SUIVEUR DE SOLEIL

Enjeux sociétaux: environnement, sécurité, énergie

Numéro du candidat:22356

En quoi le système suiveur de soleil est un optimiseur de collecte d'énergie solaire?

Plan

I-Fonctionnement du suiveur solaire

II-Etude de la trajectoire du soleil et de son rayonnement

1-Trajectoire solaire

2-Rayonnement solaire

III-Etude du caractère optimisateur du suiveur

1-Effet du soleil sur le suiveur

2-Ratio GHI/POA

3- Etude du Gain de transposition

INTRODUCTION

Kalahari

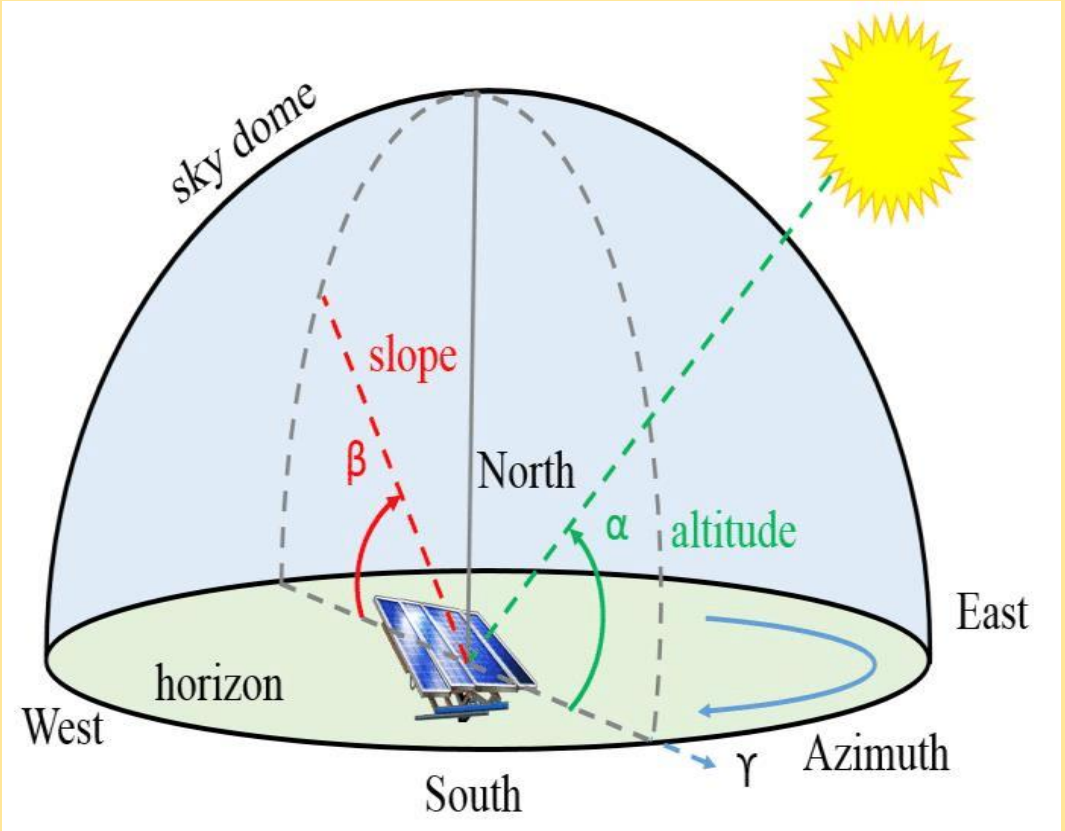
Désert d'Arabie

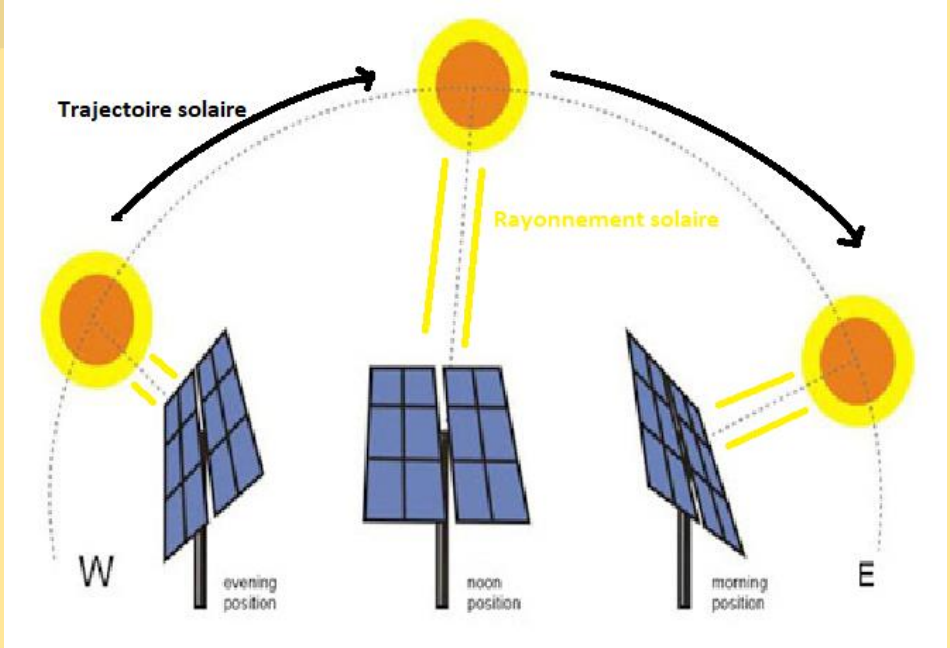
Désert de Gobi

