



Modeller och verklighet för datateknik

VT 2024

Arbetsuppgift 3:

Solen som energikälla

Grupp 5

Namn: Noor Baradee

Namn: Omar Al Ayoubi

Namn: Julia Tadic

Namn: Mohammed Hajjo

Datum för inlämning: 2024/03/11

Innehållsförteckning

Modeller och verklighet för datateknik VT 2024.....	1
Innehållsförteckning.....	2
1. Inledning.....	3
1.1 Koppling till framtida tillämpningar.....	3
Teoretisk bakgrund.....	4
Resultat.....	12
Diskussion.....	17
Referenser.....	20
Appendix A: Python-kod.....	21

1. Inledning

Förnyelsebara energikällor är något man får höra talas om väldigt ofta världen över, inte minst på en lokal nivå. Detta har en väldigt bra grund till sig. Vår jord hettas upp som en effekt av den fossila förbränningen vårt samhälle förlitar sig på. Även om vissa menar att detta är helt av naturliga skäl och att detta fenomen alltid varit en del av jordens historia så menar andra att man inte kan ignorera den takt denna uppvärmning sker i. Det har enligt ett flertal vetenskapliga studier visat att just de utsläpp till förbränningen av fossila bränslen har en egenskap som gör att de absorberar jordens utgående infraröda strålning och värmes upp vilket i sin tur gör att jorden värmes upp. Man menar att detta inte är en långsiktig lösning och att förändring av denna process är viktig för att våra barn och barnbarn ska uppleva jorden på samma sätt som vi har gjort. Man vill i så stor grad som möjligt förhindra dessa utsläpp genom att implementera energikällor som inte präglas av dessa utsläpp och som inte är ändliga på samma sätt som fossila bränslen, därav namnet förnyelsebara energikällor. En väldigt populär och starkt växande teknik för detta är implementering av solpaneler. Starkt växande i den mening att verkningsgraden för dessa paneler blivit högre med och att tillverkningskostnaden blivit lägre med forskningen gång.

AU3 omfattar två större delmoment, insamling av information och teoretisk bakgrund för solpaneler, samt modellering av solpaneler och dess effekt under olika förhållanden för att få en uppskattning för hur man kan förvänta sig att en solpanel beter sig i olika städer runtom i Sverige. Och på så sätt kunna avgöra vilken vinkel i väderiktning samt vilken vinkel i altitud en panel ska ha.

Arbetet är utfört enligt ett kompendium, Problembeskrivning [1] vilket beskriver de olika förhållanden vi ska anta, vilka platser och vilka dagar/tidsintervall vi ska utföra denna simulering för. Kompletterande information för beräkning och teori som presenteras vidare i rapporten är till en betydande del inlärda med hjälp av boken Physics [2] och föreläsningsmaterial [3].

1.1 Koppling till framtida tillämpningar

De färdigheter som vi utvecklar under hela arbetsgången kan förväntas vara behändiga i framtida arbeten då datavetenskap till stor del utnyttjas vid beräkningar med hjälp av modellering och simulering. I många fall tar de hänsyn till verkliga förhållanden som en

process, material eller kropp osv. Allt arbete vars syfte är att främja elevers problemlösningsmetodik, innovationskapacitet samt kritiskt applicering av logik är extremt nyttiga och väldigt uppskattade av elever vars kunskap och färdigheter får ett tillämpat syfte i mer eller mindre abstrakta och övnings vänliga miljöer.

Teoretisk bakgrund

Under de senaste två decennierna har solenergins bidrag till världens totala energiförsörjning ökat avsevärt. Denna del av rapporten kommer att visa hur solceller producerar el.

Energi från solen är den mest förekommande och absolut fritt tillgängliga energin på vår planet. För att kunna använda denna energi behöver vi hjälp från det näst vanligaste grundämnet på jorden, sand. Sanden måste omvandlas till 99,999 % rena kiselkristaller för att användas i solceller. För att uppnå detta måste sanden genomgå en komplex reningsprocess där sand och kol smälts tillsammans för att skapa rå kisel efteråt med hjälp av andra processer som gör att kislet omvandlas till en gasformig kiselförening. Detta blandas sedan med väte för att få högt renat polykristallint kisel. Kislet omformas till mycket tunna skivor som kallas kiselskivor. Kisel skivorna är hjärtat i en solcell.

När solljuset träffar elektronerna i kiselatomen tillåter tillföring av energin från fotonerna, elektronen att bryta sig loss från elektronbanan och röra sig oberoende av kiselatomen.

Men denna rörelse av elektronerna är slumpmässig. Det resulterar inte i någon strömutgång. För att göra elektronflödet enkelriktat behövs en drivkraft. Ett enkelt och praktiskt sätt att producera drivkraften är en PN-övergång.

En PN-övergång är en yta mellan två halvledarmaterial. Dessa två material är dopade med olika atomer för att få olika elektriska egenskaper.

Sidan n har överflöd av elektroner tack vare materialet som den är dopad med. Vanligtvis är det ett material som har fler valenselektroner än kisel. Fosfor har en extra valenselektron än halvledarmaterialet kisel. Fosfor atomerna ersätter kisel atomerna och kisel får då ett överskott på elektroner.

Sidan p är dopad med material som leder till brist på elektroner. Vanligtvis är det bor för att den har färre valenselektroner än kisel. Här ersätter bor atomerna kisel atomerna och kisel får då underskott av atomer och blir positivt laddad istället.

Då dessa två ytor träffar varandra, diffunderar elektronerna från n-ytan till p-ytan för att fylla ut hålen. Med dessa hål menar man brist på elektroner, alltså befinner sig dessa hål i p-ytan där det finns plats för elektroner.

Solpanelen har olika lager. En av dem är ett lager av celler. Dessa solceller är sammankopplade och elektronerna samlas i samlingskkenor.

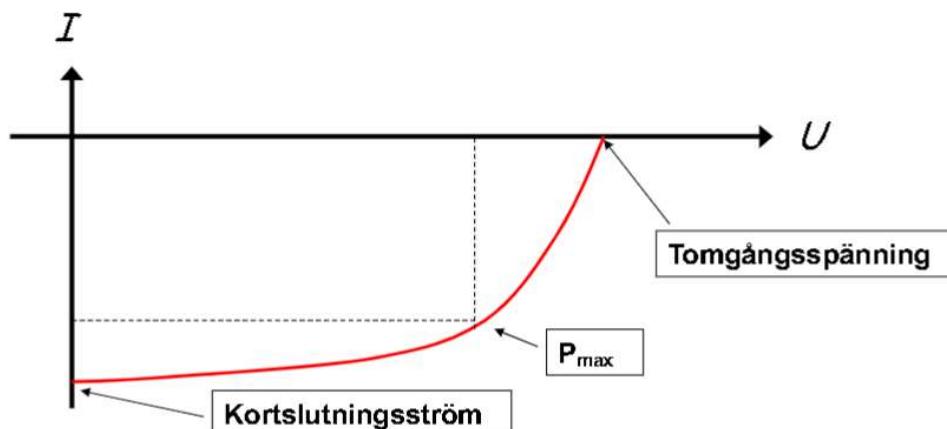
Den övre negativa sidan av den ena cellen är ansluten till baksidan av nästa cell genom kopparremsor och bildar en seriekoppling. När man kopplar dessa seriekopplade celler parallellt med en annan cellserie får man solpanelen.

En enda solcell producerar bara cirka 0,5 spänning. Kombinationen av serie- och parallellkoppling av cellerna ökar ström- och spänningsvärdena till ett användbart område. Lagret av EVA-plåt på båda sidor av cellerna är för att skydda dem från stötar, vibrationer, fukt och smuts.

