret holdt fast i samme hellingsvinkel i minimum 2 min. Det ble utført to serier med eksperimenter, men kun den siste ga data siden datamaskinen som logget resultatene krasjet under første måleserie.

Dersom senitvinkelen er kjent, kan innfallsvinkelen finnes når asimutvinkelen til pyranometeret sammenfaller med Solas asimutvinkel. I dette tilfellet er $\theta = \theta_z - \beta$. Senitvinkelen ble funnet ved bruk av likn. (2.20), og timevinkel og soltid ut fra likn. (2.17).

Figuren viser at pyranometeret ved direkte innstråling og ved større innfallsvinkler enn ca 60°, gir betydelig feil resultat for innstrålingen normalt på senitvinkelen (G_n). Målinger med hellingsvinkel på $-10°$ ser ikke ut til å gi mye informasjon. Antakelig blir størsteparten av solinnstrålingen reflektert ved denne innfallsvinkelen og den strålingen som detekteres vil hovedsakelig bestå av albedo- og diffuskomponenter. Spesielt siden pyranometeret ved denne hellingvinklen ser store deler av fysikkbyningen.



Figur A.1: Innstråling ved forskjellige vinkler. X betegner beregnet innstråling normalt på innstrålingsretningen.

Tillegg B

Vektfaktorer for kjøling

Det er het utviklet en geometrisk metode for beregning av atmosfærisk innstråling når en horisontprofil er kjent.

B.1 Romvinkler

På samme måte som en vinkel er definert som en del av en enhetssirkel er en romvinkel definert som en del av en enhetskule. En romvinkel måles i SI-enheten steradianer eller sr, alternativt kan den måles i kvadratgrader eller $(^\circ)^2$ som ikke er en SI-enhet. 1 sr er det samme som $(180/\pi)^2\ (^\circ)^2 \approx 3280\ (^\circ)^2$. Siden alle vinkler i oppgaven er gitt i grader er det naturlig å angi romvinkler i kvadratgrader. Romvinklen for en halvkule er gitt ved overflaten til en halvkule med radius $r = 1$:

$$\omega_0 = \frac{1}{2}\int_0^{2\pi} d\phi \int_{\frac{\pi}{2}}^{\pi} sin\theta d\theta = 2\pi \tag{B.1}$$

$$\omega_0 = \frac{1}{2}\int_{0^\circ}^{360^\circ} d\phi \int_{90^\circ}^{180^\circ} sin\theta d\theta = 64800/\pi\ (^\circ)^2 \tag{B.2}$$

Det er dermed 2π sr eller $64800/\pi\ (^\circ)^2$ i en halvkule, eller 4π sr eller $129600/\pi\ (^\circ)^2$ i en kule.

Kjøling (eller oppvarming) av objekter foregår ved stråling, varmeledning og konveksjon. Kun to av disse prinsippene er har vesentlig bidrag til kjøling av paneler og det er stråling og konveksjon. På grunn av turbulens og forflytning av luftmasser vil varmeledning i luft være neglisjerbar i forhold til forflytning av molekylene

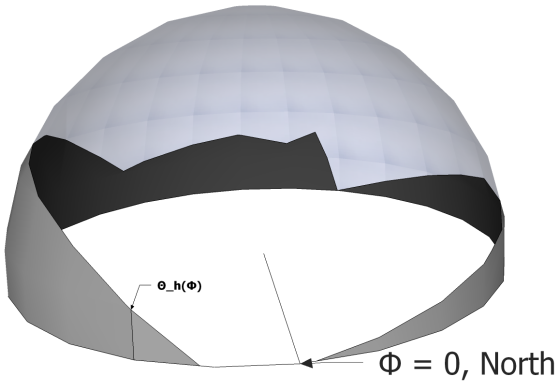
som har mottat denne energimengden. Dermed finner en uttrykket for kjøling av paneler:

$$P_k = P_{Rad} + P_{konv} \tag{B.3}$$

Her er varmetapet for kjøling dermed definert som positivt dersom systemet taper energi og negativt dersom system mottar energi.

B.2 Horisont

Et eksempel på en horisont tegnet for et horisontalt plan er vist i figur B.1. Her tilsvarer $\Theta = \frac{\pi}{2} = 90°$ horisontalplanet og $\Theta = \pi = 180°$ senit. Φ er definert lik 0 i nordlig retning. $\Theta_o(\Phi)$ er vinkelen som tilsvarer toppen til horisonten i retning Φ



Figur B.1: Eksempel på horisont, tegnet i Google Sketchup

R er innkommende infrarød stråling fra omgivelsene og himmelen og kan igjen inndeles i to eller tre komponenter. R_{sky} er innstråling fra himmelen, R_{ground} er innstråling fra bakken og $R_{horizon}$ er innstråling fra omgivelser. Panelet ser omgivelsene gjennom en romvinkel på 2π steradianer eller nøyaktig $64800/\pi\ (°)^2$. Disse komponentene er funnet ved å integrere arealet som panelet ser av respektivt himmelen, omgivelsene og bakken. Dette arealet er så normalisert mot det totale arealet for en halvkule A_o (B.2). I programmet må nødvendigvis denne integrasjonen gjøres numerisk siden data fra horisontprofilen og skyggeprofilene er tegnet inn for hånd. Det finnes dermed ingen analystisk funksjon for horisonten eller skyggeprofilene som kan integreres over romvinklene. Et uttrykk for numerisk

romvinkelintegrasjon er

$$A_0 \approx \sum_{i=0}^{N_i} \sum_{j=0}^{Nj} \Delta A_{0,ij} \tag{B.4}$$

Her er N_i antall steg i Φ fra $\Phi = 0°$ til $\Phi = 360°$ og N_j antall steg i Θ fra $\Theta = 90°$ til $\Theta = 180°$. $\Delta\Phi_0 = \frac{360°}{N_i}$ og $\Delta\Theta_0 = \frac{360°}{N_j}$ er steglengder i de to vinkelretningene. $\Delta A_{0,ij}$ er arealet en liten del av kula deles opp i. Dette arealet er vist i figur B.2. Det er en funksjon av steglengder og de momentane vinklene. Det er funnet to forskjellige metoder å summere overflatene på og begge benytter enheter i grader, siden det er mest naturlig å se for seg vinklene for horisonten i grader og ikke i radianer. Metodene er testet med $N_i = 4N_j$ der $N_j \in \{90, 2500\}$ intervall på $\Delta N_j = 2$. Det vil si steglengder $\Delta\Phi_0 = \Delta\Theta$ fra ca $0,03°$ til $1°$. Tiden beregning tok er funnet programmatisk. Feil som funksjon av tid er plottet i figur B.5. Den store spredningen i feilen ved lav prosesseringstid skyldes en begrensing i maskinvaren for hvor nøyaktig en datamskin kan beregne tiden. Det laveste tiden som ble funnet var på $0,015$ s. Her er usikkerheten oppad på omtrent $+0,08$ s og nedad på $-0,015$ s. Den absolutte relative feilen i forhold til det analytiske resultatet er plottet i figur B.4.

1. Overflaten deles opp i rektangler med sider $\Delta\Phi_j$ og $\Delta\Theta_0$, der $\Delta\Phi_i$ er gitt i lign. (B.9).

$$A_0 \approx \sum_{i=0}^{N_i} \sum_{j=0}^{Nj} \Delta\Phi_j \Delta\Theta_0 \tag{B.5}$$

2. En annen metode er å dele opp overflaten i trapeser der en vilkårlig $\Delta A_{0,ij}$ er illustrert i figur B.2. Høyden i hvert trapesoid blir kalkulert ved hjelp av Pytagoras læresetning. Kravet er at steglengden er så lav at $\Delta\Theta_0$ er tilnærmet rett.

$$h = \sqrt{\Delta\Theta^2 + \left(\frac{\Delta\Phi_j - \Delta\Phi_{j-1}}{2}\right)^2}$$

Dermed er arealet

$$\Delta A_{0,ij} = \left(\frac{\Delta\Phi_j + \Delta\Phi_{j-1}}{2}\right) \sqrt{\Delta\Theta^2 + \left(\frac{\Delta\Phi_j - \Delta\Phi_{j-1}}{2}\right)^2} \tag{B.6}$$

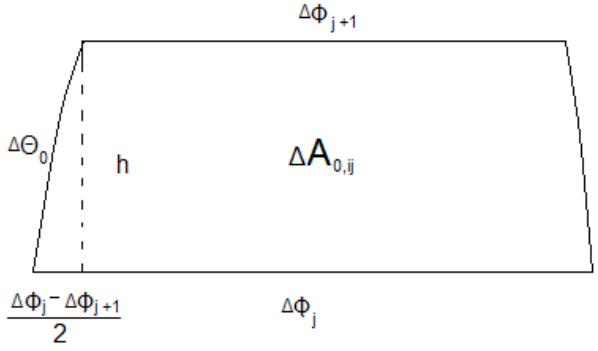
Dette gir større nøyaktighet enn summering av infinitesimale rektangler si-

den toppen $\Delta\Phi_{i+1}$ av arealet $\Delta A_{0,ij}$ vil være kortere enn grunnlinjen $\Delta\Phi_i$. Ut i fra figur B.4 er det mulig å se at feilen er $< 0,01\%$ ved steglengder på $1°$, og konvergerer raskere mot det analytiske svaret enn metode 1. Feilen er $< 5 \cdot 10^{-5}$ ved $\Delta\Phi_0 = \Delta\Theta_0 \approx 0,03$.

$$\Theta_{j+1} = \Theta_j + \Delta\Theta \tag{B.7}$$

$$\Delta\Phi_j = \Delta\Phi_0 sin(\Theta_j) \tag{B.8}$$

$$\Delta\Phi_{j+1} = \Delta\Phi_0 sin(\Theta_{j+1}) \tag{B.9}$$



Figur B.2: Den infinitesimale overflaten til en enhets-halvkula. Sidene $\Delta\Theta_0$ nærmer seg rette for små arealer. Det gjør ΔA tilnærmet et trapes og øker nøyaktigheten til numerisk romvinkelintregrasjon.

De forskjellige vektfaktorene er dermed

$$W_{f,s} = F_s \frac{A_s}{A_0} = \frac{F_s}{A_0} \sum_{i=0}^{N_i} \sum_{j=j(\Theta_0)}^{Nj} Pr(\Phi, \Theta + 90°) \Delta A_{0,ij} \tag{B.10}$$

$$W_{f,h} = F_s \frac{A_h}{A_0} = \frac{F_s}{A_0} \sum_{i=0}^{N_i} \sum_{j=0}^{j(\Theta_0)} Pr(\Phi, \Theta + 90°) \Delta A_{0,ij} \tag{B.11}$$

$$W_{f,g} = F_g \tag{B.12}$$

$W_{f,x}$ er vektfaktoren mot x tilsvarende himmelen, horisonten og bakken og $Pr(\Phi, \Theta + 90°)$ er verdier fra skyggeprofilene og horisontprofilen. Θ får et tillegg på $90°$ siden horisontalen er definert ved $\Theta = 0°$ i profilene, og tar verdiene $\{0°,\ 90°\}$. Vekt-

faktorene vil vektes med en årlig skyggeprofil $Pr(n)$ der n er dagen i året. Denne profilen er naturlig å bruke dersom det eksisterer en klar årlig variasjon i skyggen. For eksempel vil ofte løvtrær ha en naturlig forkjell i bladtetthet, spesielt ved store breddegrader nær begge polarsirker. Det kreves ingen ny integrasjon, siden denne profilen er uavhengig av vinkler. $j(\Theta_0)$ er indeksen ved $\Theta = \Theta_0(\Phi)$, som tilsvarer den nedre grensen for vektfaktoren mot himmelen.

$$\Theta_0(\Phi) = \max\left(\Theta_\beta(\Phi),\ \Theta_h(\Phi)\right) \tag{B.13}$$

$\Theta_\beta(\Phi)$ er en funksjon av hellingsvinkelen β til panelet og vil variere med Φ. Ekstremalverdiene av denne funksjonen vil være gitt av β, $\Theta_\beta(\Phi = \Phi(\gamma)) = -\beta$ og $\Theta_\beta(\Phi = \Phi(\gamma + 180°)) = \beta$. Dette tilsvarer den trigonometriske funksjonen $\Theta_\beta(\Phi) = \beta cos(\Phi - \gamma)$.Kun positive verdier for Θ_β er ønskelige, siden området under horisontalplanet tilsvarer synsfeltet mot bakken og er gitt ved lign. 2.27. Selv om panelet er montert i et bratt terreng, om det blir sett bort fra jordas krumming, vil synsfeltet likevel være mot bakken. Dette gir relasjonen