Sahara

Projet Désertec

I- Fonctionnement du suiveur solaire



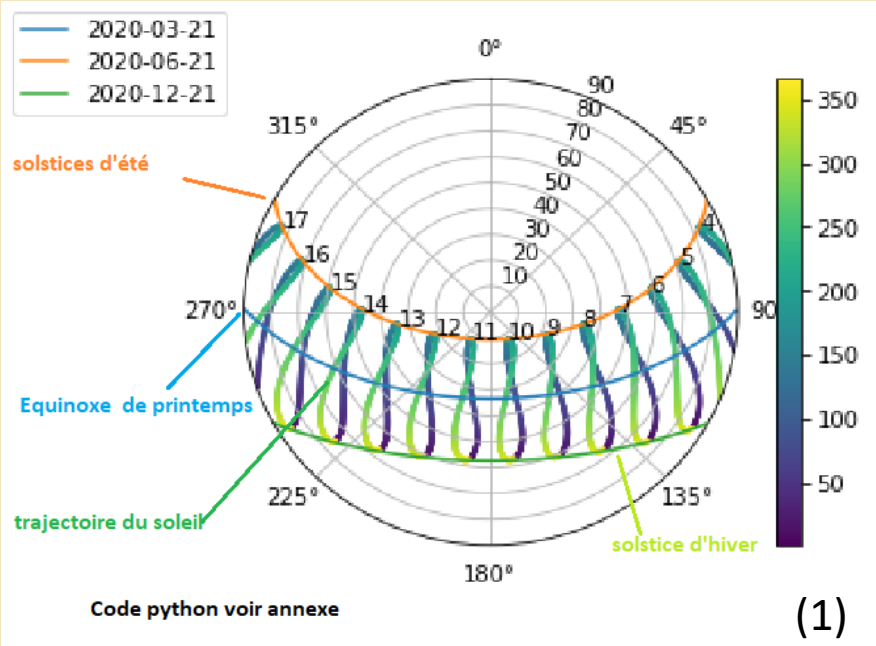


La suivi du soleil se fait selon deux axes principaux : l'azimut et l'élévation.

II-Etude de la trajectoire solaire et de son rayonnement

1-Trajectoire solaire

a-coordonnées polaires



Code python voir annexe

(1)

La déclinaison du soleil se calcule à partir de l'égalité suivante:

N:Nombre de jours depuis minuit a du 1er janvier

$$\delta = 23{,}41 \sin\left(\frac{360}{365}[N+284]\right) \quad (2)$$

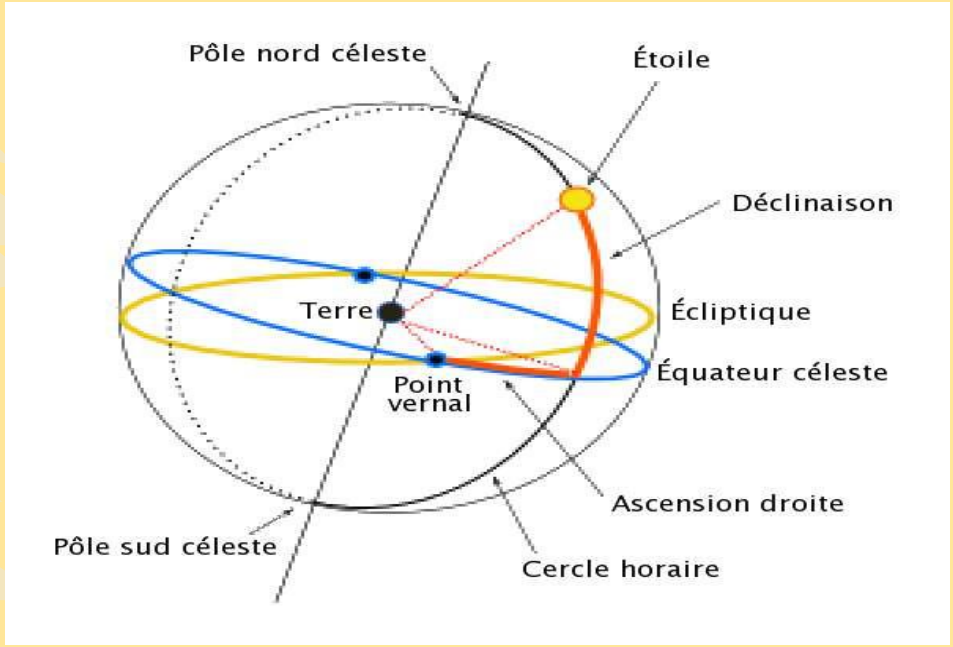
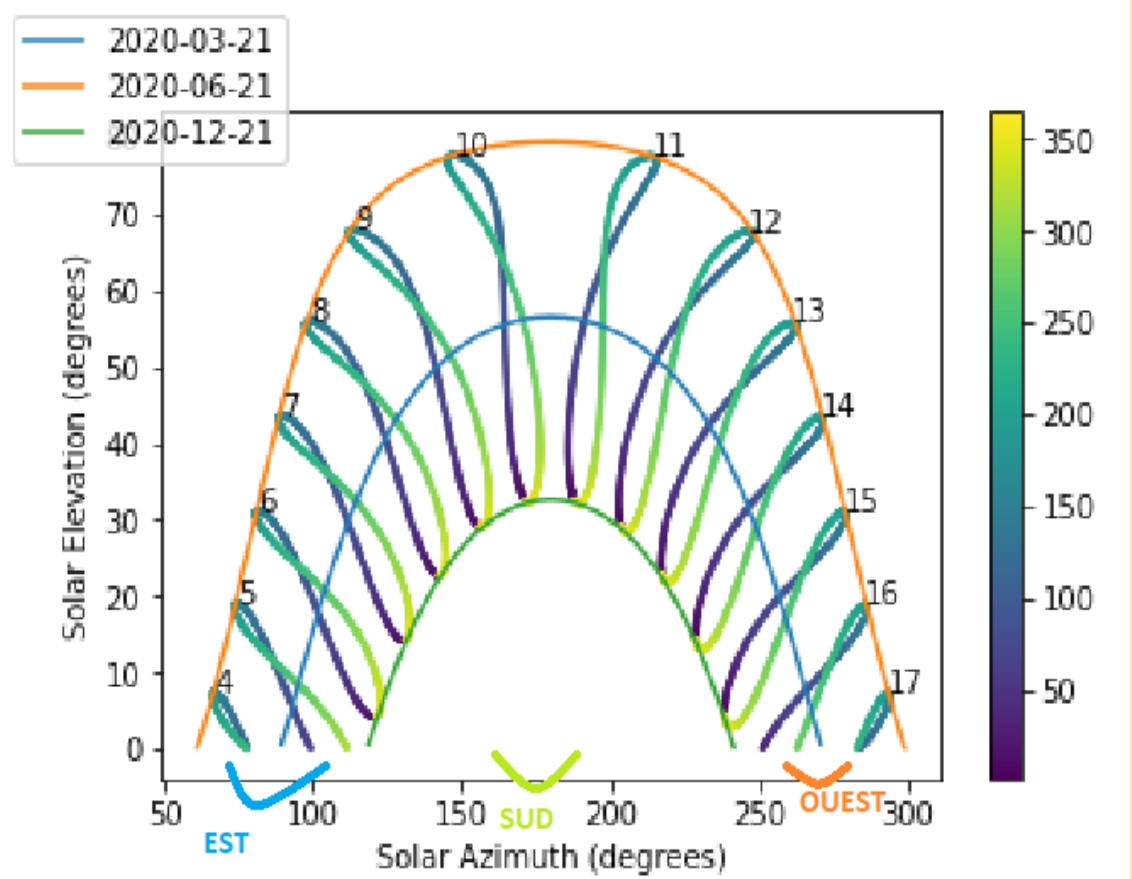