Karakteristiken hos en solcell visas normalt som en kurva där strömmen plottas mot spänningen. När det gäller att bedöma den mest extrema produktiviteten hos solcellen, är det två faktorer som spelar roll. Materialets egenskaper och designen. Materialens egenskaper utser förmågan att absorbera solljus och den elektriska ledningsförmågan. Dessutom påverkas effektiviteten av solcellens design, inklusive designen av PN-övergången.

Verkningsgrad hos en solcell är andelen av den verkliga elektriska kraften som produceras av den solcellen till den maximala effekten som kan skapas från hur mycket ljus som solcellen utsätts för. Det är en andel av hur produktivt solcellen kan omvandla solenergi till elektrisk energi. Detta handlar alltså om hur mycket energi vi kan få utav en solcell.

Effektivitet tar en betydande del i användbarheten av solcellen. Ju högre produktivitet än solcell har, desto mer elektrisk energi den kan leverera från den tillgängliga solenergin.



Figur 1: Solcellens karakteristik (Jörgens föreläsningsbilder [3])

I Bild 1 är P_{max} den maximal effekt vi kan få ut från solcellen. Med hjälp av denna punkt kan man bestämma solcellen effektivitet/ verkningsgrad genom att analysera förhållandet mellan den maximala effekten och solintensiteten, alltså mellan hur mycket energi vi kan få ut från solcellen och hur mycket den får in. I en dålig solcell är kurvan mer rak mellan ström och spänning och P_{max} blir mindre på grund av följande funktion:

$$P = I * U \quad (1)$$

För att beräkna solens position på himlen vid en viss tidpunkt tar vi hänsyn till några specifika vinklar. Deklinationvinkeln bestämmer solen vertikala position på himlen med hjälp av ekvationen

$$\delta_s = -23.44 * \cos\left(\frac{365}{360} * n\right) \quad (2)$$

Där n avser dagen på året. Timvinkeln ansvarar för att bestämma solens horisontella positionen med hjälp av ekvationen

$$\Omega = 15 * t - 180 \quad (3)$$

Där t är tiden på dagen. Med hjälp av dessa värden kan vi bestämma solens altitud som är solens höjd över horisonten med

$$\theta_s = \arcsin[\sin(\phi) \times \sin(\delta_s) + \cos(\phi) \times \cos(\delta_s) \times \cos(\Omega)] \quad (4)$$

Där ϕ är latitud är en vinkel som bestämmer en plats på jordytan i förhållande till ekvatorn där själva vinkeln sträcker sig mellan 0° och 90° och kan bestämmas genom att mäta vinkeln mellan en plats på jordytan och ekvatorn.

Azimutvinkeln kan också bestämmas med hjälp av ekvationen

$$\alpha_s = 180 + \arccos\left[-\frac{\sin(\phi) \times \sin(\theta_s) - \sin(\delta_s)}{\cos(\phi) \times \cos(\theta_s)}\right] \quad (5)$$

Där α_s är det horisontella avståndet från norr. Då det gäller instrålning eller intensiteten från solen som träffar jorden så representeras det av ekvationen

$$I = 1.1 * 1360 * 0.7 * \left(\frac{1}{\sin(\theta_s)}\right)^{0.678}. \quad (6)$$

Med detta gjort, kan intensiteten mot solpanelen beräknas. Detta är alltså mängden solenergi som träffar solpanelen. Denna intensitet kan beräknas enligt

$$I_p = I * [\cos(\theta_p - \theta_s) * \cos(\alpha_p - \alpha_s) + (1 - \cos(\alpha_p - \alpha_s)) * \sin(\theta_p) * \sin(\theta_s)] \quad (7)$$

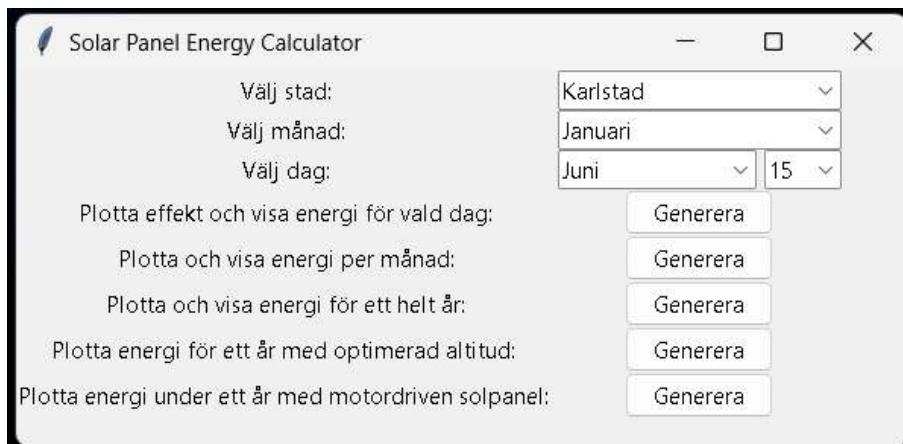
Solpanelens elektriska effekt beräknas med hjälp ut av intensiteten mot panelen samt verkningsgraden ϵ och Arena A för panelen.

$$P = \epsilon * I_p * A \quad (8)$$

Utförande

Grunden av programmet byggdes upp av metoder för beräkningar av solen och solpanelens egenskaper såsom vinklar och intensitet, formlerna som användes var givna i problembeskrivningen för arbetsuppgift 3 [1]. Programmet är konstruerat för att göra beräkningar på en solpanel med en verkningsgrad $\Sigma = 0,15$ och en area på $50m^2$. I programmet som utvecklades i Spyder användes biblioteket numpy för att kunna använda matematiska operationer, funktionerna i numpy arbetar endast med radianer vilket blev en större anpassning att ta hänsyn till för användning av de givna formlerna i problembeskrivningen[1].

Ett användargränssnitt skapades för att enkelt kunna ange vilken stad solpanelen ska befina sig i, vilken dag på året användaren vill plotta effekten för eller vilken månad energin ska beräknas för. När användaren väljer att generera energi för ett år eller en månad visas plottar för både effekten med och utan hänsyn till stadens soltimmar.



Figur 2: Användargränssnitt för programmet (gruppens applikation under köring)

För att programmet skulle kunna plotta en acceptabel figur för vårt ändamål användes en array med index för att erhålla effekten för var 15 minuter under ett dygn. Denna funktion, calc_effect_24_hours(), används av programmet för att lösa alla deluppgifter i AU3.

```
#Retunerar en array med 49 index som erhåller effekten för varje halvtimme under en dag
def calc_effect_24_hours():
    array = [0] * 97
    for t in np.arange(0, 24, 0.25):
        P = effekt * get_panelintensitet(get_solaltitud(t), get_solazimut(get_timvinkel(t), get_solaltitud(t), get_deklinationsvinkel(dag))) * area
        if P < 0:
            P = 0
        array[int(t * 4)] = P
    return array
```

Figur 3: Funktion i programmet för att få effekt och energi under 24 timmar (programkod)

Deluppgift 1 krävde att programmet skulle kunna plotta effekten i Watt som funktion av tiden för ett helt dygn utan hänsyn till antalet soltimmar. Programmet uppnår kravet genom att plotta arrayen från calc_effect_24_hours(), vilket har kedje anrop till alla grundfunktioner som utgör de matematiska

ekvationerna för solen och panelens egenskaper. Programmets användargränssnitt tillåter användaren att plotta effekten under alla årets dagar.