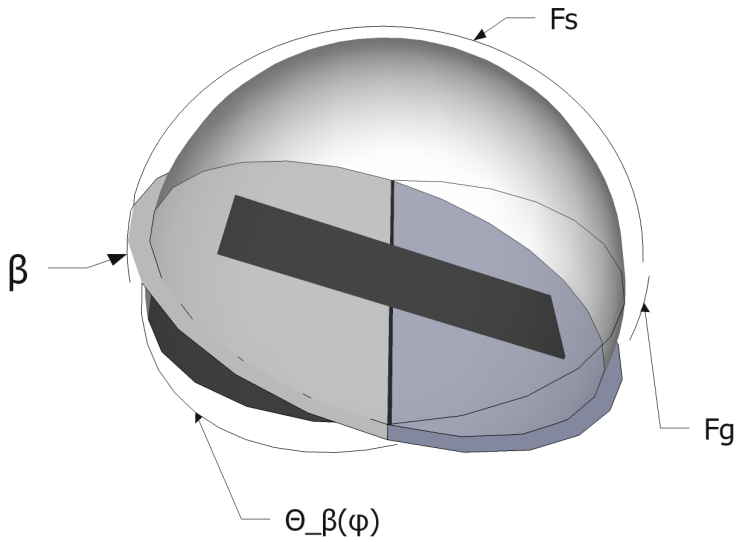
$$\Theta_\beta\left(\Phi\right) = \max\left[0°,\ \beta cos(\Phi - \gamma)\right] \tag{B.14}$$

der γ er asimutvinkelen. Det er gjort forskjell på innstråling fra bakke og horisont, siden disse kan ha forskjellig temperatur. Jordoverflaten vil trolig har en temperatur T_g lik utetemparaturen, mens horisonten ofte består av hus eller andre bygninger. Temperaturen T_h for horisonten vil dermed naturlig ligge noen grader over utetemperaturen T_a. Innstrålingen R_{inn} fra de forskjellige områdene innenfor synsfeltet kan dermed skrives som

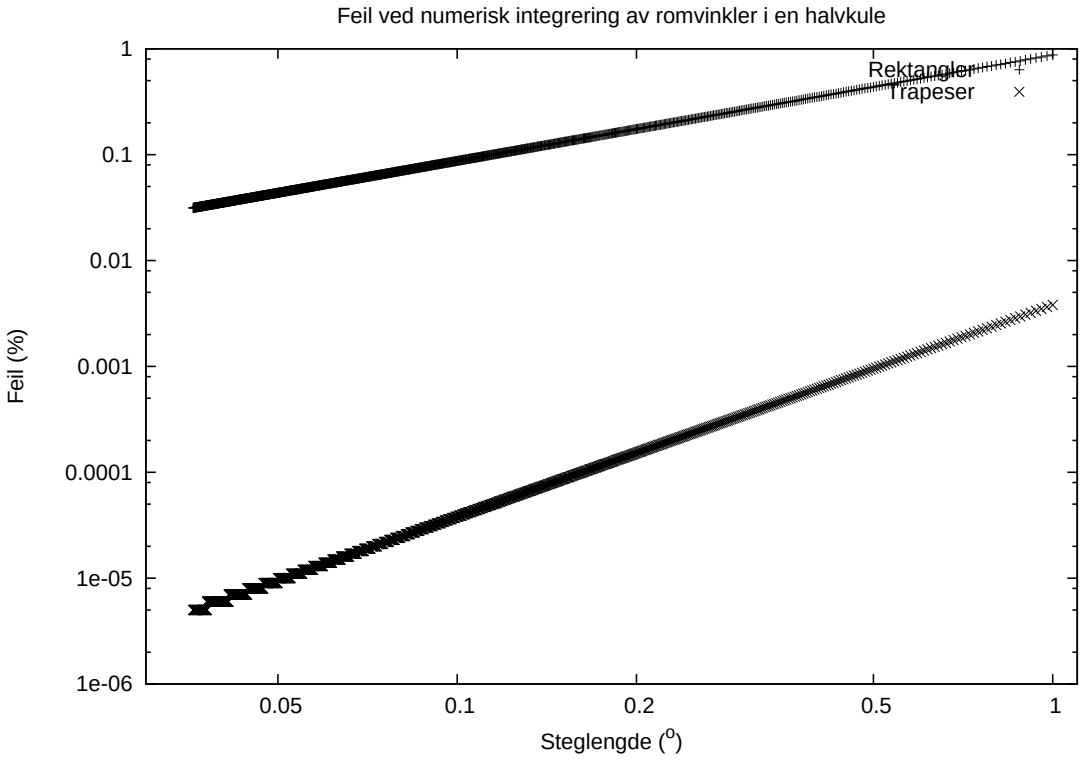
$$R_{inn} = R_{atm} + R_h + R_g = \sigma\left(W_{f,atm}\epsilon_{atm}T_a^4 + W_{f,h}\epsilon_h T_h^4 + W_{f,g}\epsilon_g T_g^4\right) \tag{B.15}$$

hvor R_{atm}, R_h og R_g er den termiske innstrålingen fra henholdsvis atmosfæren, horisonten og jordoverflaten.

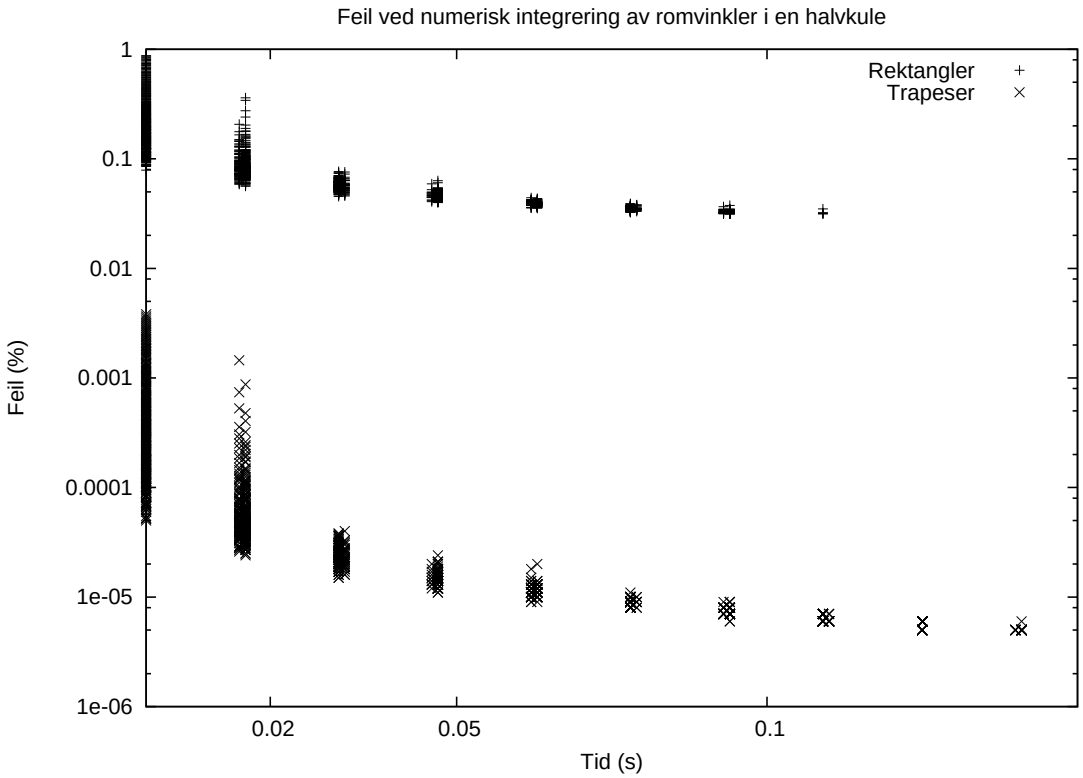
Denne metoden vil overestimere det totale arealet endel siden den ikke tar hensyn til at toppbredden vil være noe kortere enn bunnen. For steglengder opp mot 1° er feilen på 1%, men synker lineært mot $0,05\%$ ved steglengde på $0,03°$. figur B.5 viser at for å få en presisjon på $0,05\%$ kreves omtrent $2 - 3$ s med prosessortid på en moderne CPU.



Figur B.3: Eksempel for et panel med hellingsvinkel β. F_s og F_g er ikke vinkler, men synsfelt som definert i lign. 2.24 og 2.28



Figur B.4: Log-log plot av den relative feilen i prosent ved numerisk integrasjon som funksjon av steglengden. Steglengden minker omvendt proporsjonalt med antall steg.



Figur B.5: Log-log plott av feil ved numerisk integrasjon som funksjon av tiden integrasjonen tar.

Det er lett å se at trapesmetoden er ovelegen metoden som benytter rektangler. Ved steglengder rundt 1° er differansen i avviket mellom metodene på to størrelsesordner, mens den øker til over tre ved kortere steglengder. I programmet er det ikke behov for spesielt stor nøyaktighet, en feil på rundt 0,005% som inntreffer ved steglengder på 1° vil si at den trygt kan neglisjeres i forhold til andre feilkilder. For eksempel vil usikkerheten i horisontprofilen være mye høyere ettersom den er nødt til å tegnes opp for hånd og på øyemål.

Ut i fra figur B.5 vil en feil på mindre enn 0,001% kreve rundt 0,02 s ved prosessering. Integrasjon av halvkulen vil bli foretatt en gang per simulering for å finne normaliseringskonstanten. Det kun mulighet til å tegne opp en horisont i programmet og paneler som har forskjellige hellingvinkeler og asimutvinkler vil se samme horisont, slik at integrasjonen kun vil bli foretatt en gang per simulering. Dette utgjør en neglisjerbar effekt på simuleringstiden.

Tillegg C

Kildekode for innlesing og databehandling

C.1 Skyhøyde

```
1  package readnplot;
2
3  import java.io.File;
4
5  /**
6    * Main class for finding Cloud heights
7    */
8  public class Main {
9
10     public static void main(String[] args) throws IOException {
11         CloudHeights data = new CloudHeights("HL.txt");
12         data.calc();
13         data.showData();
14     }
15
16 }
17
18 public class CloudHeights extends sollib.Data {
19
20     private Hashtable<Date, Double> data;
21     private Vector<Date> dates;
22     private double[] series;
23     private int[] frequencies = new int[100];
24     private double[][] newHeights = new double[100][100];
25     GregorianCalendar now = (GregorianCalendar) GregorianCalendar.getInstance();
26
27     public CloudHeights(String filename) {
28         super(new File(filename));
29         data = new Hashtable<Date, Double>(20000);
30         dates = new Vector<Date>(20000);
31         int index, prevIndex = 0;
32         double cloudHeight;
33         String[] line = readLine(";");
34         now.set(parseInt(line[1]), parseInt(line[2]) - 1, parseInt(line[3]), parseInt(line[4]) -
               1, 0, 0);
35         Date date = now.getTime(), prevDate = date;
36         do {
37             if (line[5].equals("-")); else {
38                 cloudHeight = parseDouble(line[5]);
39                 data.put(date, cloudHeight);
40                 dates.add(date);
41                 index = (int) (cloudHeight / 100);
```

```
42                  frequencies[index]++;
43                  newHeights[prevIndex][index]++;
44                  prevIndex = index;
45              }
46              line = readLine(";");
47              try {
48                  now.set(parseInt(line[1]), parseInt(line[2]) - 1, parseInt(line[3]), parseInt(
                        line[4]) - 1, 0, 0);
49              } catch (NullPointerException e) {
50                  break;
51              }
52              prevDate = date;
53              date = now.getTime();
54          } while (line != null);
55
56
57          //Normalize
58          double sum;
59          for (int i = 0; i < frequencies.length; i++) {
60              sum = 0;
61              for (int j = 0; j < newHeights[i].length; j++) {
62                  sum += newHeights[i][j];
63              }
64
65              for (int j = 0; j < newHeights[i].length; j++) {
66                  newHeights[i][j] /= sum;
67              }
68          }
69      }
70
71      void calc() {
72
73
74
75          Formatter out = null;
76
77          try {
78              out = new Formatter(new File("heights.dat"));
79          } catch (FileNotFoundException ex) {
80              Logger.getLogger(CloudHeights.class.getName()).log(Level.SEVERE, null, ex);
81          }
82
83          out.format(Locale.ENGLISH, "#%1$15s %1$15s\n", "Height", "Relative frequency");
84          double Pakk = 0;
85          for (int i = 0; i < frequencies.length; i++) {
86              //if (frequencies[i] != 0) {
87              Pakk += frequencies[i];
88              out.format(Locale.ENGLISH, "%1$15f %2$15f\n", i / 10., Pakk);
89              //}
90          }
91
92          out.flush();
93          out.format("HÃ¸ydefordelinger\n%1$13s","$z_h$ (km) & ");
94          for (int i = 0; i < frequencies.length; i++) {
95              if (frequencies[i] > 0){
96                  out.format("%1$8.1f & ", i/10d);
97              }
98          }
99
100         double ch = 0;
101         for (int i = 0; i < newHeights.length; i++) {
102             if (MathSolar.average(newHeights[i]) > 0) {
103                 out.flush();
104
105                 out.format("\\\\\n%1$3.1f - %2$3.1f &",ch,(i / 10d));
106                 ch = (i / 10d);
107                 Pakk = 0;
108                 for (int j = 0; j < newHeights[i].length; j++) {
109                     if (newHeights[i][j] != 0) {
110                         Pakk += newHeights[i][j];
111                     out.format(Locale.ENGLISH,"%1$8.4f, ", Pakk);
112                     }
113                 }
114             }
115         out.flush();
116         }
117
```

```
118
119         out.close();
120
121     }
122
123     public void showData() {
124         for (int i : frequencies) {
125             System.out.println(i + " ");
126         }
127     }
128 }
```