Fig:schéma des coordonnées équatoriales (https://www.futura-sciences.com/sciences/definitions/univers-systeme-coordonnees-equatoriales-22/)

b-coordonnées cartésiennes

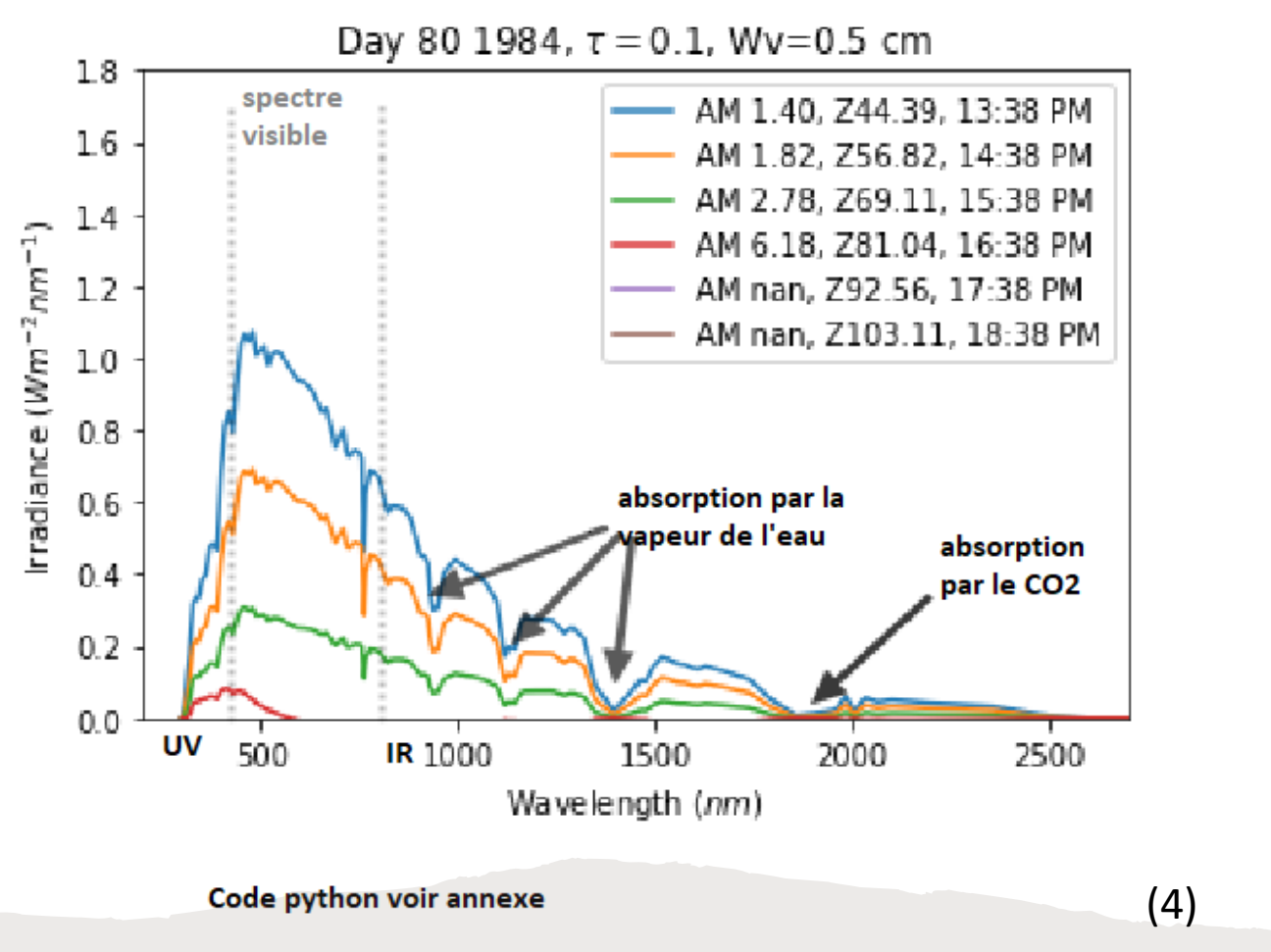
- Les trajectoires du soleil durant les solstices d'été ,d'hiver et l'équinoxe de printemps sont clairement différentes.
- En été, le soleil est très haut tandis qu'il est très bas en hiver.

La trajectoire du soleil en une position géographique dépend de la période de l'année.