För att kunna beräkna energin för olika månader på året utnyttjades funktionen calc_effect_24_hours() då integralen av kurvan ger oss energin för dagen. Genom att addera energin från relevanta dagar kunde den totala energin för en månad beräknas, detta kräver att vi använder rätt dagar för den månad som matades in i användargränssnittet. En rad if-satser efter varandra kunde ge oss rätt dagsintervall under året, se figur 4. Att beräkna energin för ett helt år blev en lättare implementering då vi inte behövde ta hänsyn till intervaller utan adderade energin för alla dagar under året.

```
#Retunerar en range som tar hänsyn till årets dagar baserat på vald månad
def get_day_range_for_month(mnad):
    range = np.arange(1,31,1)
    if mnad == "Januari":
        range = np.arange(1, 32, 1)
    if mnad == "Februari":
        range = np.arange(32, 60, 1)
    if mnad == "Mars":
        range = np.arange(60, 91, 1)
    if mnad == "April":
        range = np.arange(91, 121, 1)
    if mnad == "Maj":
        range = np.arange(121, 152, 1)
    if mnad == "Juni":
        range = np.arange(152, 182, 1)
    if mnad == "Juli":
        range = np.arange(182, 213, 1)
    if mnad == "Augusti":
        range = np.arange(213, 244, 1)
    if mnad == "September":
        range = np.arange(244, 274, 1)
    if mnad == "Oktober":
        range = np.arange(274, 305, 1)
    if mnad == "November":
        range = np.arange(305, 335, 1)
    if mnad == "December":
        range = np.arange(335, 366, 1)
    return range
```

Figur 4: Funktion i programmet som returnerar motsvarande intervall för given månad
(programkod)

För att göra dessa beräkningar med hänsyn till soltimmarna för solpanelens placeringsort användes sannolikheten för soligt eller molnigt väder. Problembeskrivningens[1] definiering av en soltimme, det vill säga en timme då solintensiteten är större än $120W/m^2$, användes för att anta dagens soltimmar, se kodsnutt i figur 5. Därefter hämtades de uppmätta soltimmarna för den relevanta staden ur en array som motsvarar tabell 2 i problembeskrivningen[1], för att sedan divideras med de beräknade soltimmarna. Sannolikheten multipliceras därefter med energin som beräknats för dagen.

```
#Retunerar tillgängliga soltimmar per månad enligt Jörgens soltimmarnas definition
def get_sun_hours(range):
    global dag
    hours = 0
    for i in range:
        dag = i;
        for t in np.arange(0,24, 1):
            I = get_solintensitet(get_solaltitud(t))
            if I > 120:
                hours += 1
    return hours
```

Figur 5: Kodsmutt av funktion för beräkning av antal soltimmar per dag (programkod)

För att beräkna den optimala altituden för solpanelen under ett helt år, utgick utvecklarna från kurvan för energin under ett år under är jämn, det vill säga när kurvan har nått sin lokala maximipunkt kommer kurvan att avta. För att underlätta beräkningen för processorn jämfördes först energin för ett helt år med var tio grads steg skillnad, när energin började avta jämfördes varje heltals vinkel -9 och +9 grader räknat från det förra toppvärdet.

```
#Optimerar solpanelens altitud och retunerar vinkel
def optimize_teta_p():
    global altitud_panel, dag
    effekt = 0
    #Testar var tionde vinkel
    for p in np.arange(1,90,10):
        altitud_panel = np.deg2rad(p)
        ny_effekt = 0
        #Energi per år för test vinkeln
        for i in np.arange(1, 366, 10):
            dag = i
            ny_effekt += get_kWh(calc_effect_24_hours())
        #Om energin börjar avta hoppa in i en ny lop
        if ny_effekt < effekt:
            #Testa alla vinklar 9 steg före och efter nuvarande vinkel
            effekt = 0
            for j in np.arange(p-9,p+9,1):
                altitud_panel = np.deg2rad(j)
                ny_effekt = 0
                #Energi per år för test vinkeln
                for k in np.arange(1, 366, 10):
                    dag = i
                    ny_effekt += get_kWh(calc_effect_24_hours())
                if ny_effekt < effekt:
                    return j-1
                effekt = ny_effekt
            effekt = ny_effekt
    return 0
```

Figur 6: Algoritmen för att hitta den optimala altituden för solpanelen (programkod)

För uppskatta hur många år det tar för solpanelen att betala av sig själv krävdes inte några större mer avancerade uträkningar utan solpanelens kostnad dividerades med energin genererat under ett år multiplicerat med kostnaden per kWh enligt

$$\frac{200\,000 \text{ kr}}{2\text{kr} \cdot E} = x \quad (\text{x})$$

där E = energi i kWh per år och x = antal år.

För implementering av motorer på solpanelen som tillåter solpanelen att ändra vinklar i förhållande till solen för en maximal effekt, antogs azimutvinkeln altituden för både solpanelen och solen att vara lika stora. När dessa är vinklar antar samma storlek blir beräkningen för solstrålarnas effekt på solpanelen endast beroende utav solens instrålnignes intensitet enligt ekvation (7)

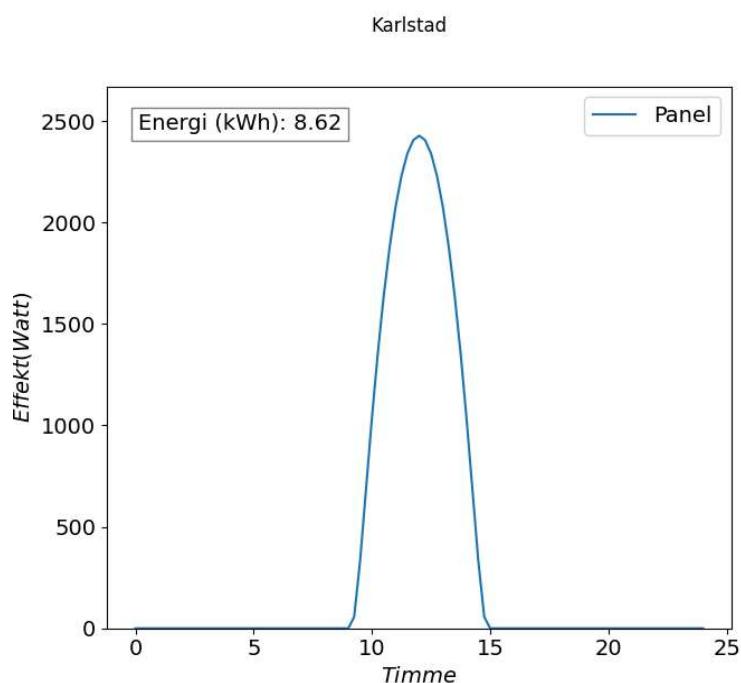
där I_p = solpanelens effekt och I = solstrålarnas effekt på jorden. Detta då uttrycket som I multipliceras med antar värdet 1. Med hänsyn till detta beräknades energin under ett år för en motordriven solpanel på samma vis som en icke motordriven solpanel med justeringen ut av att solpanelens effekt = solinstrålningens intensitet enligt funktionen nedan.

```
#Retunerar en array med 49 index som erhåller effekten för varje halvtimme
def calc_motordriven_panel_24_hours():
    array = [0] * 97
    for t in np.arange(0, 24, 0.25):
        P = effekt * get_solintensitet(get_solaltitud(t)) * area
        if P < 0:
            P = 0
        array[int(t * 4)] = P
    return array
```

