(700.288) Research Project in Smart Grids

MODELING OF PHOTOVOLTAIC SYSTEMS USING PYTHON

Mohammad Monsef

Supervisor Professor:
Univ.-Prof. Dipl.-Ing.Dr. Wilfried Elmenreich

Contents

Introduction:

Photovoltaic (PV) systems play a very important role in the transition toward efficiency and sustainable energy solutions. Modeling and simulating these systems is a critical step in analyzing their behavior under various environmental and operational conditions to apply the best solution. In the book, MATLAB version coding has been used for this purpose due to its robust numerical computing capabilities and widespread use in engineering academia and industry. This report presents a complete conversion of selected experiments and models from the book "Modeling of Photovoltaic Systems Using MATLAB: Simplified Green Codes" by Tamer Khatib and Wilfried Elmenreich into Python version coding. The book offers a detailed, component-based approach to modeling PV systems providing a realistic view for simulation, covering solar radiation, PV module characteristics, energy flow, and optimization techniques. While the original implementations for experiments were done in MATLAB , the growing popularity of Python as an open-source coding makes it important to provide equivalent implementations in Python coding for a broader use. The objective of this report is:

1. To faithfully reproduce the MATLAB-based experiments and simulations in Python, maintaining the original logic and structure as closely as possible.

2. To demonstrate the usability and flexibility of Python for PV system modeling and analysis, using useful libraries such as NumPy, pandas, matplotlib, etc.

By translating these models into Python, this paper tries to make the methodologies from the original book more accessible to audience and simplify further development and adaptation in Python-based simulation environments.

.

1 Chapter1: MODELING OF THE SOLAR SOURCE

Example 1.1 : Develop a program in Python that calculates the altitude and azimuth angles at 13:12 on July 2, for the city of Kuala Lumpur.

Solution (Implemented in Python)

The main parts of the program's structure are described as follows:

- Insert location coordinates (latitude and longitude), day number, and local mean time.
- Calculate angle of declination, equation of time, and LMST.
- Calculate AST and angle of the hour.
- Calculate altitude angle.
- Calculate azimuth angle.
- Plot results.

```python
import math
import numpy as np

# Location Kuala Lumpur, Malaysia, L = (3.12), LOD = (101.7)
L = 3.12
LOD = 101.7
N = 183
T_GMT = 8
LMT_minutes = 792  # LMT in minutes (13:12 = 792 minutes)


Ds = 23.45 * math.sin((360 * (N - 81) / 365) * (math.pi / 180))


B = (360 * (N - 81)) / 364
EoT = (9.87 * math.sin(2 * B * math.pi / 180)) - (7.53 * math.cos(B * math.
    pi / 180)) - (1.5 * math.sin(B * math.pi / 180))


Lzt = 15 * T_GMT


if LOD >= 0:
    Ts_correction = (-4 * (Lzt - LOD)) + EoT
else:
    Ts_correction = (4 * (Lzt - LOD)) + EoT

# Solar time
Ts = LMT_minutes + Ts_correction

# Hour angle degree
Hs = (15 * (Ts - (12 * 60))) / 60

```

```python

sin_Alpha = (math.sin(L * math.pi / 180) * math.sin(Ds * math.pi / 180)) + \
            (math.cos(L * math.pi / 180) * math.cos(Ds * math.pi / 180) * math.cos(Hs * math.pi / 180))
Alpha = math.degrees(math.asin(sin_Alpha))

# Azimuth angle calculation
Sin_Theta = (math.cos(Ds * math.pi / 180) * math.sin(Hs * math.pi / 180)) / math.cos(Alpha * math.pi / 180)
Theta = math.degrees(math.asin(Sin_Theta))


print(f"Altitude angle (Alpha): {Alpha:.2f} degrees")
print(f"Azimuth angle (Theta): {Theta:.2f} degrees")
```

Results

- **Altitude angle (Alpha):** `70.04°`
- **Azimuth angle (Theta):** `-3.31°`

****

Example 1.2: Modify the developed Python code in Example 1.1 to calculate the altitude and azimuth angle profile (every 5 min) for the whole solar day of the 2nd of July for the city of Kuala Lumpur.