C.2 Duggpunktstempertur

```
1
2  package dewpoint;
3
4  import java.io.File;
5  import java.io.FileNotFoundException;
6  import java.util.ArrayList;
7  import java.util.Arrays;
8  import java.util.Date;
9  import java.util.Formatter;
10 import java.util.GregorianCalendar;
11 import java.util.Hashtable;
12 import java.util.Locale;
13 import java.util.Vector;
14 import static java.lang.Double.parseDouble;
15 import static java.lang.Integer.parseInt;
16 import java.util.logging.Level;
17 import java.util.logging.Logger;
18 import sollib.MathSolar;
19
20 /**
21  *
22  * @author Espen Holtebu
23  */
24 public class DewPoint extends sollib.Data {
25
26     private static final int TD_MEAN = 0;
27     private static final int TD_AMP = 1;
28     private static final int TD_MODEL1 = 2;
29     private static final int TD_MODEL2 = 3;
30     private static final int TA_MEAN = 4;
31     private static final int TA_MAX = 5;
32     private static final int TA_MIN = 6;
33     private static final int TA_DEV = 7;
34     private static final int TD_MAX = 8;
35     private static final int TD_MIN = 9;
36     private Hashtable<Date, double[]> t;
37     private Vector<Date> dates;
38     GregorianCalendar now;
39     GregorianCalendar toMorrow;
40     private double latitude;
41     private double longitude;
42
43     /**
44      * @param args the command line arguments
45      */
46     public static void main(String[] args) {
47         DewPoint dp = new DewPoint("W:/master/data/TdOslo/TDOSLO.dat");
48         dp.importTs();
49         dp.print();//printDev();
50
51     }
52
53     private DewPoint(String fn) {
54         super(new File(fn));
55         latitude = 59.91;
56         longitude = 10.75;
57         t = new Hashtable<Date, double[]>(5000);
```

```
58          dates = new Vector<Date>(5000);
59          now = (GregorianCalendar) GregorianCalendar.getInstance();
60          toMorrow = (GregorianCalendar) GregorianCalendar.getInstance();
61      }
62
63      private void importTs() {
64
65
66
67          //Ambient temperatures
68          double Tam = 0000,
69                  Tan = 1000,
70                  Tax = -2000;
71
72          //Dew point temperatures
73          double Tdm = 0000,
74                  Tdn = 1000,
75                  Tdx = -2000;
76
77          double[] Ta = new double[24];
78          double[] Td = new double[24];
79          int Tidx = 0;
80
81          String[] line = readLine(";");
82
83          toMorrow.set(2008, 0, 1, 0, 0, 0);
84          Date date = toMorrow.getTime();
85          toMorrow.set(2008, 0, 2, 0, 0, 0);
86
87          Formatter out = null;
88          try {
89              out = new Formatter(
90              new File("w:/master/shared/data/tdseriesDiurn.dat"));
91          } catch (FileNotFoundException ex) {
92              Logger.getLogger(DewPoint.class.getName()).log(Level.SEVERE, null, ex);
93          }
94
95          do {
96              now.set(parseInt(line[1]), parseInt(line[2]) - 1,
97                parseInt(line[3]), parseInt(line[4]) - 1, 0, 0);
98              if (newDay()) { //Calculate values for the day
99                  System.out.println(date);
100                 //Previous day
101                 int dayOfYear = now.get(GregorianCalendar.DAY_OF_YEAR) - 1;
102                 double decl = MathSolar.getDeclination(dayOfYear);
103                 //Find sunset and sunrise (negative since going from solar time to local time)
104                 double addHour = -MathSolar.solarTimeDifference(
105                     dayOfYear, longitude, +1) / 60d;
106                 double sunsetHourAngle = Math.acos(
107                     -MathSolar.tan(latitude) * MathSolar.tan(decl)) * 180 / Math.PI;
108                 double sunsetSolarTime = 12 + sunsetHourAngle / 15d;
109                 double sunsetStdTime = sunsetSolarTime - addHour;
110                 double sunriseSolarTime = 12 - sunsetHourAngle / 15d;
111                 double sunriseStdTime = sunriseSolarTime - addHour;
112                 //Find length of night in full hours
113                 double lengthOfDay = sunsetStdTime - sunriseStdTime;
114
115                 for (int i = 0; i < Ta.length; i++) {
116                     //Find max Ta
117                     Tax = Math.max(Ta[i], Tax);
118                     //Find min Ta
119                     Tan = Math.min(Ta[i], Tan);
120                     //Find mean Ta
121                     Tam += Ta[i];
122                     //Find mean nightly Td values
123                     //if(i < sunriseStdTime || i > sunsetStdTime){
124                     //Find max Td
125                     Tdx = Math.max(Td[i], Tdx);
126                     //Find min Td
127                     Tdn = Math.min(Td[i], Tdn);
128                     //Find mean Td
129                     Tdm += Td[i];
130                     //lengthOfNight++;
131                     //}
132                 }
133                 Tam /= 24;
134                 Tdm /= 24;
```

```
135
136                 double TdAmp = (Tdx - Tdn);
137                 double deltaTd = TdAmp / 2;
138                 //Equaling minimun temperature
139                 double avgTdModel1 = Tan;
140                 double Tdn1 = avgTdModel1 - deltaTd;
141                 double Tdx1 = avgTdModel1 + deltaTd;
142                 //Hubbard et. al.
143                 double avgTdModel2 = -0.0360 * Tam + 0.9679 * Tan + 0.0072 * (Tax - Tan);
144                 double Tdn2 = avgTdModel2 - deltaTd;
145                 double Tdx2 = avgTdModel2 + deltaTd;
146                 //Add data
147                 t.put(date, new double[]{
148                     Tdm, TdAmp, avgTdModel1, avgTdModel2, Tam, Tax, Tan, Tax - Tan, Tdx, Tdn});
149                 dates.add(date);
150
151                 //Print Td time series
152                 double TdMod1 = 0;
153                 double TdMod2 = 0;
154                 double tau;
155                 out.format("#dayOfYear,i, Td[i], TdMod1, TdMod2,i\n");
156                 for (int i = 0; i < 24; i++) {
157                     tau = (i - sunriseStdTime - 1.5) / (lengthOfDay + 2.5);
158
159                     if (tau > 0 && tau < 2) {
160                         TdMod1 = Tdn1 + deltaTd * (Math.cos(tau * Math.PI) + 1);
161                         TdMod2 = Tdn2 + deltaTd * (Math.cos(tau * Math.PI) + 1);
162                     } else {
163                         TdMod1 = Tdx1;
164                         TdMod2 = Tdx2;
165                     }
166                     out.format(Locale.ENGLISH, "%1$3d %2$3d %3$15f %4$15f %5$15f\n",
167                         dayOfYear, i, Td[i], TdMod1, TdMod2, i);
168
169                 }
170                 out.format("\n\n");
171                 out.flush();
172
173                 //Reset values
174                 Tam = 0000;
175                 Tan = 1000;
176                 Tax = -2000;
177                 Tdm = 0000;
178                 Tdn = 1000;
179                 Tdx = -2000;
180                 Ta = new double[24];
181                 Td = new double[24];
182                 Tidx = 0;
183             }
184             date = now.getTime();
185             //System.out.println(date + " " + Tidx);
186             //Replace missing value with previous value (occurs twice)
187             if (line[5].equals("x")) {
188                 Ta[Tidx] = Ta[Tidx - 1];
189             } else {
190                 Ta[Tidx] = parseDouble(line[5]);
191             }
192             if (line[6].equals("x")) {
193                 Td[Tidx] = Td[Tidx - 1];
194             } else {
195                 Td[Tidx] = parseDouble(line[6]);
196             }
197             Tidx++;
198             line = readLine(";");
199         } while (line != null);
200         out.close();
201     }
202
203     private boolean newDay() {
204         if (now.before(toMorrow)) {
205             return false;
206         } else {
207             toMorrow.add(GregorianCalendar.DAY_OF_YEAR, 1);
208             return true;
209         }
210     }
211
```

```
212     private void print() {
213         Formatter out = null;
214 
215         try {
216             out = new Formatter(new File("W:/master/shared/data/td/tdOslo.dat"));
217         } catch (FileNotFoundException ex) {
218             Logger.getLogger(DewPoint.class.getName()).log(Level.SEVERE, null, ex);
219             System.exit(1);
220         }
221 
222         out.format(Locale.ENGLISH,
223             "  #%1$16s %2$15s %3$15s %4$15s %5$15s %6$15s %7$15s %8$15s %9$15s %10$15s\n",
224                 "TD_MEAN", "TD_MEAN_DEV", "TD_MODEL1", "TD_MODEL2", "TA_MEAN", "TA_MAX",
225                  "TA_MIN", "TA_DEV", "TD_MAX", "TD_MIN");
226         int i = 0, day = 0;
227 
228         int month = 0;
229         double[] observed = new double[31];
230         double[] predicted1 = new double[31];
231         double[] predicted2 = new double[31];
232         double[][] stats1 = new double[12][];
233         double[][] stats2 = new double[12][];
234         for (Date date : dates) {
235             double[] val = t.get(date);
236             out.format(Locale.ENGLISH,
237                     "%1$3d %2$15f %3$15f %4$15f %5$15f %6$15f %7$15f %8$15f %9$15f %10$15f %11
                            $15f\n", i++,
238                     val[TD_MEAN],
239                     val[TD_AMP],
240                     val[TD_MODEL1],
241                     val[TD_MODEL2],
242                     val[TA_MEAN],
243                     val[TA_MAX],
244                     val[TA_MIN],
245                     val[TA_DEV],
246                     val[TD_MAX],
247                     val[TD_MIN]);
248             out.flush();
249             if (date.getMonth() == month) {
250                 observed[day] = val[TD_MEAN];
251                 predicted1[day] = val[TD_MODEL1];
252                 predicted2[day++] = val[TD_MODEL2];
253             } else {
254                 stats1[month] = MathSolar.statistics(predicted1, observed);
255                 stats2[month] = MathSolar.statistics(predicted2, observed);
256                 observed = new double[31];
257                 predicted1 = new double[31];
258                 predicted2 = new double[31];
259                 month++;
260                 day = 0;
261             }
262         }
263         stats1[month] = MathSolar.statistics(predicted1, observed);
264         stats2[month] = MathSolar.statistics(predicted2, observed);
265         //Model 1 stats
266         out.format("\n\nStat metode 1\n");
267         out.format(Locale.ENGLISH,
268             "%1$10s & %2$10s & %3$10s & %4$10s & %5$10s & %6$10s & %7$10s \\\\\\n",
269             "MÃ¥ned","MAE (K)","RMSE (K)","r","$R^2$","E","d");
270         for (i = 0; i < stats1.length; i++) {
271             out.format(
272                 "%1$10d & %2$10.2f & %3$10.2f & %4$10.2f & %5$10.2f & %6$10.2f & %7$10.2f\\\\\n"
                        ,
273                 i+1,
274                 stats1[i][0],
275                 stats1[i][1],
276                 stats1[i][2],
277                 stats1[i][3],
278                 stats1[i][4],
279                 stats1[i][5]);
280         }
281         //Model 2 stats
282         out.format("\n\nStat metode 1\n");
283         out.format(Locale.ENGLISH,
284             "%1$10s & %2$10s & %3$10s & %4$10s & %5$10s & %6$10s & %7$10s \\\\\\n",
285             "MÃ¥ned","MAE (K)","RMSE (K)","r","$R^2$","E","d");
286         for (i = 0; i < stats2.length; i++) {
```

```
287             out.format(
288              "%1$10d & %2$10.2f & %3$10.2f & %4$10.2f & %5$10.2f & %6$10.2f & %7$10.2f\\\\\n",
289                 i+1,
290                 stats2[i][0],
291                 stats2[i][1],
292                 stats2[i][2],
293                 stats2[i][3],
294                 stats2[i][4],
295                 stats2[i][5]);
296         }
297         out.close();
298     }
299 
300     private void printDev() {
301         double[][] range = new double[24][t.size() / 24];
302 
303 
304         double[] avgDev = new double[24];
305         double[] stDev = new double[24];
306         double[] ts;
307         now.setTime(dates.get(0));
308         int today = now.get(GregorianCalendar.DATE);
309         boolean newDay;
310         int days = 0;
311         //Avereage deviation
312         for (Date date : dates) {
313             newDay = now.get(GregorianCalendar.DATE) != today;
314             if (newDay) {
315                 days++;
316             }
317             today = now.get(GregorianCalendar.DATE);
318 
319             ts = t.get(date);
320             now.setTime(date);
321             int idx = now.get(GregorianCalendar.HOUR_OF_DAY);
322             double deviation = ts[0] - ts[1];
323             //Average
324             avgDev[idx] += deviation;
325             stDev[idx] += deviation * deviation;
326             range[idx][days] = deviation;
327 
328         }
329 
330         Formatter out = null;
331         try {
332             out = new Formatter("c:/shared/data/td/tds_timer.dat");
333             out.format("%1$5s %2$10s\n", "Avvik", "Antall");
334         } catch (FileNotFoundException ex) {
335             Logger.getLogger(DewPoint.class.getName()).log(Level.SEVERE, null, ex);
336         }
337 
338         for (int i = 0; i < avgDev.length; i++) {
339             avgDev[i] /= days;
340             stDev[i] = Math.sqrt(stDev[i] / days - avgDev[i] * avgDev[i]);
341             Arrays.sort(range[i]);
342 
343             double lowest = range[i][0];
344             double highest = range[i][range[i].length - 1];
345             double dx = (highest - lowest) / 20;
346 
347             int[] values = new int[21];
348             int idx = 0;
349             for (int j = 0; j < range[i].length; j++) {
350                 if (range[i][j] > lowest + dx) {
351                     idx++;
352                     lowest += dx;
353                 }
354                 values[idx]++;
355             }
356 
357             out.format("\n\n#Time " + i + "\n");
358             for (int j = 0; j < values.length; j++) {
359                 out.format(Locale.ENGLISH, "%1$5f %2$10d\n", lowest, values[j]);
360                 lowest -= dx;
361             }
362             out.flush();
363 
```

```
364         }
365         out.close();
366         try {
367             out = new Formatter("c:/shared/data/td/tds.dat");
368             out.format("%1$5s %2$10s %3$10s\n", "Time", "dev", "stDev");
369             for (int i = 0; i < avgDev.length; i++) {
370                 out.format(Locale.ENGLISH, "%1$5d %2$10f %3$10f\n", i, avgDev[i], stDev[i]);
371             }
372             out.close();
373         } catch (FileNotFoundException ex) {
374             Logger.getLogger(DewPoint.class.getName()).log(Level.SEVERE, null, ex);
375         }
376
377
378
379     }
380 }
```