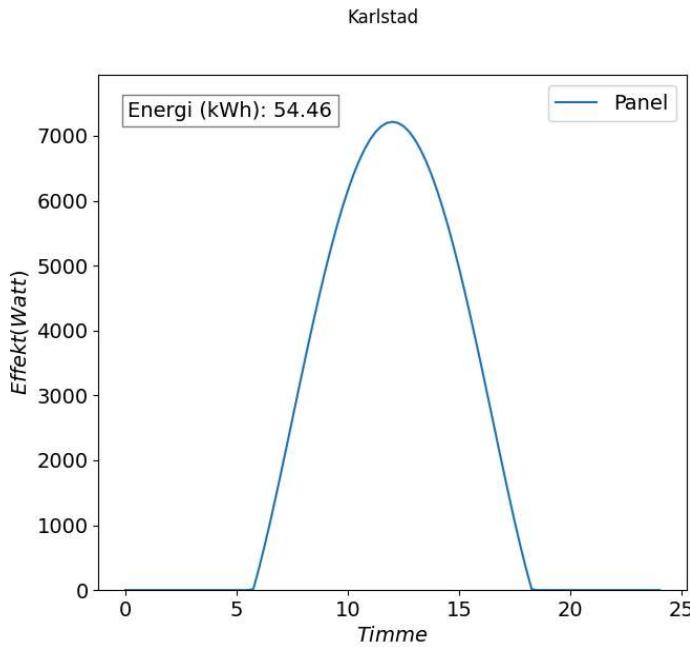
Figur 7: Funktion för att beräkna effekten under 24 timmar för en motordriven solpanel
(programkod)

Resultat

Arbetet har resulterat i ett program som kan grafiskt representera vad en teoretisk solpanel kan förväntas leverera i effekt baserat på geografisk plats (stad i Sverige), väderstreck riktning, och altitud för solpanel. Programmet kan räkna ut den uppskattade energin i kiloWattimmar med och utan hänsyn till antal uppskattade soltimmar, det vill säga den tid då solens instrålning är över 120 W/m^2 . För grupp nr 5 skulle programmet kunna plotta och beräkna alla deluppgifter för staden Karlstad. Då användaren trycker på generera effekt och energi för en dag dyker plottar upp som liknar följande figurer.

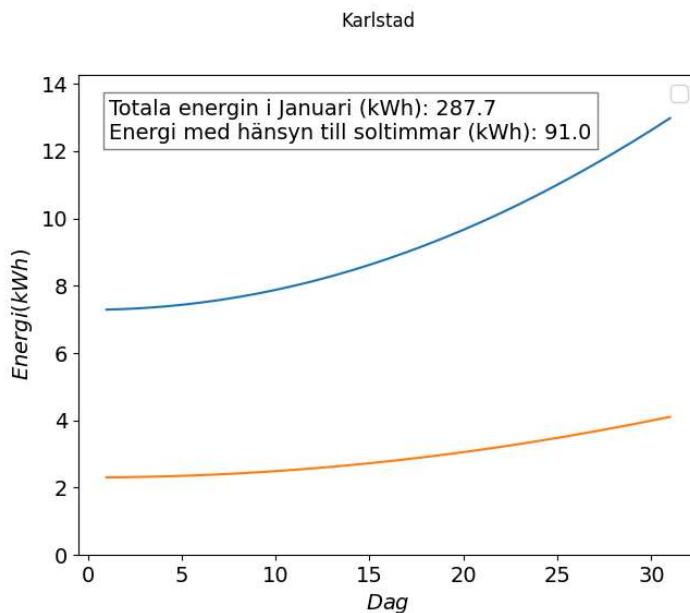


Figur 8: Effektkurva och beräknad energi för den 15 januari (genererad figur av programmet)

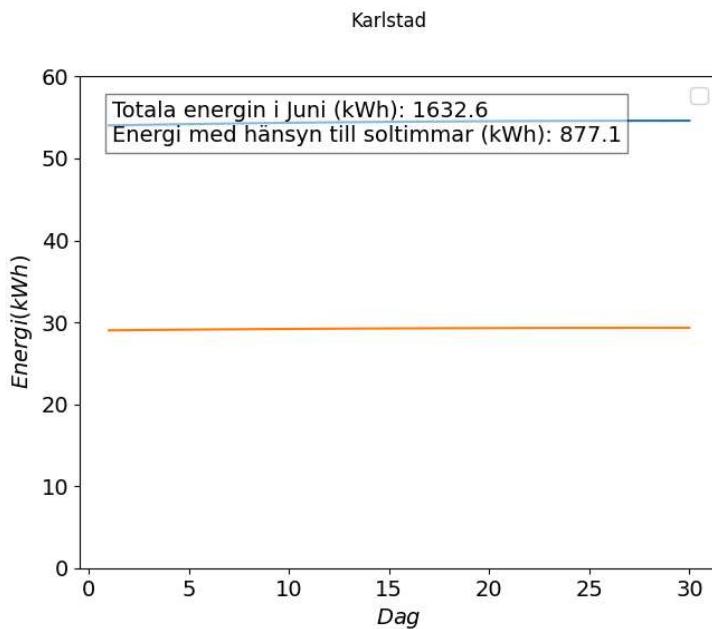


Figur 9: Effektkurva och beräknad energi för den 15 juni (genererad figur av programmet)

Användaren kan mata in önskbar månad i kombination boxen för alla månader på året och trycka på knappen bredvid "Plotta och visa energi per månad" för att få upp figurer med med energi på y-axeln och dagen i månaden på x-axeln. Figuren visar då två kurvor, en blå kurva som inte tar hänsyn till antalet soltimmar och en orange kurva som använder sannolikheten som beräknats med hjälp av tabell 2 i problembeskrivningen[1]. I en ruta skrivs den beräknade energin ut i kWh för båda kurvornas integraler.

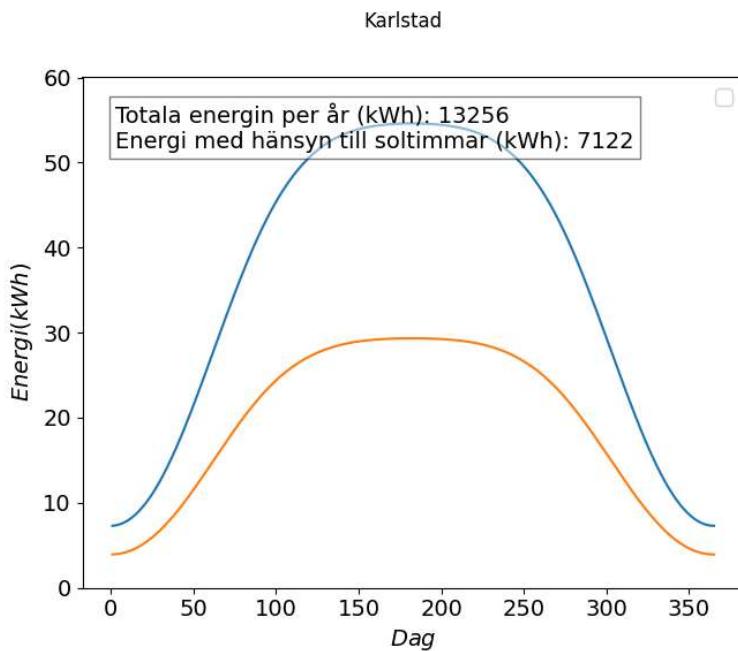


Figur 10: Kurvor och beräkning av energi med och utan hänsyn till antal soltimmar för Januari månad (genererad figur av programmet)