Solution: The solar day is defined as the duration from sunrise to sunset. Thus, the altitude and azimuth angles are required to be calculated for each hour from sunrise to sunset. The sunrise and sunset hour angles can be considered equal and calculated as

$$\omega_{s,sr} = \cos^{-1}(-\tan L \tan \delta) \tag{1}$$

In the meanwhile, the solar time of each hour angle can be calculated by rewriting Equation 1.3 as follows:

$$\frac{\omega_{s,sr}}{15^\circ} \pm 12\ \mathrm{h} = \mathrm{AST}_{s,sr} \tag{2}$$

The sign of Equation above must be minus if we want to calculate the sunrise time, while it must be plus if we are calculating the sunset time. Following that the main parts of the program's structure can be described as follows:

- Insert location coordinates (latitude and longitude) and day number.
- Calculate angle of declination.
- Calculate sunrise and sunset hour angles.
- Calculate AST of the sunrise and sunset.
- Calculate equation of time and LMST.
- Calculate the actual sunrise and sunset times.
- Set for a loop starting from the sunrise and terminating by the sunset with a step size of 5 min.
- Calculate the solar time and hour angle at each step.
- Calculate altitude angle at each step.
- Calculate azimuth angle at each step.
- Store the calculated altitude and azimuth angles in arrays.
- Plot the results.

```
import math
import numpy as np
import matplotlib.pyplot as plt



L = 3.12
LOD = 101.7
N = 183
T_GMT = 8
Step = 5

Ds = 23.45 * math.sin((360 * (N - 81) / 365) * (math.pi / 180))
Lzt = 15 * T_GMT

B = (360 * (N - 81)) / 364
EoT = (9.87 * math.sin(2 * B * math.pi / 180)) - \
      (7.53 * math.cos(B * math.pi / 180)) - \
      (1.5 * math.sin(B * math.pi / 180))

if LOD >= 0:
    Ts_correction = (-4 * (Lzt - LOD)) + EoT
else:
    Ts_correction = (4 * (Lzt - LOD)) + EoT

Wsr_ssi = -math.tan(Ds * math.pi / 180) * math.tan(L * math.pi / 180)
Wsrsr_ss = math.degrees(math.acos(Wsr_ssi))

ASTsr = abs(((Wsrsr_ss / 15) - 12) * 60)
```

```python
ASTss = ((Wsrsr_ss / 15) + 12) * 60

Tsr = ASTsr + abs(Ts_correction)
Tss = ASTss + abs(Ts_correction)

Alpha = []
Theta = []
Time = []

LMT = Tsr
while LMT <= Tss:
    Ts = LMT + Ts_correction
    Hs = (15 * (Ts - (12 * 60))) / 60

    sin_Alpha = (math.sin(L * math.pi / 180) * math.sin(Ds * math.pi / 180)) + \
                (math.cos(L * math.pi / 180) * math.cos(Ds * math.pi / 180) * math.cos(Hs * math.pi / 180))

    Alpha_i = math.degrees(math.asin(sin_Alpha))
    Alpha.append(Alpha_i)

    Sin_Theta = (math.cos(Ds * math.pi / 180) * math.sin(Hs * math.pi / 180)) / math.cos(Alpha_i * math.pi / 180)
    Theta_i = math.degrees(math.asin(Sin_Theta))
    Theta.append(Theta_i)

    Time.append(LMT)
    LMT += Step

plt.figure(figsize=(10, 6))

plt.subplot(2, 1, 1)
plt.plot(Time, Alpha, label='Altitude (Alpha)')
plt.xlabel('Time (min)')
plt.ylabel('Angle (degree)')
plt.title('Altitude Angle vs Time')
plt.grid(True)
plt.xlim(400, 1200)

plt.subplot(2, 1, 2)
plt.plot(Time, Theta, color='red', label='Azimuth (Theta)')
plt.xlabel('Time (min)')
plt.ylabel('Angle (degree)')
plt.title('Azimuth Angle vs Time')
plt.grid(True)
plt.xlim(400, 1200)

plt.tight_layout()
plt.show()
```

results:

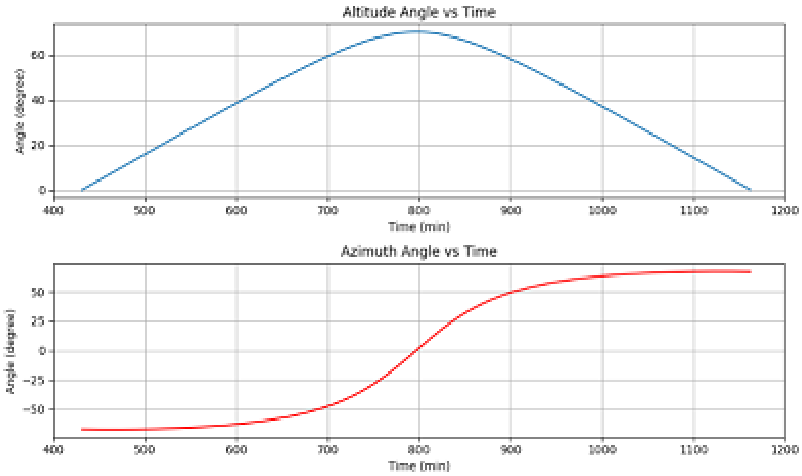


Figure 1: A day's profile of the Sun's altitude and azimuth angles (Example 1.2).

****

Example 1.3: Develop a Python code that calculates the spectral emissive power of a 288 K blackbody, for wavelengths in the range of (1–60) μm. After that calculate the power emitted between the wavelength of 20 and 30 μm.

Solution: The first part of the example can be solved by simply implementing Equation

$$E_\lambda = \frac{3.74 \times 10^8}{\lambda^5 \left[\exp\left(\frac{14{,}400}{\lambda T}\right) - 1\right]}$$

and calculating its value for the requested wavelength range as follows:

```
import numpy as np
import matplotlib.pyplot as plt


T = 288
E_lambda = []

for lamda in range(1, 61):  # from 1 to 60
    E_lambda_i = (3.74 * 10e8) / (lamda**5 * (np.exp(14400 / (lamda * T)) -
        1))
    E_lambda.append(E_lambda_i)

lamda_values = list(range(1, 61))


plt.plot(lamda_values, E_lambda)
plt.xlabel('Wavelength [~$\mu$.m]')
```

```python
plt.ylabel('Intensity [W/ m  ~$\mu$.m]')
plt.title('Spectral Energy Distribution')
plt.grid(True)
plt.show()
```

results:

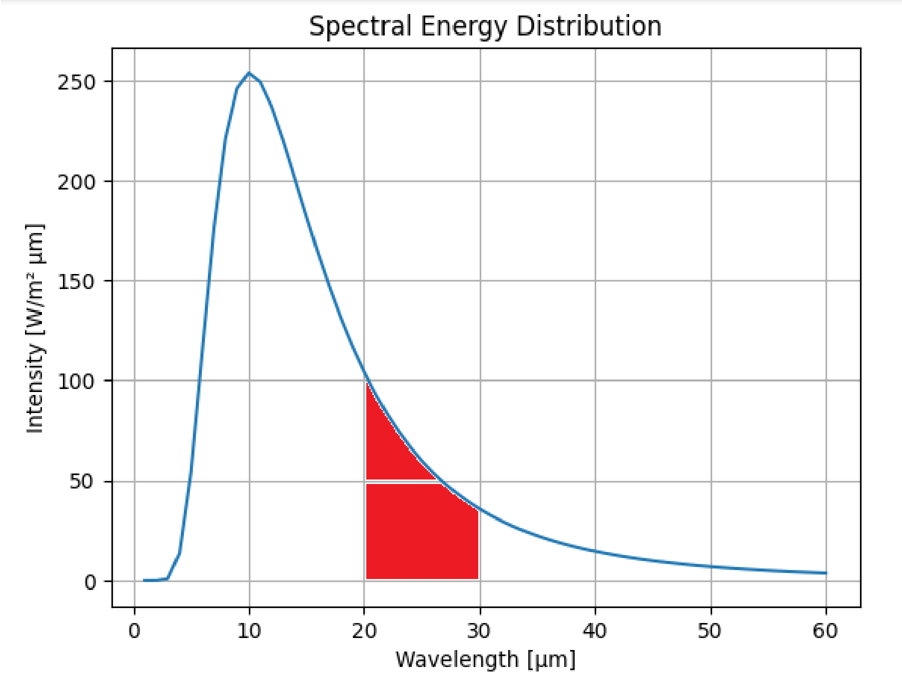


Figure 2: Spectral emissive power of a 288 K blackbody, for wavelengths in the range of (1–60) μm

In order to calculate the emitted power between the wavelength value of 20 and 30 μm., the shaded area in Figure 2 can be calculated as follows:

$$\sum_{\lambda=20}^{30} E_\lambda = \sum_{\lambda=20}^{30} \frac{3.74 \times 10^8}{\lambda^5 \left[\exp\left(\frac{14{,}400}{\lambda T}\right) - 1\right]}$$

```python

import numpy as np

T = 288
E_lambda = []

for lamda in range(20, 31):  # from 20 to 30 inclusive
    E_lambda_i = (3.74 * 10e8) / (lamda**5 * (np.exp(14400 / (lamda * T)) - 1))
    E_lambda.append(E_lambda_i)


E_lambda = np.array(E_lambda)

Power = np.sum(E_lambda)
print("Total Power:", Power)
```

answer= 704.0801 W/m2

****

Example 1.4: Develop a Python program that predicts the hourly extraterrestrial solar radiation profile for Nablus city, Palestine, on the 31st of March.

Solution: The hourly values of the altitude angle of the selected location must be calculated first. After that the value of the hourly solar radiation can be generated using Equation as follows:

$$G_{\mathrm{exH}} = G_0 \left(1 + 0.0333 \cos\left(\frac{360N}{365}\right)\right) (\sin L \sin\delta + \cos L \cos\delta \cos\omega)$$

```

import numpy as np
import matplotlib.pyplot as plt



L = 32.22
LOD = 35.27
N = 90
T_GMT = +3
Step = 60


Ds = 23.45 * np.sin((360 * (N - 81) / 365) * (np.pi / 180))

# Equation of time
B = (360 * (N - 81)) / 364
EoT = (9.87 * np.sin(2 * B * np.pi / 180)) - \
      (7.53 * np.cos(B * np.pi / 180)) - \
      (1.5 * np.sin(B * np.pi / 180))

Lzt = 15 * T_GMT
Ts_correction = (-4 * (Lzt - LOD) + EoT) if LOD >= 0 else (4 * (Lzt - LOD) + EoT)


Wsr_ssi = -np.tan(np.radians(Ds)) * np.tan(np.radians(L))
Wsrsr_ss = np.degrees(np.arccos(Wsr_ssi))

ASTsr = abs(((Wsrsr_ss / 15) - 12) * 60)
ASTss = ((Wsrsr_ss / 15) + 12) * 60

Tsr = ASTsr + abs(Ts_correction)
Tss = ASTss + abs(Ts_correction)


LMT = np.arange(Tsr, Tss + Step, Step)
sin_Alpha = []

for t in LMT:
```

```
    Ts = t + Ts_correction
    Hs = (15 * (Ts - 12 * 60)) / 60
    sin_Alpha_i = (np.sin(np.radians(L)) * np.sin(np.radians(Ds)) +
                   np.cos(np.radians(L)) * np.cos(np.radians(Ds)) * np.cos(np.radians(Hs)))
    sin_Alpha.append(sin_Alpha_i)

sin_Alpha = np.array(sin_Alpha)


Go = 1367~W/m^2

Gext = Go * (1 + 0.0333 * np.cos(np.radians(360 * N / 365)))
GextH = Gext * sin_Alpha


plt.figure(figsize=(8, 5))
plt.plot(LMT, GextH, color='gray', linestyle='-', linewidth=1)
plt.scatter(LMT, GextH, color='black', s=10)
plt.xlabel('Time (min)', fontsize=12)
plt.ylabel(r'Solar radiation (W/m$^2$)', fontsize=12)
plt.xlim(400, 1200)
plt.ylim(100, 1250)
plt.grid(True)
plt.tight_layout()
plt.show()
```