C.3 Periodelengder

```
1  package weatherperiodlengths;
2
3  import java.io.BufferedReader;
4  import java.io.File;
5  import java.io.FileNotFoundException;
6  import java.io.FileReader;
7  import java.io.IOException;
8  import java.util.Scanner;
9  import java.util.logging.Level;
10 import java.util.logging.Logger;
11
12 /**
13  * Reads data from file.
14  * @author Espen Holtebu
15  */
16 public class Data {
17     public static final String EMPTY_LINE = "NaN";
18
19     private PartOfTheCountry setPart() {
20         if (
21                 county.equals("Østfold") ||
22                 county.equals("Oslo") ||
23                 county.equals("Akershus") ||
24                 county.equals("Vestfold") ||
25                 county.equals("Hedmark") ||
26                 county.equals("Oppland") ||
27                 county.equals("Buskerud")
28            ) return PartOfTheCountry.OSTLANDET;
29         if (
30                 county.equals("Telemark") ||
31                 county.equals("Aust-Agder") ||
32                 county.equals("Vest-Agder")
33            ) return PartOfTheCountry.SORLANDET;
34                 if (
35                 county.equals("Hordaland") ||
36                 county.equals("Rogaland") ||
37                 county.equals("SognogFjordane")
38            ) return PartOfTheCountry.VESTLANDET;
39                 if (
40                 county.equals("MÃ¸reogRomsdal") ||
41                 county.equals("SÃ¸r-TrÃ¸ndelag") ||
42                 county.equals("Nord-TrÃ¸ndelag")
43            ) return PartOfTheCountry.MIDTNORGE;
44         if (
45                 county.equals("Nordland") ||
46                 county.equals("Troms") ||
47                 county.equals("Finnmark")
48            ) return PartOfTheCountry.NORDNORGE;
49                 else
50                     return PartOfTheCountry.UNSPECIFIED;
51
```

```
52      }
53
54
55
56      private String municipality;
57      private String county;
58      private PartOfTheCountry countryPart;
59      Scanner sc;
60
61
62      public Data(File file, String name){
63          String[] s = name.split(",");
64
65          municipality = remBlanks(s[0]);
66          System.out.println(municipality);
67          county = remBlanks(s[1]);
68          System.out.println(county);
69          countryPart = setPart();
70
71          try {
72              sc = new Scanner(new BufferedReader(new FileReader(file)));
73          } catch (FileNotFoundException ex) {
74              Logger.getLogger(Data.class.getName()).log(Level.SEVERE, null, ex);
75          } catch (IOException ex) {
76              Logger.getLogger(Data.class.getName()).log(Level.SEVERE, null, ex);
77          }
78      }
79
80      protected String remBlanks(String s) {
81          String[] strings = s.split(" ");
82          s = "";
83          for(int i = 0; i< strings.length; i++){
84              if(strings[i] !=null)
85              s+= strings[i];
86          }
87          return s;
88      }
89
90
91
92      public String getName(){
93          return county + "," + municipality;
94      }
95
96      public PartOfTheCountry getPart(){
97          return countryPart;
98      }
99
100
101     public String[] readLine(){
102         if (sc == null){
103             return null;
104         }
105         if(!sc.hasNext()){
106             sc.close();
107             sc = null;
108             return null;
109         }
110         String line[] = sc.nextLine().split("\\s");
111         //check for empty lines
112         if(line == null) {
113             String t[] = {EMPTY_LINE};
114             return t;
115         }
116
117         //Skip commented lines
118         while(line[0].startsWith("#")){
119              line = sc.nextLine().split("\\s");
120         }
121
122         return line;
123     }
124
125
126 }
```

```
1  package weatherperiodlengths;
2
3  import java.awt.FlowLayout;
4  import java.awt.event.ActionEvent;
5  import java.awt.event.ActionListener;
6  import java.io.File;
7  import java.io.IOException;
8  import java.util.ArrayList;
9  import java.util.logging.Level;
10 import java.util.logging.Logger;
11 import javax.swing.JButton;
12 import javax.swing.JFrame;
13
14 /**
15  * Main class
16  * @author ESpen Holtebu
17  */
18 public class Main extends JFrame {
19
20     private ArrayList<WeatherPeriods> periods;
21     private ArrayList<SolisData> solisData = null;
22     private File path;
23     JButton importBtn = new JButton("Import solis");
24     JButton periodsBtn = new JButton("Calculate periods");
25
26     /**
27      * @param args the command line arguments
28      */
29     public static void main(String[] args) throws IOException {
30
31         if (args.length != 1) {
32             throw new IllegalArgumentException("Enter full data path as argument");
33         }
34         new Main(args[0]);
35     }
36
37     private Main(String path) throws IOException {
38         super("Databehandler");
39         setLayout(new FlowLayout(FlowLayout.CENTER));
40         this.path = new File(path);
41         if (!this.path.exists()) {
42             throw new IllegalArgumentException("Wrong path entered");
43         }
44         System.out.println(path + " exists");
45         ButtonHandler handler = new ButtonHandler();
46         importBtn.addActionListener(handler);
47         add(importBtn);
48         periodsBtn.addActionListener(handler);
49         add(periodsBtn);
50         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
51         setBounds(40, 40, getWidth(), 100);
52         setVisible(true);
53     }
54
55     private void delFiles(File file) {
56         File list[] = file.listFiles();
57         for (File fil : list) {
58             if (fil.isDirectory()) {
59                 delFile(fil);
60                 fil.delete();
61             }
62         }
63     }
64
65     private void delFile(File file) {
66         File list[] = file.listFiles();
67         for (File fil : list) {
68             if (fil.isDirectory()) {
69                 delFile(file);
70             }
71             fil.delete();
72         }
73
74     }
75
76     private class ButtonHandler implements ActionListener {
77
```

```
78          public void actionPerformed(ActionEvent e) {
79              try {
80                  if (e.getSource() == periodsBtn) {
81                      calcPeriods();
82                  } else if (e.getSource() == importBtn) {
83                      importData();
84                  }
85              } catch (IOException ex) {
86                  System.err.println("IOEXception occured when trying: " + e.getSource());
87                  Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);
88              }
89          }
90
91          private void importData() throws IOException {
92              File[] dataFiles = path.listFiles();
93              solisData = new ArrayList<SolisData>(dataFiles.length);
94              File directory = path;
95              //delFiles(directory);
96              directory.mkdir();
97
98              for (File data : dataFiles) {
99                  if (data.isFile()) {
100                     String name = data.getName();
101                     int start = name.indexOf('(') + 1;
102                     int end = name.indexOf(')');
103                     name = name.substring(start, end);
104                     SolisData sd = new SolisData(data, name);
105                     sd.print(directory);
106                     solisData.add(sd);
107                 }
108             }
109
110         }
111
112         private void calcPeriods() throws IOException {
113
114             if (solisData == null) {
115                 System.out.println("Import first");
116                 return;
117             }
118
119             //Set the limit to 0.5 as was the convetion in the last project
120             //for (double cwiLimit = 0.5; cwiLimit < 0.90; cwiLimit += 0.05) {
121             double cwiLimit = 0.5;
122             periods = new ArrayList<WeatherPeriods>(40);
123             for (SolisData data : solisData) {
124                 periods.add(new WeatherPeriods(data, cwiLimit));
125             }
126             periods.trimToSize();
127
128
129             PeriodLengthController places[] = new PeriodLengthController[5];
130
131
132             for (WeatherPeriods wp : periods) {
133                 PartOfTheCountry cp = wp.getPartOfTheCountry();
134                 int idx = cp.ordinal();
135                 //new PeriodLengthController(wp).printDists();
136                 if (places[idx] == null) {
137                     places[idx] = new PeriodLengthController(path, wp, cp);
138                 } else {
139                     places[idx].addData(wp);
140                 }
141             }
142             for (PeriodLengthController plc : places) {
143                 if (!(plc == null)) {
144                     plc.printDists();
145                 }
146             }
147             periods = null;
148
149         }
150     }
151 }
```

```
1 package weatherperiodlengths;
```

```
2
3 /**
4  * Defines the parts of Norway
5  * @author Espen Holtebu
6  */
7 public enum PartOfTheCountry {
8     OSTLANDET, VESTLANDET, SORLANDET, MIDTNORGE, NORDNORGE, UNSPECIFIED};
```

```
1  package weatherperiodlengths;
2
3  /**
4   * Specific weather periods
5   * @author Espen Holtebu
6   */
7  public class Period {
8      public static final int WINTER = 0;
9      public static final int SPRING = 1;
10     public static final int SUMMER = 2;
11     public static final int AUTUMN = 3;
12
13     private boolean good;
14     private int length;
15     /**Length of the next period, -1 indicates no information*/
16     private int nextLength = -1;
17     private double avgCwiNext;
18     private int dayOfYear;
19
20     public Period(boolean good, int length, double cwiLimit,double avgCwi, int dayOfYear){
21         this.good = good;
22         this.length = length;
23         this.dayOfYear = dayOfYear;
24     }
25
26
27     public boolean good(){
28         return good;
29     }
30
31     public int getLength(){
32         return length;
33     }
34
35     public void setNextAvgCwi(double avgCwiNext){
36         this.avgCwiNext = avgCwiNext;
37     }
38
39     public double getNextAvgCwi(){
40         return  avgCwiNext;
41     }
42
43     public void setNextLength(int nL){
44         this.nextLength = nL;
45     }
46
47     public int getNextLength(){
48         return nextLength;
49     }
50
51     public int getDayOfYear(){
52         return dayOfYear;
53     }
54
55
56
57
58 }
```

```
1 package weatherperiodlengths;
2
3 import java.io.IOException;
4 import java.io.Serializable;
5 import static java.lang.Math.*;
6 import java.util.Formatter;
7 import java.util.Locale;
8 import java.util.Random;
```

```
9
10 /**
11  * Calculates the distribustion of periods
12  * @author Espen Holtebu
13  */
14 public class PeriodDistribution implements Serializable {
15
16     /**The distribution of of lengths in the next period.
17      * A period is assumed to hav a length equal to maxLength or shorter*/
18     private double nextLengths[];
19     private double totPeriods;
20     private double q;
21     private double beta;
22     private double chiSquare;
23     private double c;
24     private double beta2;
25     private int missing = 0;
26
27     PeriodDistribution(int maxLength) {
28         nextLengths = new double[maxLength];
29     }
30
31     /**
32      * Adds a period with a specific length to the distribution
33      * @param length
34      */
35     void addLength(int length) {
36         if (length < 1 || length > nextLengths.length) {
37             throw new IllegalArgumentException(
38                     "Period length not withing permittable range: " + length);
39         }
40         nextLengths[length - 1]++;
41
42     }
43
44     /**
45      * Adds to the total of missing periods
46      */
47     void addMissing(){
48         this.missing++;
49     }
50
51     /**
52      * Calulates the total number of periods
53      */
54     private void totalPeriods() {
55         totPeriods = 0;
56         for (int i = 0; i < nextLengths.length; i++) {
57             totPeriods += nextLengths[i];
58         }
59     }
60
61     /**
62      * Normalize the distribution
63      */
64     private void normalize() {
65         for (int i = 0; i < nextLengths.length; i++) {
66             nextLengths[i] /= totPeriods;
67         }
68     }
69
70     /**
71      * Estimates parameters for the Weibull distribution
72      */
73     void estimateParameters() {
74         totalPeriods();
75         q = 1 - nextLengths[0] / totPeriods;
76         beta = Math.log(
77                 Math.log(q - nextLengths[1] / totPeriods) / Math.log(q)
78             ) / Math.log(2);
79         normalize();
80         chiSquare = 0;
81         for (int perLength = 3; perLength < 9; perLength++) {
82             double Fl = F(perLength) * totPeriods;
83             double diff = nextLengths[perLength - 1] - Fl;
84             chiSquare += diff * diff / Fl;
85         }
```

```
86          chiSquare /= 13;
87
88      }
89
90      private double p(double l) {
91          return Math.pow(q, Math.pow(l - 1, beta)) -
92                  Math.pow(q, Math.pow(l, beta));
93      }
94
95      private double par(double l) {
96          return Math.pow(1 + (c*l),- beta2);
97      }
98
99      private double F(double l) {
100         return 1 - pow(q, pow(l, beta));
101     }
102
103     double pickNext() {
104         return 0;
105     }
106
107     /**Prints the distribution of period lengths to the stream given by distFile
108      * @param distFile - the formatted OutputStream
109      * @param format   - The format of the data
110      * @throws java.io.IOException
111      */
112     void saveDistributionToFile(Formatter distFile) throws IOException {
113         estimateParameters();
114
115         String formatS = "\n%1$7s   %2$13s   %3$23s";
116         String formatNumbers = "\n%1$7d   %2$13f   %3$23f";
117
118         distFile.format("#Antall data for denne fordelingen er " + totPeriods);
119         distFile.format("\n#Modellen har parametere: q = " + q + " og beta = " +
120                         beta + ". Chi square er " + chiSquare);
121         distFile.format("\n#Summen av de 100 fÃ¸rste elementene i modellen: " +
122                         sumModel());
123         distFile.format(formatS, "#Lengde",
124                                  "Sannsynlighet", "Modellert sannsynlighet");
125         for (int i = 0; i < nextLengths.length; i++) {
126             double p = p(i + 1);
127             distFile.format(Locale.ENGLISH, formatNumbers, i + 1, nextLengths[i], p);
128         }
129
130         distFile.format("\n");
131     }
132
133     /**
134      * Saves the Weibull probability plot data to the stream distFile
135      * @param distFile the stream
136      * @throws IOException
137      */
138     void saveWPPToFile(Formatter distFile) throws IOException {
139
140         String formatS = "\n%1$7s   %2$13s   %3$23s";
141         String formatNumbers = "\n%1$7f   %2$13f   %3$23f %4$23f";
142         distFile.format("#Antall data for denne fordelingen er " + totPeriods +
143                         " Mangler " + missing + " andel mangel: " +
144                         (double)missing/totPeriods);
145         distFile.format("\n#Modellen har parametere: q = " + q + " og beta = " +
146                         beta + ". Chi square er " +
147                         chiSquare + " med 13 frihetsgrader-");
148         distFile.format("\n#Pareto har parametere: c = " + c +
149                         " og beta = " + beta2);
150         distFile.format("\n#Summen av de 100 fÃ¸rste elementene i modellen: " +
151                        sumModel());
152         distFile.format(formatS, "#Lengde",
153                        "Sannsynlighet", "Modellert sannsynlighet");
154         double Pakk = 0;
155         double modelPakk = 0;
156         for (int i = 0; i < nextLengths.length; i++) {
157             modelPakk += p(i + 1);
158             Pakk += nextLengths[i];
159             //Ln or not ln(i+1)
160             distFile.format(Locale.ENGLISH, formatNumbers,
161                     i + 1.,
162                     Math.log(-Math.log(1 - Pakk)),
```

```
163                     Math.log(-Math.log(1 - modelPakk)),
164                     Math.log(-Math.log(par(i+1)))
165                     );
166         }
167
168         distFile.format("\n");
169     }
170
171     private double sumModel() {
172         double sum = 0;
173         for (int i = 1; i < 100; i++) {
174             sum += p(i);
175         }
176         return sum;
177     }
178
179 }
```

```
1  package weatherperiodlengths;