Figur 11: Kurvor och beräkning av energi med och utan hänsyn till antal soltimmar för Juni
(genererad figur av programmet)

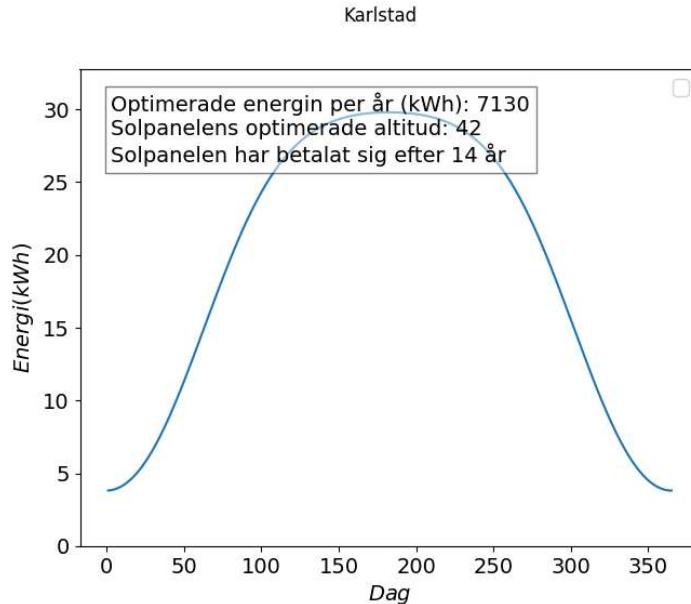
Om användaren vill se energin för solpanelen under ett år med och utan hänsyn till antal soltimmar under ett helt år kan användaren generera kurvor och beräkningar genom att trycka på “Plotta och visa energin för ett helt år” och få upp följande figur för staden karlstad.



Figur 12: Kurvor och beräkning av energi för ett helt år för en solpanel i Karlstad (genererad figur av programmet)

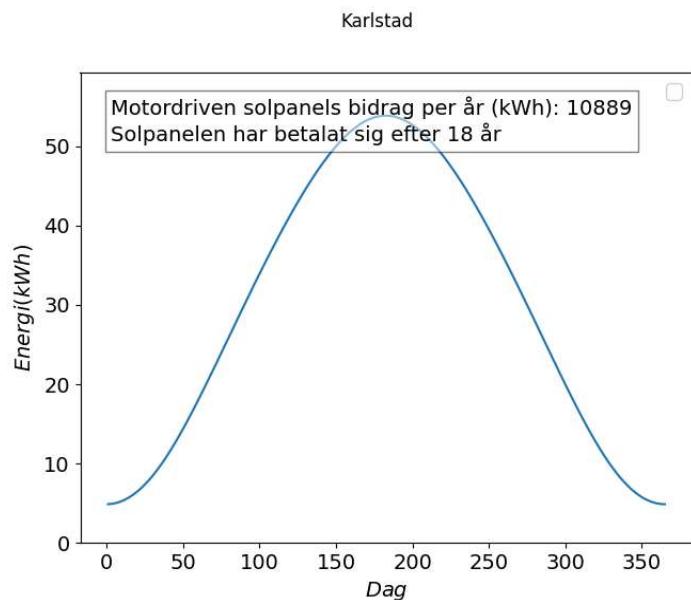
För att se hur kurvan och den totala energin hade sett ut för ett år om panelen monteras med en optimerad altitud trycker användaren på “Plotta energi för ett år med optimerad altitud”, då

visas den resulterade kurvan med hänsyn till antalet soltimmar för staden och använder sig utav algoritmen för att hitta den optima altituden. I figuren skrivs även den beräknade energin för året ut med hänsyn till antalet soltimmar, solpanelens optima altitud i grader och efter hur många år som solpanelen har betalat av sig själv om den kostar 200 000 kr.



Figur 13: Genererad figur för solpanel med optimerad altitud (genererad figur av programmet)

Användaren trycker på "Plotta energi under ett år med motordriven solpanel" för att få upp en figur där användaren kan se den totala energin och efter hur många år den motordrivna solpanelen tar för att betala av sig själv om den kostar 400 000 kr.



Figur 14: Genererad figur för motordriven solpanel i vald stad med inforuta (genererad figur av programmet)

Diskussion

Ekman Solar Technology är ett nystartat företag som jobbar med solceller och dess beräkningar. Företaget hade i uppdrag att skapa ett program som tar hänsyn till olika parametrar för att utföra sina beräkningar och optimera under diverse förhållanden. Programmet bör kunna plotta effekten som funktion av tiden på dygnet för en stationär solpanel i januari och juni månad då det antas vara soligt väder dygnet runt. Även skulle energiproduktionen beräknas för månaderna januari och juni samt energiproduktionen för ett helt år. Beräkningarna utförs både med och utan hänsyn till antalet soltimmar . Ytterligare en funktion för programmet var att optimera altituden för att maximera energiproduktionen.

Kostnaden för att installera en stationär solpanel är 200 000 kr, underskott/överskott av el kan antingen köpas eller säljas för 2 kr/kWh. Med denna information vill företaget räkna på efter hur många år solpanelen betalar sig med den optimerade altituden om ett hushåll förbrukar 14 000 kWh/år. Programmet ska även kunna beräkna energiproduktionen för en motordriven solpanel som följer solens rörelse för optimal effekt under ett helt år. En motordriven solpanel kostar dubbelt så mycket som den stationära panelen, det vill säga 400 000 kr och programmet ska även kunna ange antal år för när den anses att ha betalat av sig själv.

Uppgifterna bör lösas med programmeringsspråket python och är baserade på information från den givna problembeskrivningen[1] och dess tabeller. Tabell 1 innehåller latitud (ϕ), azimutvinkel (α_p) och altitud (θ_p) för olika städer i Sverige. Tabell 2 innehåller solskenstiden i timmar per månad för de relevanta städerna.

Välet av att göra ett användargränssnitt som tillåter användaren att välja olika städer och göra beräkningar för alla månader och dagar på året, gjordes för att kunna jämföra plottar och göra antagande om hur korrekt programmet simulerar datan. Att plotta figurer för månader och år var även ett val med samma baktanke. Dessa tillägg var till stor nytta under konstrueringen av programmet. Figurerna skvallrade även om hur pass noga programmets beräkningar är, det är tydligt att programmet inte jobbar tidskontinuerligt utan använder sig av större tidsdiskreta steg som motsvarar mätvärdet var 15:de minut. För att få mer exakta beräkningar kunde vi ha minskat dessa samplings steg, dock görs detta med utbyte av processorkraft och tid.

Utvecklarna av programmet valde att multiplicera sannolikheten för vädret med hela den genererade energin för var dag under året vilket kunde ha implementerats mer

verklighetstroget genom att till exempel använda en slumpmässig-nummer-generator i koden som använde sannolikheten för väder omständigheterna vid varje timme. Detta hade resulterat i olika värden för den totala energin med hänsyn till soltimmar varje gång vi genererade figuren samt en väldigt ojämnn kurva. Det var önskvärt men tidspressat.

Vid beräkning av den optimala altituden för solpanelen användes steg om 10 dagar under året för att lätta på processorkraften vilket hade haft en större inverkan på resultatet om vi hade implementerat sannolikheten för väder omständigheterna genom att använda en slumpmässig-nummer-generator. Eftersom att vi fortfarande har en jämn kurva efter beräkning av energin med hänsyn till soltimmar i programmet, spelar inte 10 dagars steget någon större roll på resultatet, om vi hade haft en ojämnn kurva hade 10 dagars steget kunnat resultera i felaktiga beräkningar för den optimerade altituden vilket även var en faktor till valet av att beräkna soltimmarna på det sätt som det görs.