(3)

2-Spectre de l'irradiance solaire



Code python voir annexe

(4)

- Les photons atteignent la terre avec une énergie dont la relation est :

$$E = \frac{hc}{\lambda}$$

- La loi de Planck donne :

$$I(\nu, t) = \frac{2h\nu^3}{c^2} * \frac{1}{e^{\frac{h\nu}{kT}} - 1}$$

- Longueur d'onde: nm
- Irradiance :W/m²*nm

III-Etude du caractère optimisateur du suiveur

1-Effet du soleil sur le suiveur

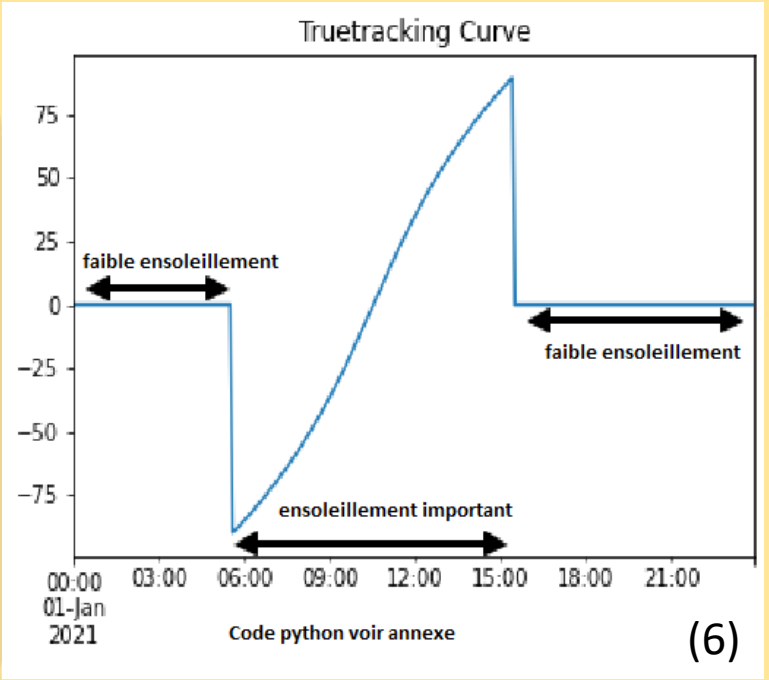
a-truetracking (5)

- Convertir les angles d'azimut et de lacets solaires dans le cadre de référence NED.

$$\gamma'_s = 360 - \gamma_s$$
$$\gamma'_t = 360 - \gamma_t$$

- Recherche de l'angle de suivi idéal, sans retour en arrière ni arrêts mécaniques.

$$a = \cos(\gamma'_s)\sin(\theta_z)\sin(\gamma'_s) - \sin(\gamma'_t)\sin(\theta_z)\cos(\gamma'_s)$$
$$b = \cos(\theta_z)$$
$$\varphi = -\arctan\left(\frac{b}{a}\right)$$



(6)

γ_s :angle du soleil en azimuth

γ_t :angle de lacet du tracker

θ_z :angle du soleil par rapport au zenith

ϕ : angle du tracker

b-backtracking (7)

- Calcul de l'angle de suivi :

$$GCR = \frac{W_T}{\Delta pp}$$

$$\varphi_t = arcos(GCR)$$

- Le suivi n'est pas affecté tant que la condition suivante est vérifiée:

$$\text{abs}(\varphi_{t,ideal}) < \varphi_{t,cutoff}$$

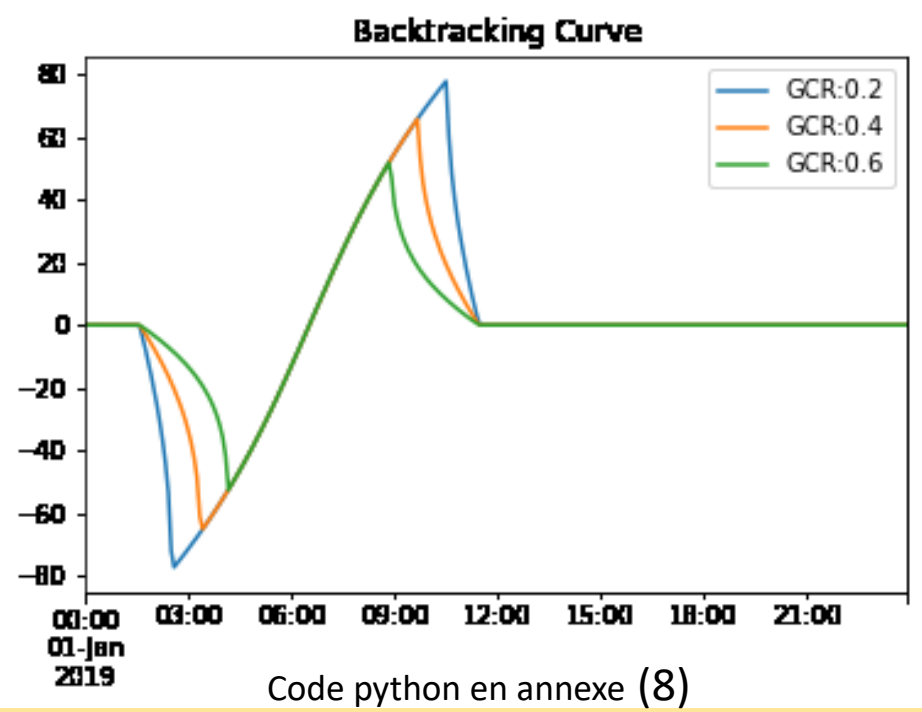
- Dans le cas contraire, l'angle du tracker suit l'algorithme suivant:

$$si\,(\varphi_{t,ideal} < 0)\; alors\; \varphi_t = 90 - \arcsin\left(\frac{(\sin(90 - \varphi_{t,ideal}))}{GCR}\right) + \varphi_{t,ideal}$$

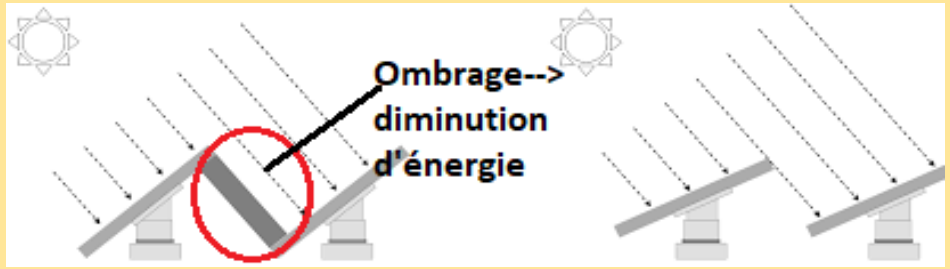
$$si\,(\varphi_{t,ideal} > 0)\; alors\; \varphi_t = \arcsin\left(\frac{(\sin(90 - \varphi_{t,ideal}))}{GCR}\right) - 90 + \varphi_{t,ideal}$$