2
3  import java.io.File;
4  import java.io.IOException;
5  import java.text.DecimalFormat;
6  import java.util.Arrays;
7  import java.util.Formatter;
8  import java.util.LinkedList;
9  import java.util.Locale;
10 import java.util.Random;
11 import java.util.logging.Level;
12 import java.util.logging.Logger;
13
14 /**
15  *
16  * @author Administrator
17  */
18 public class PeriodLengthController {
19     //static constants
20     private static DecimalFormat df = new DecimalFormat("0.00");
21
22     public static final int GOOD = 0;
23     public static final int BAD = 1;
24     public static final int WINTER = 0;
25     public static final int SPRING = 1;
26     public static final int SUMMER = 2;
27     public static final int AUTUMN = 3;
28     //Values from Gjermund
29     /**End of winter*/
30     private static final int WINTER_LMT = 12;
31     /**End of spring*/
32     private static final int SPRING_LMT = 82;
33     /**End of summer*/
34     private static final int SUMMER_LMT = 240;
35     /**End of autumn*/
36     private static final int AUTUMN_LMT = 311;
37     private static final int MAX_PERIOD_LENGTH = 200;
38     private String location;
39     private double cwiLimit;
40     private PeriodDistribution[][][][] dist;
41     private double avgCWI[];
42     private double nextGoodAvgCWI[];
43     private double nextBadAvgCWI[];
44     private File path;
45     private PeriodDistribution totalGood;
46     private PeriodDistribution totalBad;
47
48     /** Constructor to save new distribuitons from irradiation meassurements*/
49     public PeriodLengthController(File path, WeatherPeriods periods) {
50         location = periods.getStation();
51         //latitude = periods.getLatitude();
52         //longitude = periods.getLongitude();
53         cwiLimit = periods.getCwiLimit();
54         this.path = path;
55         initDistributions();
56         addData(periods);
57         serialize();
58     }
```

```
59
60      /** Constructor to save new distribuitons from irradiation meassurements*/
61      public PeriodLengthController(File path, WeatherPeriods periods, PartOfTheCountry p) {
62          this(path,periods);
63          this.location = p.name();
64      }
65
66      /** Constructor that deserializes distributions from file
67       * @param path - path to file containing distributions
68       */
69      public PeriodLengthController(String path, Random rng) {
70
71      }
72
73      public void addData(WeatherPeriods wp) {
74          double[] addCwis = wp.getCWIS();
75          for (int i = 0; i < avgCWI.length; i++) {
76              avgCWI[i] += addCwis[i];
77          }
78
79          LinkedList<LinkedList<Period>> periodsOfPeriods = wp.getPeriods();
80          int weatherType, season, length, nextLength;
81          for(LinkedList<Period> periods: periodsOfPeriods){
82              for(Period period: periods){
83                  weatherType = period.good() ? GOOD : BAD;
84                  season = getSeason(period.getDayOfYear());
85                  length = (int) Math.min(4, period.getLength());
86                  nextLength = period.getNextLength();
87                  if (nextLength > 0) {
88                      //No season analysis
89                      dist[weatherType][0][length-1].addLength(nextLength);
90                      switch(weatherType){
91                      case GOOD:
92                          totalGood.addLength(nextLength);
93                          break;
94                      case BAD:
95                          totalBad.addLength(nextLength);
96                          break;
97                      default:
98                          throw new IllegalArgumentException(
99                                  "Good or bad weather only (what else?)");
100                     }
101                 } else {
102                     dist[weatherType][0][length-1].addMissing();
103                     switch(weatherType){
104                     case GOOD:
105                         totalGood.addMissing();
106                         break;
107                     case BAD:
108                         totalBad.addMissing();
109                         break;
110                     default:
111                         throw new IllegalArgumentException(
112                                 "Good or bad weather only (what else?)");
113                     }
114                 }
115                 //Save total distribution
116
117             }
118         }
119     }
120
121     /** Calculates the season based on the day of year*/
122     private int getSeason(int dayOfYear) {
123         if(dayOfYear < 90 || dayOfYear > 270)
124             return 0;
125         return 1;
126 //        if (dayOfYear < WINTER_LMT) {
127 //            return WINTER;
128 //        } else if (dayOfYear < SPRING_LMT) {
129 //            return SPRING;
130 //        } else if (dayOfYear < SUMMER_LMT) {
131 //            return SUMMER;
132 //        } else if (dayOfYear < AUTUMN_LMT) {
133 //            return AUTUMN;
134 //        } else {
135 //            return WINTER;
```

```
136 //        }
137     }
138 
139     private void initDistributions() {
140 
141         avgCWI = new double[25];
142         totalGood = new PeriodDistribution(MAX_PERIOD_LENGTH);
143         totalBad  = new PeriodDistribution(MAX_PERIOD_LENGTH);
144         dist = new PeriodDistribution[2][1][4];
145         for (int i = 0; i < dist.length; i++) {
146             for (int j = 0; j < dist[i].length; j++) {
147                 for (int k = 0; k < dist[i][j].length; k++) {
148                     dist[i][j][k] = new PeriodDistribution(MAX_PERIOD_LENGTH);
149                 }
150             }
151         }
152     }
153 
154     public void printDists() {
155         try {
156             path.mkdir();
157             File outDir = new File(path,"/CWI/");
158             outDir.mkdir();
159             File out = new File(outDir,"cwis"+location+df.format(cwiLimit)+"_good.dat");
160             out.createNewFile();
161             Formatter distOut = new Formatter(out);
162             String formatString = "\n%1$8s %2$8s %3$8s";
163             String formatNumbers = "\n%1$8d %2$8f %3$8f";
164             String formatNumbers2 = "\n%1$8f %2$6f";
165 
166 //              normalizeCWIs();
167 
168             distOut.format(formatString, "Cwi", "Observed", "");
169             saveCWIDistributionToFile(distOut, formatNumbers2, avgCWI);
170 //              out = new File(outDir,"cwis"+location+df.format(cwiLimit)+"_bad.dat");
171 //              out.createNewFile();
172 //              distOut = new Formatter(out);
173 //              saveCWIDistributionToFile(distOut, formatNumbers2, nextGoodAvgCWI, cwiLimit);
174 
175             outDir = new File(path,"/periods/"+location);
176             outDir.mkdirs();
177             for (int i = 0; i < dist.length; i++)
178                 for (int j = 0; j < dist[i].length; j++)
179                     for (int k = 0; k < dist[i][j].length; k++)
180                         print(
181                                 outDir,
182                                 (j == 0 ? "Vinter" : "Sommer")+"-"+(i == GOOD ? "Bra" : "Daarlig
                                        ")+"-"+(k+1)+"-dager",
183                                 dist[i][j][k]);
184 
185             print(outDir, location + "totalGOOD", totalGood);
186             print(outDir, location + "totalBAD", totalBad);
187         } catch (IOException ex) {
188             Logger.getLogger(PeriodLengthController.class.getName()).log(Level.SEVERE, null, ex)
                    ;
189         }
190         System.out.println(location + "written to file!");
191     }
192 
193     private void print(File dir, String name, PeriodDistribution dist) throws IOException{
194         File outFile = new File(dir,name + ".dat");
195         outFile.createNewFile();
196         Formatter out = new Formatter(outFile);
197         System.out.println("SAving " + location);
198         dist.saveDistributionToFile(out);
199         out.flush();out.close();outFile = null;
200 
201         outFile = new File(dir, name + "WPP.dat");
202         outFile.createNewFile();
203         out = new Formatter(outFile);
204         dist.saveWPPToFile(out);
205         out.flush();out.close();outFile = null;
206     }
207 
208     void saveCWIDistributionToFile(Formatter distFile, String format, double[] array) throws
            IOException{
209         normalizeCWIs();
```

```
210         double cwi = 0;
211         for (int i = 0; i < array.length; i++) {
212             cwi = i*0.05;
213             distFile.format(Locale.ENGLISH,format, cwi,  array[i]);
214         }
215         distFile.flush();
216         distFile.close();
217     }
218 
219     private void serialize() {
220         //Not yet implemented
221     }
222 
223     private void normalizeCWIs(){
224         //int badLimit = (int)(cwiLimit/0.05 +1);
225         //nextBadAvgCWI = Arrays.copyOf(avgCWI, badLimit);
226         //nextGoodAvgCWI = Arrays.copyOfRange(avgCWI, badLimit, avgCWI.length);
227         double sum = sum(avgCWI);
228         for (int i = 0; i < avgCWI.length; i++) {
229             avgCWI[i]/=sum;
230         }
231 //          sum = sum(avgCWI);
232 //          for (int i = 0; i < nextGoodAvgCWI.length; i++) {
233 //              nextGoodAvgCWI[i]/=sum;
234 //          }
235     }
236 
237     private double sum(double[] array){
238         double sum = 0;
239         for (int i = 0; i < array.length; i++) {
240             sum += array[i]*0.05;
241         }
242         return sum;
243     }
244 
245     /**
246      * Randomly selects the next period by monte carlo, based upon the
247      * distribution corresponding to the given parameters
248      * @param weatherType -
249      * @param season
250      * @param length
251      * @return
252      */
253     public double pickNext(int weatherType, int season, int length) {
254         return dist[weatherType][season][length].pickNext();
255     }
256 
257 
258 }
```

```
1  package weatherperiodlengths;
2  
3  import java.io.Serializable;
4  import java.text.DecimalFormat;
5  import java.util.Date;
6  import java.util.Formatter;
7  import java.util.GregorianCalendar;
8  import java.util.Locale;
9  
10 /**
11  * Holds information about a specific day
12  * @author Espen Holtebu
13  */
14 class SData implements Serializable {
15     public static final double MAX_IRR = 9000;
16     private static int CWI_FACTOR = 1;
17     private double irr;
18     private double cwIrr;
19     private double cwi;
20     private double Ta;
21     private boolean good;
22     private int dayOfYear;
23 
24 
25     SData(int dayOfYear, double irradition, double Ta) {
26         //this.date = (GregorianCalendar) now.clone();
```

```
27          this.dayOfYear = dayOfYear;
28          this.irr = irradition;
29          this.cwIrr = Double.NaN;
30          this.Ta = Ta;
31      }
32
33
34      /**
35       *
36       * @return the day of year {0,364}
37       */
38      public int getDayOfYear() {
39          return dayOfYear;
40      }
41
42      public double getDailyIrradiance() {
43          return this.irr;
44      }
45
46      public double getCWIrr() {
47          return cwIrr;
48      }
49
50      public void setCWIrr(double cwIrr) {
51          this.cwIrr = cwIrr;
52          this.cwi = irr*CWI_FACTOR/cwIrr;
53          this.good = cwi >= 0.5 * CWI_FACTOR;
54      }
55
56      public double getCWI() {
57          return this.cwi;
58      }
59
60      public boolean good() {
61          return good;
62      }
63
64      public double getTa() {
65          return this.Ta;
66      }
67      private static final String dayDF = "%1$14d";
68      private static final String irrDF = "%1$15f";
69      private static final String cwIrrDF = "%1$28f";
70      private static final String cwiDF = "%1$15f";
71      private static final String goodDF = "%1$8d";
72      private static final String taDF = "%1$23f";
73
74      public String toString(Formatter formatter) {
75          /*
76           #Day Of year
77           #Irradiation
78           #Clear weather irradiance
79           #K_T, cs
80           #Good
81           #Ambient temperature" */
82          return formatter.format(dayDF, dayOfYear).
83                  format(irrDF, irr).
84                  format(cwIrrDF, cwIrr).
85                  format(cwiDF, cwi).
86                  format(goodDF, good ? 10 : 0).
87                  format(taDF + "    0\n", Ta).toString();
88      }
89  }
```

```
1  package weatherperiodlengths;
2
3  import java.io.BufferedWriter;
4  import java.io.EOFException;
5  import java.io.File;
6  import java.io.FileInputStream;
7  import java.io.FileNotFoundException;
8  import java.io.FileOutputStream;
9
10 import java.io.FileWriter;
11 import java.io.IOException;
12 import java.io.ObjectInputStream;
```

```
13 import java.io.ObjectOutputStream;
14 import java.util.ArrayList;
15 import java.util.Arrays;
16 import java.util.Formatter;
17 import java.util.GregorianCalendar;
18 import java.util.Locale;
19 import java.util.logging.Level;
20 import java.util.logging.Logger;
21 import static java.lang.Math.*;
22 import static weatherperiodlengths.SolarMath.*;
23
24 /**
25  * Imports data and calculates clear weather irradiance
26  * @author Espen Holtebu
27  */
28 public class SolisData extends Data {
29
30     public static final int GARDERMOEN = 4780;
31     public static final int FYS = 18700;
32     public static final int RANA = 79530;
33     public static final int TROMSØ = 90450;
34     private static final double MAX_IRR = 9000;
35     private int replaced = 0;
36     private double latitude;
37     private double longitude;
38     private double extIrr_T[] = new double[365];
39     private double extIrr_[] = new double[365];
40     private double cwIrrTheory_T[] = new double[365];
41     private double cwIrrTheory_[] = new double[365];
42     private double maxIrr = 0, minIrr = 1e100;
43     //Found by inspecting plots
44     private int maxIrrDay = 172, minIrrDay = 354;
45     private double cwIrrIntrp[] = new double[365];
46     private double maximums[];
47     private ArrayList<SData> data;
48     private File sdFile;
49
50     ArrayList<SData> getDatas() {
51         return data;
52     }
53
54     public double getLatitude() {
55         return latitude;
56     }
57
58     public double getLongitude() {
59         return longitude;
60     }
61
62     private boolean serialized() {
63         return sdFile.exists();
64     }
65
66     private void serialize() {
67         ObjectOutputStream out = null;
68
69         try {
70             sdFile.createNewFile();
71             out = new ObjectOutputStream(new FileOutputStream(sdFile));
72             for (SData sdata : data) {
73                 out.writeObject(sdata);
74             }
75             out.close();
76         } catch (IOException exc) {
77             Logger.getLogger(SolisData.class.getName()).log(Level.SEVERE, null, exc);
78         }
79     }
80
81     private void readSerializedData() {
82         ObjectInputStream in = null;
83         try {
84             in = new ObjectInputStream(new FileInputStream(sdFile));
85             while (true) {
86                 data.add((SData) in.readObject());
87             }
88         } catch (EOFException eofex) {
89             System.out.println(eofex.getMessage());
```

```
90              return;
91          } catch (IOException ex) {
92              ex.printStackTrace();
93          } catch (ClassNotFoundException ex) {
94              ex.printStackTrace();
95          }
96      }
97
98      /**
99       * Finds clear weather irradiation by theory
100      */
101     private void calcCWs() {
102         double lat = latitude;
103         //Convert to radians:
104         double deg2rad = PI / 180;
105         lat *= deg2rad;
106         double tiltA = PI / 4;
107
108         double vfGround = viewFactorGround(tiltA);
109         double vfSky = viewFactorSky(tiltA);
110         double albedo = 0.25;
111         double sin3beta2 = pow(sin(tiltA / 2), 3);
112
113         for (int dayOfYear = 0; dayOfYear < cwIrrTheory_T.length; dayOfYear++) {
114             cwIrrTheory_T[dayOfYear]
115                     = cwIrrTheory_[dayOfYear]
116                     = extIrr_T[dayOfYear]
117                     = extIrr_[dayOfYear] = 0;
118             double hourA = -PI;
119             double dHourA = 0.01;
120             //Timestep in hours
121             double dt = (dHourA / deg2rad) / 15;
122             double cosAOI;
123             double B = B(dayOfYear);
124             double decl = declination(B);
125             //W/m^2
126             double G_on = extRad(B);
127             double transmittance;