Tolkningen av vad det innebär att ”solpanelen betalar av sig själv” spelar stor roll på resultatet för beräkningen av antal år. Gruppen valde först att beräkna antal år för att solpanelen ska anses som att har betalat av sig själv med dessa handskrivna beräkningar nedan i figur x.

$$f = 14\ 000 \text{ kWh}$$

t = tillverkning i kWh av solpanel

$$\text{net} = t \cdot 2 - (f - t) \cdot 2$$

$$\text{antal år} = \frac{200\ 000 \text{ kr}}{\text{net}}$$

Figur 15: Handskrivna beräkningar för deluppgift 4 (gruppens anteckningar)

Denna beräkning kunde resultera i negativa antal år då energin solpanelen genererat under ett år inte översteg halva förbrukningen i ett hushåll under denna tid. Detta tolkades som att solens instrålning inte gav solpanelen tillräcklig effekt och kommer aldrig att täcka kostnaden av energiförbrukningen. En annan tolkning är att all energi som solpanelen bidrar med antas bidra till att solpanelen betalar av sig själv. Denna tolkning var den som gruppen valde att utgå från i slutändan. Beräkningen för den senaste tolkningen krävde dock ingen information om hushållets förbrukning under ett år utan använde endast kvoten av solpanelens kostnad genom produkten av energin från solpanelen och kostnaden för en kWh.

Referenser

- [1] Jörgen Ekman, Magnus Ödmo, Fakulteten för teknik och samhälle MAU.
Au3_Problembeskrivning.pdf.
- [2] David Young, Shane Stadler. Physics 11th edition. Cutnell&johnsson, Wiley
- [3] Jörgen Ekman, Fakulteten för teknik och samhälle MAU. Föreläsningsslides.
- [4] <https://www.youtube.com/watch?v=Lq6LRgKpTw&t=48s>

Appendix A: Python-kod

```
# -*- coding: utf-8 -*-
"""

Created on Sat Mar 2 20:23:46 2024

@author: julia

"""

import numpy as np
import matplotlib.pyplot as plt
import tkinter as tk
from tkinter import ttk

#Tabell 1 i problembeskrivningen
stad_info = [[1, 68.4, 140, 20],
             [2, 65.6, 150, 25],
             [3, 63.8, 160, 30],
             [4, 63.2, 170, 35],
             [5, 59.4, 180, 40],
             [6, 59.3, 190, 45],
             [7, 57.7, 200, 50],
             [8, 57.6, 210, 55],
             [9, 56.9, 220, 60],
             [10, 55.7, 230, 65]]

#Tabell 2 i problembeskrivningen
solskenstid = [[0, 21, 87, 152, 195, 201, 197, 140, 84, 38, 1, 0, 1116],
               [23, 77, 162, 217, 281, 299, 295, 234, 153, 91, 37, 3, 1872],
               [35, 81, 156, 213, 270, 287, 276, 221, 153, 95, 43, 23, 1853],
               [32, 74, 142, 196, 237, 243, 244, 199, 132, 81, 38, 22, 1640],
               [49, 78, 160, 208, 267, 274, 274, 225, 167, 100, 47, 38, 1887],
               [44, 75, 151, 216, 277, 277, 280, 235, 170, 96, 45, 33, 1899],
               [47, 68, 144, 200, 252, 242, 246, 205, 150, 95, 44, 32, 1725],
               [41, 70, 156, 243, 317, 315, 314, 261, 188, 102, 42, 31, 2080],
               [37, 58, 129, 192, 234, 231, 234, 195, 140, 82, 33, 24, 1589],
               [47, 64, 135, 205, 253, 243, 247, 219, 160, 100, 47, 32, 1752]]
```

```

#Variablerna som aldrig förändras
area = 50      #Area på solpanelen
effekt = 0.15  #Effekten för solpanelen
I0 = 1360     #Totala energin(W) från fotoner som träffar solpanelen

#Variabler som förändrar beroende på val av stad/datum/månad
dag = 1          #Dag på året 1-365
latitud = np.deg2rad(68.4)      #Solpanelens latitud (radianer)
azimuntvinkel_panel = np.deg2rad(140) #Solpanelens azimutvinkel (radianer)
altitud_panel = np.deg2rad(20)      #Solpanelens altitud (radianer)

#Retunerar solens azimutvinkel (radianer)
def get_solazimuth(timvinkel, altitud, deklinationsvinkel):
    if timvinkel < 0:
        value = np.pi - np.arccos((np.sin(latitud) * np.sin(altitud) - np.sin(deklinationsvinkel)) /
(np.cos(latitud) * np.cos(altitud)))
    elif timvinkel > 0:
        value = np.pi + np.arccos((np.sin(latitud) * np.sin(altitud) - np.sin(deklinationsvinkel)) /
(np.cos(latitud) * np.cos(altitud)))
    else:
        value = np.pi
    return value

#Retunerar solens deklinationsvinkel (radianer)
def get_deklinationsvinkel(dag):
    return np.deg2rad(-23.44 * np.cos(np.deg2rad((360/365) * dag)))

#Retunerar timvinkeln (radianer)
def get_timvinkel(timme):
    timvinkel = (15 * timme) - 180
    if timvinkel == 0:
        timvinkel = 0.1
    return np.deg2rad(timvinkel)

#Retunerar solens altitud (radianer)
def get_solaltitud(t):

```

```

altitud = np.arcsin(np.sin(latitud) * np.sin(get_deklinationsvinkel(dag)) + np.cos(latitud) *
np.cos(get_deklinationsvinkel(dag)) * np.cos(get_timvinkel(t)))

if altitud < 0 or altitud > np.deg2rad(90):
    return 0
elif altitud >= 0 or altitud <= np.deg2rad(90):
    return altitud

#Retunerar I solens intensitet som når jordytan Wm^-2
def get_solinintensitet(solaltitud):
    if solaltitud == 0 or solaltitud == np.pi:
        return 0
    if solaltitud != 0 and solaltitud != np.pi:
        return 1.1 * IO * (0.7**2*(np.sin(solaltitud))**(-0.678))

#Retunerar intensiteten mot solpanelen Wm^-2
def get_panelintensitet(solaltitud, solazimut):
    return get_solinintensitet(solaltitud) * (np.cos(altitud_panel - solaltitud) * np.cos(azimuntvinkel_panel -
solazimut) + (1 - np.cos(azimuntvinkel_panel - solazimut)) * np.sin(altitud_panel) * np.sin(solaltitud))

#Retunerar integral av inparameter/array
def get_kWh(kurva):
    return (np.trapz(kurva, dx = 0.25)/1000)

#Plottar en kurva för effekten samt skriver ut energi i kWh för valt datum
def plot_effect_day(city):

    array = calc_effect_24_hours()
    fig, ax = plt.subplots()
    fig.suptitle(city)
    ax.set_ylimits(0, 1.1 * max(array))
    ax.plot(np.arange(0, 24.25, 0.25), array, label='Panel')
    ax.set_ylabel(r'$Effekt (Watt)$', fontsize=14)
    ax.set_xlabel(r'$Timme$', fontsize=14)
    ax.tick_params(labelsize=14)
    ax.legend(fontsize=14)
    text = "Energi (kWh): " + str(round(get_kWh(array), 2))
    ax.text(0.05, 0.95, text, transform=ax.transAxes, fontsize=14, verticalalignment='top',
bbox=dict(facecolor='white', alpha=0.5))