Result:

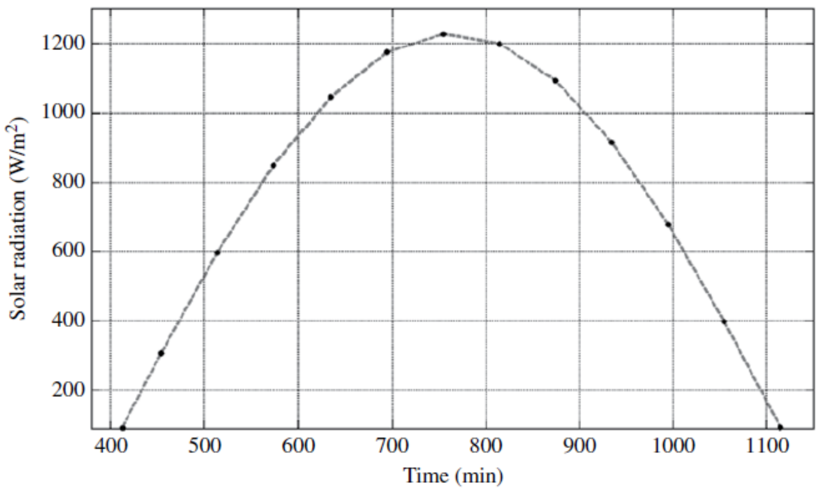


Figure 3: Daily extraterrestrial solar radiation for Nablus city

****

Example 1.5: Develop a Python code that predicts hourly global and diffuse solar radiation profile on a horizontal surface for Kuwait City, Kuwait, on the 2nd of May from sunrise time to sunset time.

Solution: The program required is divided into two parts: the calculation of the altitude angle and then the calculation of the solar radiation component. The first part is illustrated in previous examples like Example 1.2. In the meanwhile the second part, equations needed (from 1.19 to 1.24 in the book) are coded as follows:

```python
import numpy as np
import matplotlib.pyplot as plt

# Location: Kuwait City
L = 29.36
LOD = 47.97
N = 122
T_GMT = +3
Step = 60


Ds = 23.45 * np.sin(np.radians((360 * (N - 81)) / 365))


B = (360 * (N - 81)) / 364
EoT = (9.87 * np.sin(np.radians(2 * B))) - \
      (7.53 * np.cos(np.radians(B))) - \
      (1.5 * np.sin(np.radians(B)))


Lzt = 15 * T_GMT
Ts_correction = (-4 * (Lzt - LOD)) + EoT if LOD >= 0 else (4 * (Lzt - LOD)) + EoT


Wsr_ssi = -np.tan(np.radians(Ds)) * np.tan(np.radians(L))
Wsrsr_ss = np.degrees(np.arccos(Wsr_ssi))


ASTsr = abs(((Wsrsr_ss / 15) - 12) * 60)
ASTss = ((Wsrsr_ss / 15) + 12) * 60
Tsr = ASTsr + abs(Ts_correction)
Tss = ASTss + abs(Ts_correction)


LMT = np.arange(Tsr, Tss + Step, Step)
sin_Alpha = []

for t in LMT:
    Ts = t + Ts_correction
    Hs = (15 * (Ts - (12 * 60))) / 60
    sin_Alpha_i = (np.sin(np.radians(L)) * np.sin(np.radians(Ds)) +
                   np.cos(np.radians(L)) * np.cos(np.radians(Ds)) * np.cos(np.radians(Hs)))
    sin_Alpha.append(sin_Alpha_i)

sin_Alpha = np.array(sin_Alpha)


A = 1160 + (75 * np.sin(np.radians((360 / 365) * (N - 275))))
```

```python
k = 0.174 + (0.035 * np.sin(np.radians((360 / 365) * (N - 100))))
C = 0.095 + (0.04 * np.sin(np.radians((360 / 365) * (N - 100))))
G_B_norm = A * np.exp(-k / sin_Alpha)
G_B = G_B_norm * sin_Alpha
G_D = C * G_B_norm
G_T = G_B + G_D


valid_mask = sin_Alpha > 0.1
LMT_valid = LMT[valid_mask]
G_T_valid = G_T[valid_mask]


G_A = np.array([0.000, 0.2431, 0.4422, 0.5966, 0.865, 0.976,
                1.031, 1.016, 0.936, 0.788, 0.5904, 0.3541, 0.1439]) * 1e3
G_A = G_A[:len(LMT_valid)]


plt.figure(figsize=(9, 5))
plt.plot(LMT_valid, G_T_valid, 'k--o', markersize=4, label='Developed method')
plt.plot(LMT_valid, G_A, 'k-^', markersize=4, label='Actual data')
plt.xlabel('Time (min)', fontsize=12)
plt.ylabel(r'Global solar radiation (W/m$^2$)', fontsize=12)
plt.xlim(350, 1100)
plt.ylim(100, 1050)
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()
```