128             double cosThetaZ;
129
130
131             while (hourA < PI) {
132                 cosThetaZ = cosThetaZ(lat, decl, hourA);
133                 transmittance = transmittance(cosThetaZ);
134                 //A few parts vanish since Azimuth angle is zero
135                 cosAOI = cosAOIsouth(lat, decl, tiltA, hourA);
136
137                 if (cosThetaZ > 0) {
138                     double G_n = G_on * transmittance;
139                     double k = 1 / transmittance;
140                     double d = diffuseFraction(k, cosThetaZ);
141                     double G_beam_n = G_n * (1 - d) * (1 + k * d);
142
143                     double G_beam = G_beam_n * cosThetaZ;
144                     double G_beam_T = G_beam_n * cosAOI;
145
146                     double G_diffuse = G_n * cosThetaZ * d * (1 - (1 - d) * k);
147                     double G_diffuse_T = G_diffuse * vfSky * (1 + sqrt(1 - d) * sin3beta2);
148
149                     double G_refl_T = vfGround * albedo * G_on * cosThetaZ * transmittance;
150
151                     extIrr_[dayOfYear] += G_on * cosThetaZ;
152                     cwIrrTheory_[dayOfYear] += G_beam + G_diffuse;
153
154                     if (cosAOI > 0) {
155                         extIrr_T[dayOfYear] += G_on * cosAOI;
156                         cwIrrTheory_T[dayOfYear] += G_beam_T;
157                     }
158
159                     cwIrrTheory_T[dayOfYear] += G_diffuse_T + G_refl_T;
160                 }
161
162
163                 hourA += dHourA;
164             } //End while
165             cwIrrTheory_T[dayOfYear] *= dt;
166             cwIrrTheory_[dayOfYear] *= dt;
```

```
167             extIrr_[dayOfYear] *= dt;
168             extIrr_T[dayOfYear] *= dt;
169         }//End for
170     }
171
172     /**
173      * Finds clear weather irradiation by interpolation between days of maximum
174      * irradiation.
175      */
176     private void interpolateCWs() {
177         //interpolation method
178         //Fill with max daily values for all years
179
180         for (SData day : data) {
181             if (day != null) {
182                 int dayofyear = day.getDayOfYear();
183                 cwIrrIntrp[dayofyear] = max(cwIrrIntrp[dayofyear], day.getDailyIrradiance());
184             }
185         }//End for
186
187
188         maximums = Arrays.copyOf(cwIrrIntrp, cwIrrIntrp.length);
189
190         //Find maximum and minimum irradiation
191         for (int i = minIrrDay - 5; i < minIrrDay + 5; i++) {
192             minIrr = min(minIrr, cwIrrIntrp[i]);
193         }
194         cwIrrIntrp[minIrrDay] = minIrr;
195         cwIrrIntrp[minIrrDay - 1] = minIrr; //For convinience
196         for (int i = 0; i < cwIrrIntrp.length; i++) {
197             maxIrr = max(maxIrr, cwIrrIntrp[i]);
198         }
199         cwIrrIntrp[maxIrrDay] = maxIrr;
200
201         //Make array that starts at minIrrDay
202         double[] skewed = new double[cwIrrIntrp.length];
203         for (int i = 0; i < skewed.length; i++) {
204             int irrIdx = i + minIrrDay;
205             if (irrIdx < 0) {
206                 irrIdx += 365;
207             }
208             if (irrIdx > 364) {
209                 irrIdx -= 365;
210             }
211             skewed[i] = cwIrrIntrp[irrIdx];
212         }
213
214         //Fill up mac values for rising and decending Irradiance
215         int maxIrrSkewedDay = minIrrDay - maxIrrDay;
216         skewed[maxIrrSkewedDay] = maxIrr;
217         int peakIdx = 0;
218         int lastPeakIdx = peakIdx;
219         int idx, dt;
220         double dH, dH7dt;
221         for (int i = 0; i <= maxIrrSkewedDay; i++) {
222             idx = i;
223             dH = skewed[idx] - skewed[peakIdx];
224             if (dH >= 0) {
225                 peakIdx = idx;
226                 dt = peakIdx - lastPeakIdx;
227                 dH7dt = dH / dt;
228
229                 //Replace by interpolating
230                 idx = lastPeakIdx + 1;
231                 while (idx < peakIdx) {
232                     skewed[idx] = skewed[idx - 1] + dH7dt;
233                     idx++;
234                 }
235                 lastPeakIdx = peakIdx;
236             }
237         }
238
239         peakIdx = 364;
240         lastPeakIdx = peakIdx;
241         for (int i = 364; i >= maxIrrSkewedDay; i--) {
242             idx = i;
243             dH = skewed[idx] - skewed[peakIdx];
```

```
244             if (dH >= 0) {
245                 peakIdx = idx;
246                 dt = peakIdx - lastPeakIdx;
247                 dH7dt = dH / dt;
248 
249                 //Replace by interpolating
250                 idx = peakIdx + 1;
251                 while (idx < lastPeakIdx) {
252                     skewed[idx] = skewed[idx - 1] + dH7dt;
253                     idx++;
254                 }
255 
256                 lastPeakIdx = peakIdx;
257             }
258         }
259 
260         for (int i = 0; i < cwIrrIntrp.length; i++) {
261             int skewedIdx = i - minIrrDay;
262             if (skewedIdx < 0) {
263                 skewedIdx += 365;
264             }
265             if (skewedIdx > 364) {
266                 skewedIdx -= 365;
267             }
268 
269             cwIrrIntrp[i] = skewed[skewedIdx];
270         }
271 
272     }
273 
274     private void printCWs(File directory) throws IOException {
275         File dir = new File(directory, "CW/");
276         dir.mkdir();
277         File file;
278         int n = 1;
279         do {
280             file = new File(dir, getName() + n++ + ".dat");
281         } while (file.exists());
282         if (!file.createNewFile()) {
283             throw new IOException("Cant create file");
284         }
285         Formatter f = new Formatter(file);
286 
287         f.format("#" + replaced + " replaced spike(s)\n");
288         f.format("#Day of year           #Maximums       #Interpolated"+
289             "    #Theory at angle  #Theory horizontal           #Ext rad\n");
290 
291         final String intDF = "%1$12d";
292         final String irrDF = "%1$20.2f";
293         for (int i = 0; i < cwIrrIntrp.length; i++) {
294             f.format(Locale.ENGLISH,intDF, i);
295             f.format(Locale.ENGLISH,irrDF, maximums[i]);
296             f.format(Locale.ENGLISH,irrDF, cwIrrIntrp[i]);
297             f.format(Locale.ENGLISH,irrDF, cwIrrTheory_T[i]);
298             f.format(Locale.ENGLISH,irrDF, cwIrrTheory_[i]);
299             f.format(Locale.ENGLISH,irrDF + "\n", extIrr_T[i]);
300         }
301         f.flush();
302         f.close();
303     }
304 
305 
306     public SolisData(File file, String name) {
307         super(file, name);
308         System.out.print(getName());
309         data   = new ArrayList<SData>(1000);
310         sdFile = new File(getName() + ".obj");
311 
312         //if (serialized()) {
313         //    readSerializedData();
314         //} else {
315             readData();
316             interpolateCWs();
317             setCW(cwIrrIntrp);
318             serialize();
319         //}
320         System.out.println(" Done!");
```

```
321     }
322
323     public File sdFile() {
324         return sdFile;
325     }
326
327     private void readData() {
328         //Variable declarations
329         String line[], dateValues[];
330         int year, month, day, dayOfYear;
331         double irr, Ta;
332         GregorianCalendar now = (GregorianCalendar) GregorianCalendar.getInstance();
333
334         GregorianCalendar then = (GregorianCalendar) GregorianCalendar.getInstance();
335         //Make sure it's before measurements
336         then.set(1980, 1, 1);
337
338
339         //Extract latitude and longitude
340         line = readLine()[0].split(",");
341         latitude = Double.parseDouble(line[0]);
342         longitude = Double.parseDouble(line[1]);
343         //Calculate clear weather irradiation for a yaer
344         calcCWs();
345
346
347         line = readLine();
348
349         final int milliSPerDay = 1000 * 3600 * 24;
350         while (line != null) {
351             //Check for valid and compatible data
352             if ((line.length == 2 || line.length == 3) && !line[0].equals(EMPTY_LINE)) {
353
354
355                 dateValues = line[0].split("[^0-9]");
356                 year = Integer.parseInt(dateValues[0]);
357                 month = Integer.parseInt(dateValues[1]);
358                 day = Integer.parseInt(dateValues[2]);
359                 now.set(year, month - 1, day);
360                 //(0-364)
361                 dayOfYear = Math.min(now.get(GregorianCalendar.DAY_OF_YEAR) - 1, 364);
362
363                 long dt = now.getTimeInMillis() - then.getTimeInMillis();
364                 while (dt > milliSPerDay) {
365                     data.add(null);
366                     dt /= milliSPerDay;
367                 }
368                 then.set(year, month - 1, day);
369
370                 irr = Double.parseDouble(line[1]);
371                 //Check for invalid spikes and save the information about the replace
372                 if (irr > 0.8*extIrr_T[dayOfYear]) {
373                     irr = cwIrrTheory_T[dayOfYear];
374                     replaced++;
375                 }
376                 Ta = line.length == 3 ? Double.parseDouble(line[2]) : Double.NaN;
377
378                 data.add(new SData(dayOfYear, irr, Ta));
379             }
380             //Next dataline
381             line = readLine();
382         }
383
384     }
385
386     private void setCW(double[] cw) {
387         for (SData d : data) {
388             if (d != null) {
389                 d.setCWIrr(cw[d.getDayOfYear()]);
390             }
391         }
392     }
393
394     /**
395      * Prints processed data to file
396      * @throws java.io.IOException
397      */
```

```
398     void print(File directory) throws IOException {
399         File dir = new File(directory, "/theory/");
400         dir.mkdir();
401         printCWs(directory);
402         BufferedWriter out = null;
403         File file = null;
404         Formatter formatter = null;
405 
406         try {
407             int n = 1;
408             do {
409                 file = new File(dir, getName() + n++ + ".dat");
410             } while (file.exists());
411             if (!file.createNewFile()) {
412                 throw new IOException("Cant create file");
413             }
414             out = new BufferedWriter(new FileWriter(file));
415             formatter = new Formatter(out, Locale.ENGLISH);
416         } catch (FileNotFoundException ex) {
417             Logger.getLogger(SolisData.class.getName()).log(Level.SEVERE, null, ex);
418         }
419 
420         setCW(cwIrrTheory_T);
421 
422         printType(out, formatter);
423 
424         dir = new File(directory, "/Intr/");
425         dir.mkdir();
426         try {
427             int n = 1;
428             do {
429                 file = new File(dir, getName() + n++ + ".dat");
430             } while (file.exists());
431             if (!file.createNewFile()) {
432                 throw new IOException("Cant create file");
433             }
434             out = new BufferedWriter(new FileWriter(file));
435             formatter = new Formatter(out, Locale.ENGLISH);
436         } catch (Exception ex) {
437             Logger.getLogger(SolisData.class.getName()).log(Level.SEVERE, null, ex);
438         }
439         //New method
440         System.out.println(getName() + " Interpolated!");
441         setCW(cwIrrIntrp);
442         printType(out, formatter);
443 
444 
445     }
446 
447     private void printType(BufferedWriter out, Formatter formatter) throws IOException {
448         String format = "%1$4d";
449         out.append("#Data from " + getName() + "\n#\n");
450         out.append("  #N  #Day Of year   #Irradiation   #Clear weather irradiance"+
451                 "         #K_T,cs   #Good   #Ambient temperature #Broken\n");
452         int n = 0;
453 
454         for (SData d : data) {
455             if (d == null) {
456                 out.append("   " + n++ + "\t0\t0\t0\t100\t0\t0\t10\n\n");
457                 out.flush();
458             } else {
459                 formatter.format(format, n++);
460                 d.toString(formatter);
461                 formatter.flush();
462             }
463         }
464         formatter.close();
465         out.close();
466     }
467 }
```

```
1 package weatherperiodlengths;
2 
3 import java.io.EOFException;
4 import java.io.File;
5 import java.io.FileInputStream;
```

```
6  import java.io.FileNotFoundException;
7  import java.io.IOException;
8  import java.io.ObjectInputStream;
9  import java.text.DecimalFormat;
10 import java.util.ArrayList;
11 import java.util.Formatter;
12 import java.util.LinkedList;
13 import java.util.NoSuchElementException;
14 import java.util.Vector;
15
16 /**
17  *
18  * @author Espen Holtebu
19  */
20 public class WeatherPeriods {
21
22     private static DecimalFormat df = new DecimalFormat("0.00");
23     private SData[] days;
24     private LinkedList<LinkedList<Period>> periodOfPeriods;
25     private LinkedList<Period> periods = null;
26     private double avgCWI[];
27     private double avgMonthCwi[];
28     private String station;
29     private double latitude;
30     private double longitude;
31     private PartOfTheCountry countryPart;
32     private double cwiLimit;
33
34
35
36     WeatherPeriods(SolisData data, double cwiLimit) {
37         readSerializedData(data.getDatas());
38         this.station = data.getName();
39         this.latitude = data.getLatitude();
40         this.longitude = data.getLongitude();
41         countryPart = data.getPart();
42         this.cwiLimit = cwiLimit;
43         avgCWI = new double[25];
44         avgMonthCwi = new double[365];
45         calcPeriods();
46     }
47
48     public PartOfTheCountry getPartOfTheCountry(){
49         return countryPart;
50     }
51
52     public String getStation() {
53         return station;
54     }
55
56     public double getLatitude(){
57         return latitude;
58     }
59
60     public double getLongitude(){
61         return longitude;
62     }
63
64     public double getCwiLimit(){
65         return cwiLimit;
66     }
67
68     private void readSerializedData(ArrayList<SData> data) {
69         days = new SData[data.size()];
70         data.toArray(days);
71     }
72
73     private void calcPeriods(){
74         int periodLength = 1;
75         double avgCwi = 0.5;
76         boolean newPeriod = true;
77         boolean weather;
78         boolean prevWeather = false;
79         boolean weatherChange;
80         periodOfPeriods = new LinkedList<LinkedList<Period>>();
81         LinkedList<Period> period = new LinkedList<Period>();
82
```

```
83
84          System.out.println("\nAt " + station + " with cwi limit " + cwiLimit);
85          StringBuffer temp = new StringBuffer(2000);
86          for (SData sData : days) {
87              //Skip empty objects
88              if (sData == null) {
89                  periodOfPeriods.add(period);
90                  newPeriod = true;
91                  continue;
92              }
93
94              weather = sData.getCWI() > cwiLimit;
95              temp.append(sData.getCWI() + " ");
96              //check for new period and reinit data
97              if (newPeriod) {
98                  period = new LinkedList<Period>();
99                  periodLength = 1;
100                 newPeriod = false;
101                 prevWeather = weather;
102                 continue;
103             }
104
105             weatherChange = weather != prevWeather;
106             if(!(avgCwi >1.25)){
107                 avgCWI[(int)(avgCwi*20)]++;
108 //                  addMonthAvg(avgCwi);
109             }
110             if (weatherChange) {
111                 avgCwi /= periodLength;
112
113                 try {
114                     period.getLast().setNextLength(periodLength);
115                     period.getLast().setNextAvgCwi(avgCwi);
116                 } catch (NoSuchElementException e) {
117                     //System.out.println("Ny periode " + station);
118                 }
119                 period.add(new Period(weather, periodLength, cwiLimit, avgCwi, sData.
                        getDayOfYear()));
120                 periodLength = 1;
121                 avgCwi = sData.getCWI();
122             } else {
123                 periodLength++;
124             }
125             prevWeather = weather;
126         }
127     }
128
129
130
131     public LinkedList<LinkedList<Period>> getPeriods() {
132         return periodOfPeriods;
133     }
134
135     public double[] getCWIS(){
136         return this.avgCWI;
137     }
138 }
```