WT:largeur du panneau solaire
Δpp: espace entre deux suiveurs
ϕt,cutoff : angle de suivi

b-bactracking



Code python en annexe (8)

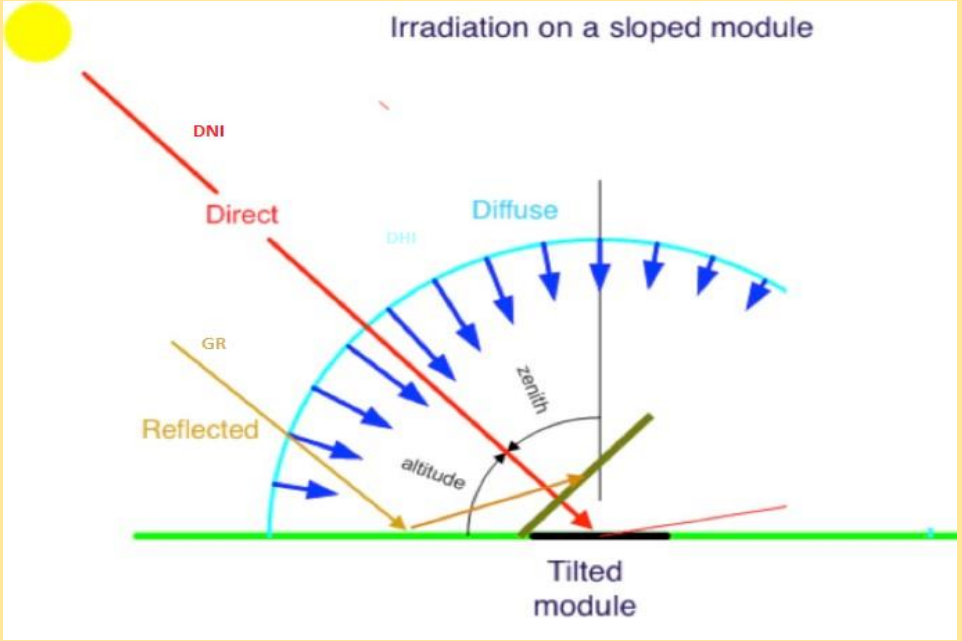


Sans backtracking

Avec backtracking

2-Ratio GHI/POA

- **GHI = DHI + DNI + GR**
- GHI: Irradiance horizontale globale
- DHI: Irradiance horizontale diffuse
- DNI: Irradiance normale directe



- **POA=GDirect + Gdiffuse + Grégléchi**
- Gdirect =DNI * cos(AOI)
- Gdiffus = DHI * [1+ cos(tilt)]/2
- Grégléchi =GHI * albedo *[1+ cos(tilt)]/2

GHI/POA: kWh/m² * nm

- On simule sur python deux courbes comparant GHI et POA en hiver et en été

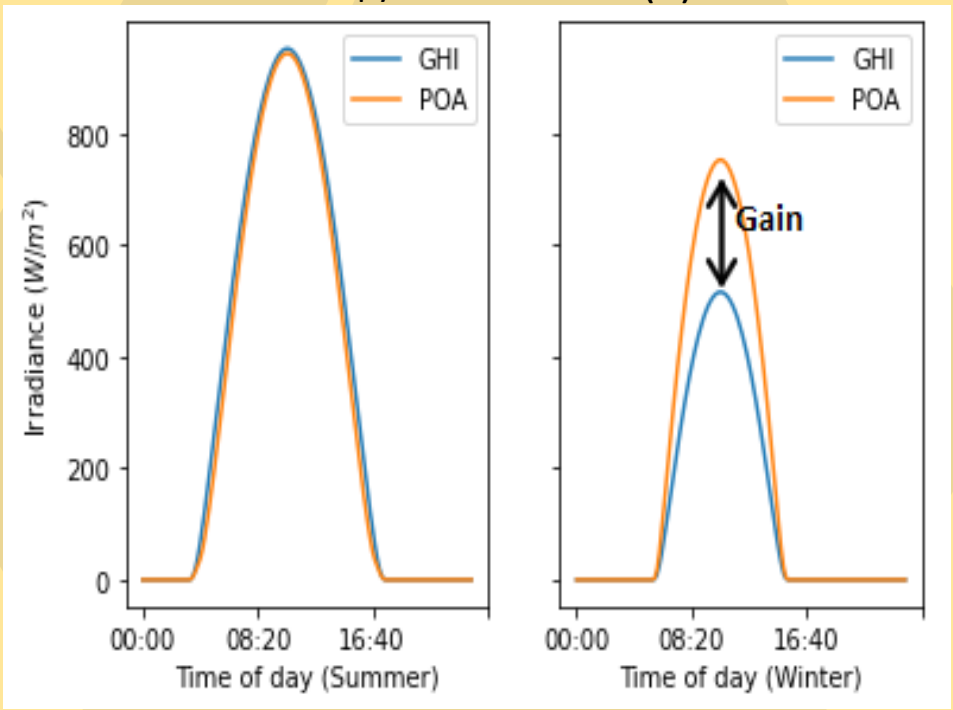
Résultat et interprétation

- En été, Le soleil en position très haute ne permet pas de gain pour une position inclinée par rapport à une horizontale.

- En hiver , sa position très basse permet d'obtenir un gain significatif donc un réseau incliné sera à un angle plus optimal par rapport à un réseau plat

La variation angulaire des panneaux du suiveur permet d'obtenir un gain d'énergie solaire

Code python en annexe (9)