```

```

plt.show()

#Retunerar en array med 49 index som erhåller effekten för varje halvtimme under en dag
def calc_effect_24_hours():
    array = [0] * 97
    for t in np.arange(0, 24, 0.25):
        P = effekt * get_panelintensitet(get_solalitud(t), get_solazimut(get_timvinkel(t),
get_solalitud(t), get_deklinationsvinkel(dag))) * area
        if P < 0:
            P = 0
        array[int(t * 4)] = P
    return array

#Retunerar en array med 49 index som erhåller effekten för varje halvtimme under en dag när tetap = tetas
def calc_motordriven_panel_24_hours():
    array = [0] * 97
    for t in np.arange(0, 24, 0.25):
        P = effekt * get_solintensitet(get_solalitud(t)) * area
        if P < 0:
            P = 0
        array[int(t * 4)] = P
    return array

#Retunerar en array med index baserat på vald månad som erhåller varje dags totala energi kWh
def generate_energi_month(month):
    global dag
    energi_dag = 0
    energi_total = 0
    energi_array = []
    for i in get_day_range_for_month(month):
        dag = i;
        energi_dag = get_kWh(calc_effect_24_hours())
        energi_total += energi_dag
        energi_array.append(energi_dag)
    return energi_array

#Retunerar en range som tar hänsyn till årets dagar baserat på vald månad
def get_day_range_for_month(manad):

```

```

range = np.arange(1,31,1)
if manad == "Januari":
    range = np.arange(1, 32, 1)
if manad == "Februari":
    range = np.arange(32, 60, 1)
if manad == "Mars":
    range = np.arange(60, 91, 1)
if manad == "April":
    range = np.arange(91, 121, 1)
if manad == "Maj":
    range = np.arange(121, 152, 1)
if manad == "Juni":
    range = np.arange(152, 182, 1)
if manad == "Juli":
    range = np.arange(182, 213, 1)
if manad == "Augusti":
    range = np.arange(213, 244, 1)
if manad == "September":
    range = np.arange(244, 274, 1)
if manad == "Oktober":
    range = np.arange(274, 305, 1)
if manad == "November":
    range = np.arange(305, 335, 1)
if manad == "December":
    range = np.arange(335, 366, 1)
return range

```

```

#Retunerar tillgängliga soltimmar per månad enligt Jörgens soltimmes definition
def get_sun_hours(range):
    global dag
    hours = 0
    for i in range:
        dag = i,
        for t in np.arange(0,24, 1):
            I = get_solintensitet(get_solaltitud(t))
            if I > 120:
                hours += 1
    return hours

```

```
#En klass för att hantera GUIt
class SolarPanelApp:

    def __init__(self, root):

        self.root = root
        self.root.title("Solar Panel Energy Calculator")

        self.city_label = ttk.Label(root, text="Välj stad:")
        self.city_label.grid(row=0, column=0)

        self.city_combobox = ttk.Combobox(root, values=["Katterjåkk", "Luleå", "Umeå", "Östersund",
"Karlstad",
"Stockholm", "Göteborg", "Visby", "Växjö", "Lund"])
        self.city_combobox.set("Karlstad")
        self.city_combobox.grid(row=0, column=1)

        self.month_label = ttk.Label(root, text="Välj månad:")
        self.month_label.grid(row=1, column=0)

        self.month_combobox = ttk.Combobox(root, values=["Januari", "Februari", "Mars", "April",
"Maj", "Juni",
"Juli", "Augusti", "September", "Oktober", "November",
"December"])
        self.month_combobox.set("Januari")
        self.month_combobox.grid(row=1, column=1)

        self.day_label = ttk.Label(root, text="Välj dag:")
        self.day_label.grid(row=2, column=0)

        self.year_combobox = ttk.Combobox(root, values=["Januari", "Februari", "Mars", "April", "Maj",
"Juni",
"Juli", "Augusti", "September", "Oktober", "November",
"December"])
        self.year_combobox.config(width=13)
        self.year_combobox.grid(row=2, column=1, sticky="w")
        self.year_combobox.set("Juni")

        self.day_combobox = ttk.Combobox(root, values=list(range(1, 32)))
        self.day_combobox.config(width=3)
        self.day_combobox.set(15)
```

```

self.day_combobox.grid(row=2, column=1, sticky="e")

self.GM_label = ttk.Label(root, text="Plotta och visa energi per månad:")
self.GM_label.grid(row=5, column=0)
self.monthly_energy_button = ttk.Button(root, text="Generera",
command=self.generate_monthly_energy)
self.monthly_energy_button.grid(row=5, column=1)

self.GD_label = ttk.Label(root, text="Plotta effekt och visa energi för vald dag:")
self.GD_label.grid(row=4, column=0)
self.daily_energy_button = ttk.Button(root, text="Generera",
command=self.generate_daily_energy)
self.daily_energy_button.grid(row=4, column=1)

self.GD_label = ttk.Label(root, text="Plotta och visa energi för ett helt år:")
self.GD_label.grid(row=6, column=0)
self.yearly_energy_button = ttk.Button(root, text="Generera",
command=self.generate_yearly_energy)
self.yearly_energy_button.grid(row=6, column=1)

self.optimize_label = ttk.Label(root, text="Plotta energi för ett år med optimerad altitud: ")
self.optimize_label.grid(row=7, column=0)
self.optimize_button = ttk.Button(root, text="Generera",
command=self.generate_optimized_energy)
self.optimize_button.grid(row=7, column=1)

self.optimize_label = ttk.Label(root, text="Plotta energi under ett år med motordriven solpanel: ")
self.optimize_label.grid(row=8, column=0)
self.optimize_button = ttk.Button(root, text="Generera",
command=self.generate_motordriven_solpanel)
self.optimize_button.grid(row=8, column=1)

#Plotta energi för ett år med motorisk panel och beräkna lönsamhet
def generate_motordriven_solpanel(self):
    global dag, altitud_panel, azimuntvinkel_panel

    set_panel_info(self.city_combobox.get())
    energi = 0

```

```

array = [0]*365
city = self.city_combobox.get()
month = self.month_combobox.get()
kWh = 0

for i in np.arange(1, 366, 1):
    dag = i
    array_24_hours = get_kWh(calc_motordriven_panel_24_hours())
    kWh = (get_probability_of_sunshine(month, city) * array_24_hours)
    array[i-1] = kWh
    energi += kWh

fig, ax = plt.subplots()
fig.suptitle(city)
ax.set_ylim(0, 1.1 *max(array))
ax.plot(np.arange(1, len(array)+1), array)
ax.set_ylabel(r'$Energi (kWh)$', fontsize=14)
ax.set_xlabel(r'$Dag$', fontsize=14)
ax.tick_params(labelsize=14)
ax.legend(fontsize=14)
text = "Motordriven solpanelens bidrag per år (kWh): " + str(int((energi))) + "\n" + "Solpanelen har betalat sig efter " + str(get_payoff_years_400000kr(energi)) + " år"
ax.text(0.05, 0.95, text, transform=ax.transAxes, fontsize=14, verticalalignment='top',
bbox=dict(facecolor='white', alpha=0.5))
plt.show()

#Plotta energi för ett år med optimerad altitud vinkel och beräkna lönsamhet
def generate_optimized_energy(self):
    global dag, altitud_panel

    set_panel_info(self.city_combobox.get())
    teta_p = optimize_teta_p()
    altitud_panel = np.deg2rad(teta_p)
    energi = 0
    array = [0]*365
    city = self.city_combobox.get()
    month = self.month_combobox.get()
    kWh = 0