C.4 Innstrålingsdata

```
1  /*
2   * To change this template, choose Tools | Templates
3   * and open the template in the editor.
4   */
5
6  package radiationtransformation;
7
8  import java.io.File;
9  import java.io.FileNotFoundException;
10 import java.util.Date;
11 import java.util.Hashtable;
```

```
12
13 /**
14  *
15  * @author Administrator
16  */
17 public class Main {
18     public static void main(String[] args) throws FileNotFoundException {
19         //Import pyranometer data
20         Hashtable<Date,Double > kdata = (new PyraData(new File("C:/data/11juni/11062009.TXT"),
               32, 18, 59.89, 10.71, 1)).getKData();
21         //Import pygeometer and temperature data
22         Hashtable<Date,double[]> edata = (new PyrgData(new File("C:/PyrgData.csv"),10.71, 1)).
               getEpsilonSkyValues();
23         //Merge data
24         Calculate result = new Calculate(kdata,edata);
25         //Calculate and print data to file for plotting
26         result.print();
27     }
28 }
```

```
1
2 package radiationtransformation;
3
4
5 import java.io.BufferedReader;
6 import java.io.File;
7 import java.io.FileNotFoundException;
8 import java.io.FileReader;
9 import java.io.IOException;
10 import java.util.Scanner;
11 import java.util.logging.Level;
12 import java.util.logging.Logger;
13
14 /**
15  *
16  * @author Administrator
17  */
18 public class Data {
19     public static final String EMPTY_LINE = "NaN";
20
21
22
23
24
25     private String name;
26
27     Scanner sc;
28
29
30     public Data(File file){
31         name = file.getName();
32         try {
33             sc = new Scanner(new BufferedReader(new FileReader(file)));
34         } catch (FileNotFoundException ex) {
35             Logger.getLogger(Data.class.getName()).log(Level.SEVERE, null, ex);
36         } catch (IOException ex) {
37             Logger.getLogger(Data.class.getName()).log(Level.SEVERE, null, ex);
38         }
39     }
40
41
42     public String getName(){
43         return name;
44     }
45
46
47     public String[] readLine(String regex){
48         if (sc == null){
49             return null;
50         }
51         if(!sc.hasNext()){
52             sc.close();
53             sc = null;
54             return null;
55         }
56         String line[] = sc.nextLine().split(regex);
```

```
57          //check for empty lines
58          if(line == null) {
59              return new String[]{EMPTY_LINE};
60          }
61
62          //Skip commented lines
63          while(line[0].startsWith("#")){
64               line = sc.nextLine().split(regex);
65          }
66
67          return line;
68      }
69
70
71  }
```

```
1   /*
2    * To change this template, choose Tools | Templates
3    * and open the template in the editor.
4    */
5
6   package radiationtransformation;
7
8   import java.io.File;
9   import java.util.Date;
10  import java.util.GregorianCalendar;
11  import java.util.Hashtable;
12  import sollib.MathSolar;
13  import static java.lang.Integer.parseInt;
14  import static java.lang.Double.parseDouble;
15
16  /**
17   *
18   * @author Espen Holtebu
19   */
20  public class PyraData extends Data {
21
22
23      private final double tiltAngle;
24      private final double azimuthAngle;
25      private final double latitude;
26      private final double longitude;
27      private final int timeZone;
28      private Hashtable<Date,Double> kdata;
29
30      public PyraData(File file, double tiltAngle, double azimuthAngle, double latitude, double
            longitude, int timeZone){
31          super(file);
32          this.tiltAngle = tiltAngle;
33          this.azimuthAngle = azimuthAngle;
34          this.latitude = latitude;
35          this.longitude = longitude;
36          this.timeZone = timeZone;
37          kdata = new Hashtable<Date, Double>();
38          readData();
39      }
40
41      public Hashtable<Date,Double> getKData(){
42
43          return kdata;
44      }
45
46      private void readData() {
47          String regex = ";";
48          GregorianCalendar now = (GregorianCalendar) GregorianCalendar.getInstance();
49          String[] line = readLine(regex);
50          while(line != null){
51              if (line[0].equals(EMPTY_LINE)) continue;
52
53              String[] day = line[0].substring(1, line[0].length()-1).split("\\.");
54
55              String[] time = line[1].substring(1, line[1].length()-1).split(":");
56
57              //Set to UTC +1
58              now.set(2000+parseInt(day[2]), parseInt(day[1])-1, parseInt(day[0]), parseInt(time
                    [0]), parseInt(time[1]), parseInt(time[2]));
```

```
59              //and account for datalogger time dilation
60              now.add(GregorianCalendar.HOUR_OF_DAY, -1);
61              now.add(GregorianCalendar.MINUTE, -38);
62              Date then = now.getTime();
63              //Make now solar time
64              MathSolar.solarTime(now, longitude, timeZone);
65
66              double irr = parseDouble(line[2])/0.174d;
67              double extRad = extRad(now);
68              double k = extRad > 0 ? irr/extRad : 0;
69
70              if(k>0) kdata.put(now.getTime(),k);
71
72              line = readLine(regex);
73          }
74      }
75
76      private double extRad(GregorianCalendar now) {
77          double hourOfDay =
78                  now.get(GregorianCalendar.HOUR_OF_DAY) +
79                  now.get(GregorianCalendar.MINUTE) / 60 +
80                  now.get(GregorianCalendar.SECOND) / 3600;
81          int dayOfYear =
82                  now.get(GregorianCalendar.DAY_OF_YEAR)-1;
83          double declination = MathSolar.getDeclination(dayOfYear);
84          double hourAngle = MathSolar.getHourAngle(hourOfDay);
85          double cosAOI = MathSolar.cosAngleOfIncidence(latitude, tiltAngle, azimuthAngle,
                declination, hourAngle);
86          if (Math.acos(cosAOI) > Math.PI / 3) {
87              return 0;
88          }
89          return MathSolar.extraterrestrialRadiation(dayOfYear) * cosAOI;
90      }
91
92  }
```

```
1   /*
2    * To change this template, choose Tools | Templates
3    * and open the template in the editor.
4    */
5   package radiationtransformation;
6
7   import java.io.File;
8   import java.util.Date;
9   import java.util.GregorianCalendar;
10  import java.util.Hashtable;
11  import sollib.MathSolar;
12  import static java.lang.Integer.parseInt;
13  import static java.lang.Double.parseDouble;
14
15  /**
16   *
17   * @author Administrator
18   */
19  public class PyrgData extends Data {
20
21      /** Stefan boltzmann's coanstant*/
22      private static final double K_SB = 5.6704e-8;
23      Hashtable<Date, double[]> newdata;
24      private final int timeZone;
25      private final double longitude;
26
27      /**Class to import pyrgeometer data*/
28      public PyrgData(File file, double longitude, int timeZone) {
29          super(file);
30          newdata = new Hashtable<Date, double[]>();
31          this.longitude = longitude;
32          this.timeZone = timeZone;
33          readData();
34      }
35
36      public Hashtable<Date, double[]> getEpsilonSkyValues() {
37          return newdata;
38      }
39
40      private void readData() {
```

```
41          Data taData = new Data(new File("C:/data/11juni/TdTa.dat"));
42
43          double Ta, Ta0 = 0, TaNext = 0, dTa7dt = 0;
44          double Td, Td0 = 0, TdNext = 0, dTd7dt = 0;
45          long dt = 0;
46          String regex = ";";
47          GregorianCalendar now = (GregorianCalendar) GregorianCalendar.getInstance();
48          GregorianCalendar nowST = null;
49          GregorianCalendar nowTa = (GregorianCalendar) GregorianCalendar.getInstance();
50          GregorianCalendar nowTaNext = (GregorianCalendar) GregorianCalendar.getInstance();
51
52          String[] line = readLine(regex);
53          String[] lineTa = taData.readLine(regex);
54          String[] lineTaNext = taData.readLine(regex);
55          TaNext = parseDouble(lineTaNext[5]) + 273.15;
56          TdNext = parseDouble(lineTaNext[6]);
57
58
59          //Init to the same time
60          nowTa.set(parseInt(lineTa[1]), parseInt(lineTa[2]) - 1, parseInt(lineTa[3]), parseInt(
                lineTa[4]), 0, 0);
61          nowTaNext.set(parseInt(lineTa[1]), parseInt(lineTa[2]) - 1, parseInt(lineTa[3]),
                parseInt(lineTa[4]), 0, 0);
62
63
64          while (line != null) {
65              while (line[0].equals(EMPTY_LINE)) {
66                  line = readLine(regex);
67              }
68
69              //Set to UTC +1 (its in daylight saving)
70              now.set(parseInt(line[0]), parseInt(line[1]) - 1, parseInt(line[2]), parseInt(line
                    [3]) - 1, parseInt(line[4]), parseInt(line[5]));
71
72              //Fast forward to next Ta-value
73              if (now.before(nowTa)) {
74                  line = readLine(regex);
75                  continue;
76              }
77
78              //Check if next ta is to be used
79              if (now.after(nowTaNext) || now.equals(nowTaNext)) {
80                  lineTa = lineTaNext;
81                  nowTa.setTime(nowTaNext.getTime());
82                  lineTaNext = taData.readLine(regex); if(lineTaNext == null) break;
83                  nowTaNext.set(parseInt(lineTaNext[1]), parseInt(lineTaNext[2]) - 1, parseInt(
                        lineTaNext[3]), parseInt(lineTaNext[4]), 0, 0);
84                  Ta0 = TaNext;
85                  Td0 = TdNext;
86                  TaNext = parseDouble(lineTaNext[5]) + 273.15;
87                  TdNext = parseDouble(lineTaNext[6]);
88                  dt = (nowTaNext.getTimeInMillis() - nowTa.getTimeInMillis()) / 1000;
89                  dTa7dt = (TaNext - Ta0) / dt;
90                  dTd7dt = (TdNext - Td0) / dt;
91              }
92              dt = (now.getTimeInMillis() - nowTa.getTimeInMillis()) / 1000;
93
94              //Make solar time
95              nowST = MathSolar.solarTimeClone(now, longitude, timeZone);
96
97              Ta = Ta0 + dTa7dt * dt;
98              Td = Td0 + dTd7dt * dt;
99
100             double hourOfDay =
101                     now.get(GregorianCalendar.HOUR_OF_DAY) +