Comparaison du gain GHI/POA pendant l'été et l'hiver

3-Comparaison d'un panneau solaire fixe et un suiveur solaire

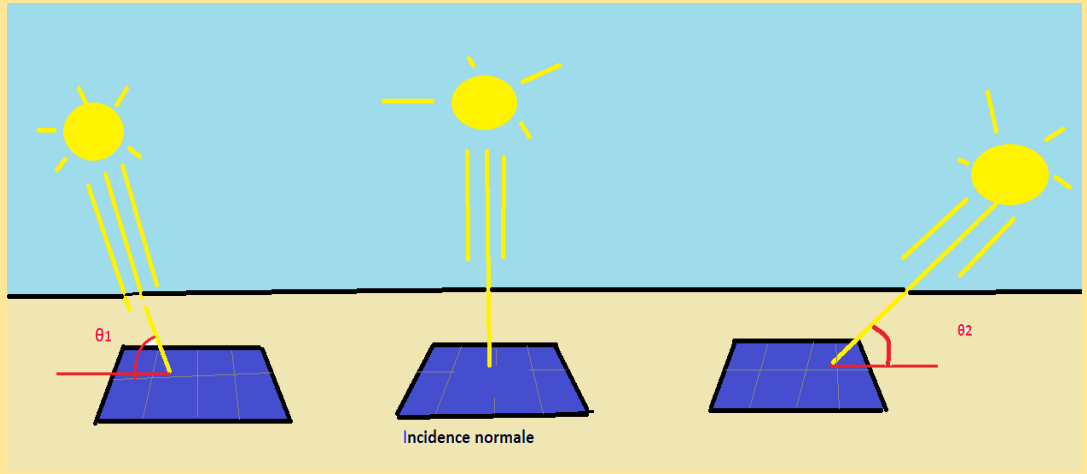
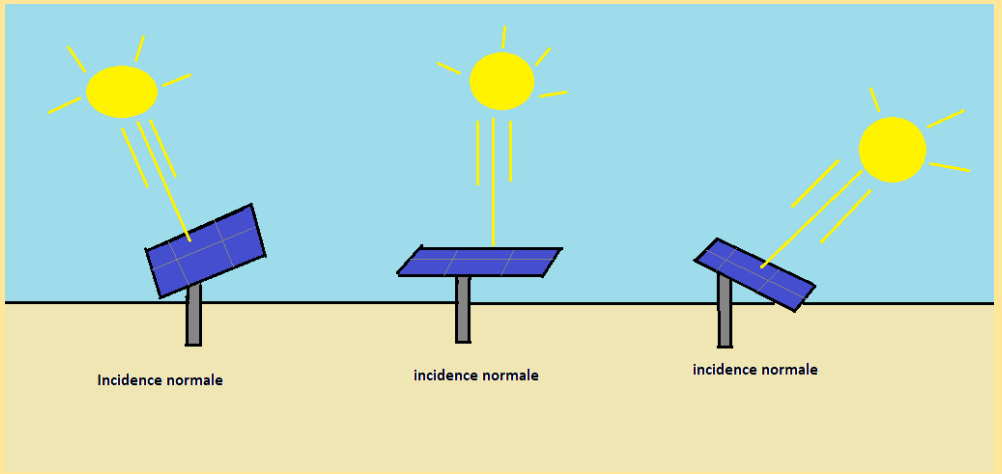


Fig:rayons soleils sur Panneau fixe

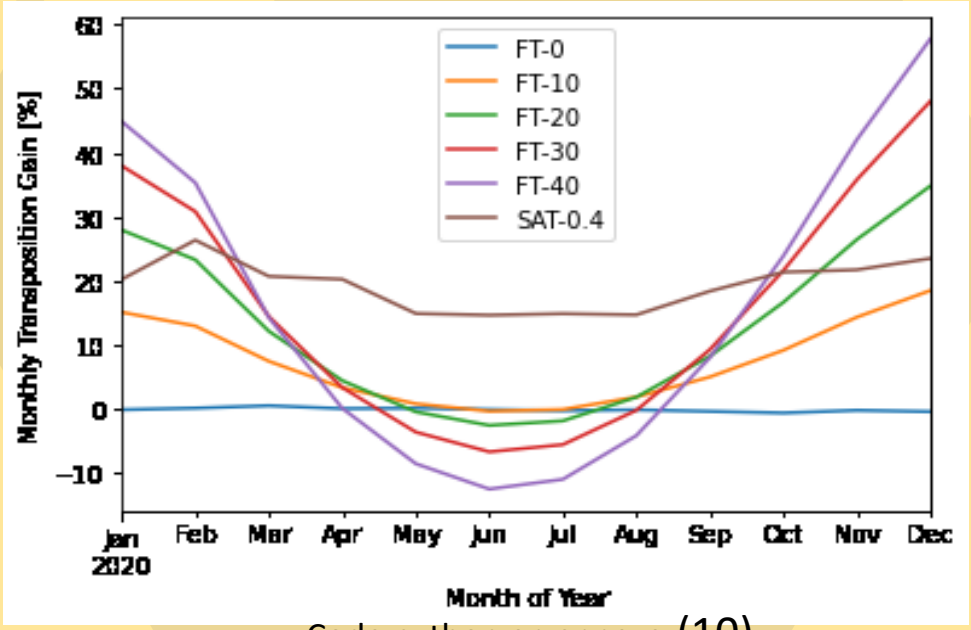


- Fig:rayons soleils sur suiveur solaire

- Pertes liées à l'orienation des panneaux:l'angle formé par les rayons solaires et le panneau solaire est appelé angle d'incidence θ
- Pour un rendement maximum, il faut un angle θ=90°

- Le gain pour les inclinaisons fixes est saisonnier
- Le gain pour un système suiveur est globalement constant sur l'année
- Pour le panneau horizontal il n'y a pas de gain

Compararaison du gain mensuel rapport au ratio GHI/POA entre un suiveur solaire,des panneaux fixes inclinées



Code python en annexe (10)

Résultat

D’après la figure précédente ,le gain du suiveur par rapport aux panneaux inclinés est d'environ 25%

Conclusion

- Le suiveur de soleil suit effectivement la trajectoire du soleil qui dépend de la position géographique et de la saison.
- L'orientation des panneaux permet d'obtenir un gain en énergie solaire.

Le système suiveur de soleil est donc un optimisateur de réception d'energie solaire.