```

```

for i in np.arange(1, 366, 1):
    dag = i
    array_24_hours = get_kWh(calc_effect_24_hours())
    kWh = (get_probability_of_sunshine(month, city) * array_24_hours)
    array[i-1] = kWh
    energi += kWh

fig, ax = plt.subplots()
fig.suptitle(city)
ax.set_ylim(0, 1.1 * max(array))
ax.plot(np.arange(1, len(array)+1), array)
ax.set_ylabel(r'$Energi (kWh)$', fontsize=14)
ax.set_xlabel(r'$Dag$', fontsize=14)
ax.tick_params(labelsize=14)
ax.legend(fontsize=14)
text = "Optimerade energin per år (kWh): " + str(int(energi)) + "\n" + "Solpanelens optimerade
altitud: " + str(teta_p) + "\n" + "Solpanelen har betalat sig efter " + str(get_payoff_years_200000kr(energi))
+ " år"
    ax.text(0.05, 0.95, text, transform=ax.transAxes, fontsize=14, verticalalignment='top',
bbox=dict(facecolor='white', alpha=0.5))
plt.show()

#Plottar energin för den valda månaden
def generate_monthly_energy(self):

    set_panel_info(self.city_combobox.get())

    month = self.month_combobox.get()
    city = self.city_combobox.get()
    energi = 0
    energi_correction = 0
    array = generate_energi_month(month)
    energi_correction_array = []

    for i in np.arange(len(array)):
        energi += array[i]
        energi_correction_array.append((get_probability_of_sunshine(month, city) * array[i]))

```

```

energi_correction += energi_correction_array[i]

fig, ax = plt.subplots()
fig.suptitle(city)
ax.set_ylimit(0, 1.1 * max(array))
ax.plot(np.arange(1, len(array)+1), array)
ax.plot(np.arange(1, len(array)+1), energi_correction_array)
ax.set_ylabel(r'$Energi (kWh)$', fontsize=14)
ax.set_xlabel(r'$Dag$', fontsize=14)
ax.tick_params(labelsize=14)
ax.legend(fontsize=14)
text = "Totala energin i " + month + " (kWh): " + str(round(energi,1)) + "\n" + "Energi med hänsyn till soltimmar (kWh): " + str(round(energi_correction,1))
ax.text(0.05, 0.95, text, transform=ax.transAxes, fontsize=14, verticalalignment='top',
bbox=dict(facecolor='white', alpha=0.5))
plt.show()

#Initierar värden från GUI innan dagliga effekten och energin plottas
def generate_daily_energy(self):
    global dag
    set_panel_info(self.city_combobox.get())
    manad = int(switch_month(self.year_combobox.get()))
    nr = int(self.day_combobox.get())
    dag = manad+nr
    plot_effect_day(self.city_combobox.get())

#Plottar energin för året, tillämpliga soltimmar och soltimmar från tabell 2
def generate_yearly_energy(self):
    global dag
    set_panel_info(self.city_combobox.get())
    array = [0]*365;
    energi = 0
    energi_correction = 0
    energi_correction_array = [0]*365
    city = self.city_combobox.get()
    month = self.month_combobox.get()
    kWh = 0

```

```

for i in np.arange(1, 366, 1):
    dag = i
    kWh = get_kWh(calc_effect_24_hours())
    energi += kWh
    array[i-1] = kWh
    prop_kWh = (get_probability_of_sunshine(month, city) * kWh)
    energi_correction += prop_kWh
    energi_correction_array[i-1] = prop_kWh

fig, ax = plt.subplots()
fig.suptitle(city)
ax.set_ylim(0, 1.1 * max(array))
ax.plot(np.arange(1, len(array)+1), array)
ax.plot(np.arange(1, len(array)+1), energi_correction_array)
ax.set_ylabel(r'$Energi (kWh)$', fontsize=14)
ax.set_xlabel(r'$Dag$', fontsize=14)
ax.tick_params(labelsize=14)
ax.legend(fontsize=14)
text = "Totala energin per år (kWh): " + str(int(energi)) + "\n" + "Energi med hänsyn till soltimmar (kWh): " + str(int(energi_correction))
ax.text(0.05, 0.95, text, transform=ax.transAxes, fontsize=14, verticalalignment='top',
bbox=dict(facecolor='white', alpha=0.5))
plt.show()

#Retunerar sannoliketen för solskenstimmar under en månad
def get_probability_of_sunshine(month, city):
    switcher = {
        "Januari": 0,
        "Februari": 1,
        "Mars": 2,
        "April": 3,
        "Maj": 4,
        "Juni": 5,
        "Juli": 6,
        "Augusti": 7,
        "September": 8,
        "Oktober": 9,
        "November": 10,
        "December": 11
    }
    return switcher[month]

```

```

    "December":11,
}

tabell_solsken = solskensid[switch_city(city)][switcher.get(month)]
probability = tabell_solsken/get_sun_hours(get_day_range_for_month(month))
return probability

#Optimerar solpanelens altitud och retunerar vinkel
def optimize_teta_p():
    global altitud_panel, dag
    effekt = 0
    #Testar var tionde vinkel
    for p in np.arange(1,90,10):
        altitud_panel = np.deg2rad(p)
        ny_effekt = 0
        #Energi per år för test vinkeln
        for i in np.arange(1, 366, 10):
            dag = i
            ny_effekt += get_kWh(calc_effect_24_hours())
        #Om energin börjar avta hoppa in i en ny lop
        if ny_effekt < effekt:
            #Testa alla vinklar 9 steg före och efter nuvarande vinkel
            effekt = 0
            for j in np.arange(p-9,p+9,1):
                altitud_panel = np.deg2rad(j)
                ny_effekt = 0
                #Energi per år för test vinkeln
                for k in np.arange(1, 366, 10):
                    dag = i
                    ny_effekt += get_kWh(calc_effect_24_hours())
                if ny_effekt < effekt:
                    return j-1
                effekt = ny_effekt
            effekt = ny_effekt
        return 0

#Sätter solpanelens vinklar beroende på vald stad
def set_panel_info(stad):

```

```
global latitud, azimuntvinkel_panel, altitud_panel
i = int(switch_city(stad))
latitud = np.deg2rad(stad_info[i][1])
azimuntvinkel_panel = np.deg2rad(stad_info[i][2])
altitud_panel = np.deg2rad(stad_info[i][3])

#Retunerar int för vald stad som motsvarar rad i tabell 1
def switch_city(city):
    switcher = {
        "Katterjåkk": 0,
        "Luleå": 1,
        "Umeå": 2,
        "Östersund": 3,
        "Karlstad": 4,
        "Stockholm": 5,
        "Göteborg": 6,
        "Visby": 7,
        "Växjö": 8,
        "Lund": 9,
    }
    return switcher.get(city, "Invalid city")

#Retunerar första dagen i månaden
def switch_month(month):
    switcher = {
        "Januari": 0,
        "Februari": 31,
        "Mars": 59,
        "April": 90,
        "Maj": 120,
        "Juni": 151,
        "Juli": 181,
        "Augusti": 212,
        "September": 243,
        "Oktober": 273,
        "November": 304,
        "December": 334,
    }
```

```
return switcher.get(month, "Invalid month")

def get_payof_years_200000kr(kWh):
    return int(100000/kWh)

def get_payof_years_400000kr(kWh):
    return int(200000/kWh)

root = tk.Tk()
app = SolarPanelApp(root)
root.mainloop()
```