102                     now.get(GregorianCalendar.MINUTE) / 60 +
103                     now.get(GregorianCalendar.SECOND) / 3600;
104             double atmIrr = parseDouble(line[13]);
105
106             newdata.put(nowST.getTime(), new double[]{0,atmIrr, Ta, Td,nowST.get(
                    GregorianCalendar.DAY_OF_YEAR) ,hourOfDay});
107
108             line = readLine(regex);
109
110         }
111     }
112 }
```

```
1  /*
2   * To change this template, choose Tools | Templates
3   * and open the template in the editor.
4   */
5  package radiationtransformation;
6
7  import java.io.BufferedOutputStream;
8  import java.io.FileNotFoundException;
9  import java.io.FileOutputStream;
10 import java.util.Collections;
11 import java.util.Date;
12 import java.util.Formatter;
13 import java.util.GregorianCalendar;
14 import java.util.Hashtable;
15 import java.util.Iterator;
16 import java.util.Locale;
17 import java.util.Vector;
18 import static sollib.MathSolar.*;
19
20 /**
21  * Class for merging and calculation of epsilon sky values
22  * @author Espen holtebu
23  */
24 class Calculate {
25     /**Stefan -Boltzmann's constant*/
26     private static final double K_SB = 5.6704e-8;
27     private final Hashtable<Date, Double> kdata;
28     private final Hashtable<Date, double[]> edata;
29     private Hashtable<Date, double[]> data;
30     double[] k;
31     double[] atmIrr;
32     double[] Ta;
33     double[] Td;
34     double[] eps;
35     double[] eps0;
36     double[] calcIrr;
37
38
39     public Calculate(
40             Hashtable<Date, Double> kdata,
41             Hashtable<Date, double[]> edata) {
42         this.kdata = kdata;
43         this.edata = edata;
44         this.data = new Hashtable<Date, double[]>();
45         sortnMerge();
46     }
47
48     private void sortnMerge() {
49         Vector vk = new Vector(kdata.keySet());
50         Collections.sort(vk);
51         Iterator itk = vk.iterator();
52         Vector ve = new Vector(edata.keySet());
53         Collections.sort(ve);
54         Iterator ite = ve.iterator();
55
56         Date nowK, nowE = (Date) ite.next();
57         outerloop:
58         while (itk.hasNext()) {
59             nowK = (Date) itk.next();
60
61             while (nowK.after(nowE)) {
62                 if (!ite.hasNext()) {
63                     break outerloop;
64                 }
65                 //System.out.println(nowE + "....." + nowK);
66                 nowE = (Date) ite.next();
67             }
68             double[] dat = edata.get(nowE);
69             dat[0] = kdata.get(nowK);
70             data.put(nowK, dat);
71             System.out.println(nowE + "....." + nowK);
72         }
73
74     }
75
```

```
76      @SuppressWarnings("empty-statement")
77      void print() throws FileNotFoundException {
78          Formatter out = null;
79
80          out = new Formatter(
81                  new BufferedOutputStream(
82                  new FileOutputStream("c:/kdep.dat")));
83          Vector v = new Vector(data.keySet());
84          Collections.sort(v);
85          Iterator it = v.iterator();
86
87
88          k = new double[v.size()];
89          double[] kavg = new double[v.size()];
90          atmIrr = new double[v.size()];
91          Ta = new double[v.size()];
92          Td = new double[v.size()];
93          eps = new double[v.size()];
94          eps0 = new double[v.size()];
95          calcIrr = new double[v.size()];
96          boolean[] newday = new boolean[v.size()];
97
98          int i = 0;
99          int epsRemoved = 0;
100         Date before = (Date) it.next(),now = before;
101         GregorianCalendar cal =
102                 (GregorianCalendar) GregorianCalendar.getInstance();
103         do{
104             //{k,atmIrr, Ta, Td, dayOfYear, hourOfDay, eps0_emp}
105             double[] value = data.get(before);
106             cal.setTime(before);
107             k[i] = value[0];
108             //Adjust for irradiation
109             atmIrr[i] = Math.max(
110                     value[1]-25*(1.367*k[i]*cosZenithAngle(
111                     59.89, value[4],value[5])),
112                     0);
113             Ta[i] = value[2];
114             Td[i] = value[3];
115             eps[i] = atmIrr[i] / K_SB / Ta[i] / Ta[i] / Ta[i] / Ta[i];
116             if(eps[i] > 1.5 || eps[i]<0.5){
117                 System.out.println("Urealistisk verdi fjernet: " + eps[i]);
118                 epsRemoved++;
119                 eps[i] = 0;
120             }
121             eps0[i] =
122                     0.711 +
123                     Td[i] * (0.0056 + 0.000073 * Td[i]) +
124                     0.013 * Math.cos(2 * Math.PI * value[5] / 24);;
125             before = now;
126             now = (Date) it.next();
127             newday[i++] = (now.getTime() - before.getTime())/60000 > 15;
128         }while (it.hasNext());
129         newday[newday.length-1] = true;
130
131
132
133
134         double[][][] avgEps;
135         double[][][] varEps;
136         int    [][][] avgEpsCnt;
137
138         int idx = 0;
139         for (int plusminus = 15; plusminus >= 0; plusminus -= 3) {
140             avgEps    = new double[6][17][40];
141             varEps  = new double[6][17][40];
142             avgEpsCnt = new    int[6][17][40];
143             int startDayI = 0;
144             int endDayI = -1;
145             out             = new Formatter(
146                     new BufferedOutputStream(
147                     new FileOutputStream(
148                     "c:/shared/data/eps/" +  plusminus +     ".dat")));
149             Formatter out2 = new Formatter(
150                     new BufferedOutputStream(
151                     new FileOutputStream(
152                     "c:/shared/data/eps/" + plusminus + "_grid.dat")));
```

```
153             for (i = 1; i < k.length; i++) {
154
155                 if (i > endDayI) {
156                     startDayI = endDayI + 1;
157                     int j;
158                     for (j = Math.min(startDayI + 1,k.length-1); !newday[j] && j < k.length-2; j
                            ++);
159                     endDayI = j-1;
160                 }
161
162
163                 //Less than 15 mins
164                 if (!newday[i] && atmIrr[i] !=atmIrr[Math.max(0, i-1)]) {
165                     if(eps[i] == 0) continue;
166                     kavg[i] = kAvg(
167                             Math.max(startDayI,i-plusminus),
168                             Math.min(endDayI,i+plusminus));
169                     out.format(Locale.ENGLISH, "%1$15f", kavg[i]);
170                     out.format(Locale.ENGLISH, "%1$15f", atmIrr[i]);
171                     out.format(Locale.ENGLISH, "%1$15f", eps[i]);
172                     out.format(Locale.ENGLISH, "%1$15f\n", eps0[i]);
173                     int kidx = (int)Math.max(0,Math.min(kavg[i]*50, 39));
174                     int epsidx = (int)Math.min(Math.max(((eps0[i]-0.65)*100),0),16);
175                     avgEps[idx][epsidx][kidx] += eps[i];
176                     avgEpsCnt[idx][epsidx][kidx]++;
177                 } else{
178                     eps[i] = 0;
179                 }
180             }
181
182
183             //Find average atmospheric radiation
184             for (int j = 0; j < avgEps[idx].length; j++) {
185                 for (int l = 0; l < avgEps[idx][j].length; l++) {
186                     avgEps[idx][j][l] = avgEpsCnt[idx][j][l]==0 ? 0 :
187                         avgEps[idx][j][l]/avgEpsCnt[idx][j][l];
188                 }
189             }
190
191             //Find the standard deviation
192             for (i = 1; i < k.length; i++) {
193                 if(eps[i] != 0){
194                     int kidx = (int)Math.max(0,Math.min(kavg[i]*50, 39));
195                     int epsidx = (int)Math.min(Math.max(((eps0[i]-0.65)*100),0),16);
196                     double dev = (eps[i] -avgEps[idx][epsidx][kidx]);
197                     varEps[idx][epsidx][kidx] += dev*dev/avgEpsCnt[idx][epsidx][kidx];
198                 }
199             }
200
201             out2.format("#antall Epsilon fjernet: "+ epsRemoved);
202             out2.format("\n#k    eps_0    eps_avg    stdev_eps    antall\n");
203             for (int eps_0 = 0; eps_0 < avgEps[idx].length; eps_0++) {
204                 for (int clearness = 0; clearness < avgEps[idx][eps_0].length; clearness++) {
205                     if(avgEpsCnt[idx][eps_0][clearness] > 10){
206                     out2.format(Locale.ENGLISH, "%1$15f", (clearness+1d)/50);
207                     out2.format(Locale.ENGLISH, "%1$15f", eps_0/100d +0.65);
208                     out2.format(Locale.ENGLISH, "%1$15f",       avgEps[idx][eps_0][clearness]);
209                     out2.format(Locale.ENGLISH, "%1$15f",     Math.sqrt(varEps[idx][eps_0][
                            clearness]));
210                     out2.format(Locale.ENGLISH, "%1$15d\n", avgEpsCnt[idx][eps_0][clearness]);
211                     }
212                 }
213                 out2.format("\n\n");
214                 out2.flush();
215             }
216
217             idx++;
218             out.close();
219             out2.close();
220         }
221
222
223
224
225
226
227     }
```

```
228
229     /**Find various average values of the clearness index*/
230     private double kAvg(int start, int end) {
231         double avg = 0;
232         for (int i = start; i <= end; i++) {
233             avg += k[i];
234         }
235         return avg / (end - start + 1);
236     }
237 }
```

C.5 MATLAB-program for beregning av GP-parametere

```
function [c,l0] = reg(fil)
%REG Summary of this function goes here
%   Detailed explanation goes here
data = importdata(fil,' ',4);

c = 0.1706;
l0= 1.313;

semilogx(data.data(:,1),data.data(:,2));
hold all;
semilogx(data.data(:,1),data.data(:,3));
op = fitoptions('Method','NonlinearLeastSquares','Lower', [0,0],'Upper',[1,5],'StartPoint',[0.2,1.0]);%,'Display','final'
pareto = fittype('log((1/c)*log(1+(l*c)/l0))','independent','l','options',op);
plot(data.data(:,1),data.data(:,2));
[yo1,yo2]=fit(data.data(:,1),data.data(:,2),pareto)
l=1:1:100;
semilogx(l,log(-log((1+(l*c)/l0).^(-1/c))));
hold off;
end
```