ANNEXE

[1]

1/2

```
from pvlib import solarposition
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

tz = 'Etc/GMT+1'
lat, lon = 33.907, 8.046

times = pd.date_range('2020-01-01 00:00:00', '2021-01-01', closed='left',
                      freq='H', tz=tz)
solpos = solarposition.get_solarposition(times, lat, lon)
# remove nighttime
solpos = solpos.loc[solpos['apparent_elevation'] > 0, :]

ax = plt.subplot(1, 1, 1, projection='polar')
# draw the analemma loops
points = ax.scatter(np.radians(solpos.azimuth), solpos.apparent_zenith,
                    s=2, label=None, c=solpos.index.dayofyear)
ax.figure.colorbar(points)

# draw hour labels
for hour in np.unique(solpos.index.hour):
    # choose label position by the smallest radius for each hour
    subset = solpos.loc[solpos.index.hour == hour, :]
    r = subset.apparent_zenith
    pos = solpos.loc[r.idxmin(), :]
    ax.text(np.radians(pos['azimuth']), pos['apparent_zenith'], str(hour))
```

2/2

```
# draw individual days
for date in pd.to_datetime(['2020-03-21', '2020-06-21', '2020-12-21']):
    times = pd.date_range(date, date+pd.Timedelta('24h'), freq='5min', tz=tz)
    solpos = solarposition.get_solarposition(times, lat, lon)
    solpos = solpos.loc[solpos['apparent_elevation'] > 0, :]
    label = date.strftime('%Y-%m-%d')
    ax.plot(np.radians(solpos.azimuth), solpos.apparent_zenith, label=label)

ax.figure.legend(loc='upper left')

# change coordinates to be like a compass
ax.set_theta_zero_location('N')
ax.set_theta_direction(-1)
ax.set_rmax(90)

plt.show()
```

[2]- https://www.sciencedirect.com/topics/engineering/solar-declination

[3]

1/2

```
from pvlib import solarposition
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

tz = 'Etc/GMT+1'
lat, lon = 33.907, 8.046

times = pd.date_range('2019-01-01 00:00:00', '2020-01-01', closed='left',
                      freq='H', tz=tz)

solpos = solarposition.get_solarposition(times, lat, lon)
# remove nighttime
solpos = solpos.loc[solpos['apparent_elevation'] > 0, :]

fig, ax = plt.subplots()
points = ax.scatter(solpos.azimuth, solpos.apparent_elevation, s=2,
                    c=solpos.index.dayofyear, label=None)
fig.colorbar(points)

for hour in np.unique(solpos.index.hour):
    # choose label position by the largest elevation for each hour
    subset = solpos.loc[solpos.index.hour == hour, :]
    height = subset.apparent_elevation
    pos = solpos.loc[height.idxmax(), :]
    ax.text(pos['azimuth'], pos['apparent_elevation'], str(hour))
```

2/2

```
for date in pd.to_datetime(['2020-03-21', '2020-06-21', '2020-12-21']):
    times = pd.date_range(date, date+pd.Timedelta('24h'), freq='5min', tz=tz)
    solpos = solarposition.get_solarposition(times, lat, lon)
    solpos = solpos.loc[solpos['apparent_elevation'] > 0, :]
    label = date.strftime('%Y-%m-%d')
    ax.plot(solpos.azimuth, solpos.apparent_elevation, label=label)

ax.figure.legend(loc='upper left')
ax.set_xlabel('Solar Azimuth (degrees)')
ax.set_ylabel('Solar Elevation (degrees)')

plt.show()
```

[4]

1/2

```
from pvlib import solarposition
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

tz = 'Etc/GMT+1'
lat, lon = 33.907, 8.046

times = pd.date_range('2019-01-01 00:00:00', '2020-01-01', closed='left',
                      freq='H', tz=tz)

solpos = solarposition.get_solarposition(times, lat, lon)
# remove nighttime
solpos = solpos.loc[solpos['apparent_elevation'] > 0, :]

fig, ax = plt.subplots()
points = ax.scatter(solpos.azimuth, solpos.apparent_elevation, s=2,
                    c=solpos.index.dayofyear, label=None)
fig.colorbar(points)

for hour in np.unique(solpos.index.hour):
    # choose label position by the largest elevation for each hour
    subset = solpos.loc[solpos.index.hour == hour, :]
    height = subset.apparent_elevation
    pos = solpos.loc[height.idxmax(), :]
    ax.text(pos['azimuth'], pos['apparent_elevation'], str(hour))
```

2/2

```
for date in pd.to_datetime(['2020-03-21', '2020-06-21', '2020-12-21']):
    times = pd.date_range(date, date+pd.Timedelta('24h'), freq='5min', tz=tz)
    solpos = solarposition.get_solarposition(times, lat, lon)
    solpos = solpos.loc[solpos['apparent_elevation'] > 0, :]
    label = date.strftime('%Y-%m-%d')
    ax.plot(solpos.azimuth, solpos.apparent_elevation, label=label)

ax.figure.legend(loc='upper left')
ax.set_xlabel('Solar Azimuth (degrees)')
ax.set_ylabel('Solar Elevation (degrees)')

plt.show()
```

[5],[7] - https://plantpredict.com/algorithm/solar-position/

[6]

```
from pvlib import solarposition, tracking
import pandas as pd
import matplotlib.pyplot as plt

tz = 'Etc/GMT+1'
lat, lon = 33.90789698269054, 8.046793436207476

times = pd.date_range('2021-01-01', '2021-01-02', closed='left', freq='5min',
                      tz=tz)
solpos = solarposition.get_solarposition(times, lat, lon)

truetracking_angles = tracking.singleaxis(
    apparent_zenith=solpos['apparent_zenith'],
    apparent_azimuth=solpos['azimuth'],
    axis_tilt=0,
    axis_azimuth=180,
    max_angle=90,
    backtrack=False,  # for true-tracking
    gcr=0.5)  # irrelevant for true-tracking

truetracking_position = truetracking_angles['tracker_theta'].fillna(0)
truetracking_position.plot(title='Truetracking Curve')

plt.show()
```

[8]

```
fig, ax = plt.subplots()

for gcr in [0.2, 0.4, 0.6]:
    backtracking_angles = tracking.singleaxis(
        apparent_zenith=solpos['apparent_zenith'],
        apparent_azimuth=solpos['azimuth'],
        axis_tilt=0,
        axis_azimuth=180,
        max_angle=90,
        backtrack=True,
        gcr=gcr)

    backtracking_position = backtracking_angles['tracker_theta'].fillna(0)
    backtracking_position.plot(title='Backtracking Curve',
                               label=f'GCR:{gcr:0.01f}',
                               ax=ax)

plt.legend()
plt.show()
```

[9]

1/3

```
from pvlib import location
from pvlib import irradiance
import pandas as pd
from matplotlib import pyplot as plt

# Pour cette exemple j'ai utilisé la localisation zne
tz = 'Etc/GMT+1'
lat, lon = 33.907, 8.046

# Create location object to store lat, lon, timezone
site = location.Location(lat, lon, tz=tz)


# Calculate clear-sky GHI and transpose to plane of array
# Define a function so that we can re-use the sequence of operations with
# different locations
def get_irradiance(site_location, date, tilt, surface_azimuth):
    # Creates one day's worth of 10 min intervals
    times = pd.date_range(date, freq='10min', periods=6*24,
                          tz=site_location.tz)
    # Generate clearsky data using the Ineichen model, which is the default
    # The get_clearsky method returns a dataframe with values for GHI, DNI,
    # and DHI
    clearsky = site_location.get_clearsky(times)
    # Get solar azimuth and zenith to pass to the transposition function
    solar_position = site_location.get_solarposition(times=times)
```

2/3

```
    # Use the get_total_irradiance function to transpose the GHI to POA
    POA_irradiance = irradiance.get_total_irradiance(
        surface_tilt=tilt,
        surface_azimuth=surface_azimuth,
        dni=clearsky['dni'],
        ghi=clearsky['ghi'],
        dhi=clearsky['dhi'],
        solar_zenith=solar_position['apparent_zenith'],
        solar_azimuth=solar_position['azimuth'])
    # Return DataFrame with only GHI and POA
    return pd.DataFrame({'GHI': clearsky['ghi'],
                         'POA': POA_irradiance['poa_global']})


# Get irradiance data for summer and winter solstice, assuming 25 degree tilt
# and a south facing array
summer_irradiance = get_irradiance(site, '06-20-2020', 25, 180)
winter_irradiance = get_irradiance(site, '12-21-2020', 25, 180)

# Convert Dataframe Indexes to Hour:Minute format to make plotting easier
summer_irradiance.index = summer_irradiance.index.strftime("%H:%M")
winter_irradiance.index = winter_irradiance.index.strftime("%H:%M")
```

3/3

```
# Plot GHI vs. POA for winter and summer
fig, (ax1, ax2) = plt.subplots(1, 2, sharey=True)
summer_irradiance['GHI'].plot(ax=ax1, label='GHI')
summer_irradiance['POA'].plot(ax=ax1, label='POA')
winter_irradiance['GHI'].plot(ax=ax2, label='GHI')
winter_irradiance['POA'].plot(ax=ax2, label='POA')
ax1.set_xlabel('Time of day (Summer)')
ax2.set_xlabel('Time of day (Winter)')
ax1.set_ylabel('Irradiance ($W/m^2$)')
ax1.legend()
ax2.legend()
plt.show()
```

[10]

1/3

```
import pvlib
from pvlib import location
from pvlib import irradiance
from pvlib import tracking
from pvlib.iotools import read_tmy3
import pandas as pd
from matplotlib import pyplot as plt
import pathlib

# get full path to the data directory
DATA_DIR = pathlib.Path(pvlib.__file__).parent / 'data'

# get TMY3 dataset
tmy, metadata = read_tmy3(DATA_DIR / '723170TYA.CSV', coerce_year=2020)
# TMY3 datasets are right-labeled (AKA "end of interval") which means the last
# interval of Dec 31, 23:00 to Jan 1 00:00 is labeled Jan 1 00:00. When rolling
# up hourly irradiance to monthly insolation, a spurious January value is
# calculated from that last row, so we'll just go ahead and drop it here:
tmy = tmy.iloc[:-1, :]

# create location object to store lat, lon, timezone
location = location.Location.from_tmy(metadata)

# calculate the necessary variables to do transposition.  Note that solar
# position doesn't depend on array orientation, so we just calculate it once.
# Note also that TMY datasets are right-labeled hourly intervals, e.g. the
# 10AM to 11AM interval is labeled 11.  We should calculate solar position in
# the middle of the interval (10:30), so we subtract 30 minutes:
times = tmy.index - pd.Timedelta('30min')
solar_position = location.get_solarposition(times)
```

2/3

```
# but remember to shift the index back to line up with the TMY data:
solar_position.index += pd.Timedelta('30min')


# create a helper function to do the transposition for us
def calculate_poa(tmy, solar_position, surface_tilt, surface_azimuth):
    # Use the get_total_irradiance function to transpose the irradiance
    # components to POA irradiance
    poa = irradiance.get_total_irradiance(
        surface_tilt=surface_tilt,
        surface_azimuth=surface_azimuth,
        dni=tmy['DNI'],
        ghi=tmy['GHI'],
        dhi=tmy['DHI'],
        solar_zenith=solar_position['apparent_zenith'],
        solar_azimuth=solar_position['azimuth'],
        model='isotropic')
    return poa['poa_global']  # just return the total in-plane irradiance


# create a dataframe to keep track of our monthly insolations
df_monthly = pd.DataFrame()

# fixed-tilt:
for tilt in range(0, 50, 10):
    # we will hardcode azimuth=180 (south) for all fixed-tilt cases
    poa_irradiance = calculate_poa(tmy, solar_position, tilt, 180)
    column_name = f"FT-{tilt}"
    # TMYs are hourly, so we can just sum up irradiance [W/m^2] to get
    # insolation [Wh/m^2]:
    df_monthly[column_name] = poa_irradiance.resample('m').sum()
```

3/3

```
# single-axis tracking:
orientation = tracking.singleaxis(solar_position['apparent_zenith'],
                                  solar_position['azimuth'],
                                  axis_tilt=0,  # flat array
                                  axis_azimuth=180,  # south-facing azimuth
                                  max_angle=60,  # a common maximum rotation
                                  backtrack=True,  # backtrack for a c-Si array
                                  gcr=0.4)  # a common ground coverage ratio

poa_irradiance = calculate_poa(tmy,
                               solar_position,
                               orientation['surface_tilt'],
                               orientation['surface_azimuth'])
df_monthly['SAT-0.4'] = poa_irradiance.resample('m').sum()

# calculate the percent difference from GHI
ghi_monthly = tmy['GHI'].resample('m').sum()
df_monthly = 100 * (df_monthly.divide(ghi_monthly, axis=0) - 1)

df_monthly.plot()
plt.xlabel('Month of Year')
plt.ylabel('Monthly Transposition Gain [%]')
plt.show()
```