Results:

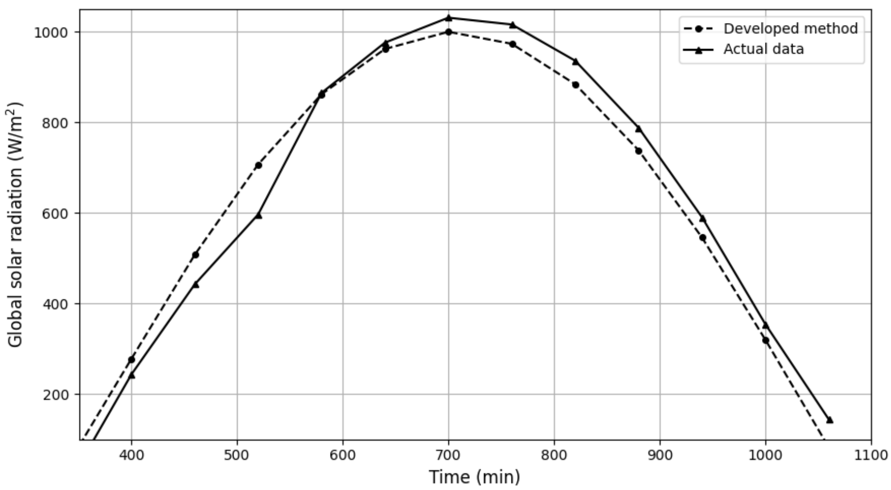


Figure 4: Global solar radiation for Kuwait City

****

Example 1.6: Develop a Python program that predicts the hourly global and diffuse solar radiation on a tilted surface for Kuwait City, Kuwait, on the 31st of March from sunrise time to sunset time. Assume that tilt angle is equal to latitude angle.

Solution:

```
import numpy as np
import matplotlib.pyplot as plt


L = 29.36
LOD = 47.97
N = 90
T_GMT = +3
Step = 60


Ds = 23.45 * np.sin(np.radians((360 * (N - 81)) / 365))


B = (360 * (N - 81)) / 364
EoT = (9.87 * np.sin(np.radians(2 * B))) - \
      (7.53 * np.cos(np.radians(B))) - \
      (1.5 * np.sin(np.radians(B)))

# Local Meridian Standard Time
Lzt = 15 * T_GMT
Ts_correction = (-4 * (Lzt - LOD)) + EoT if LOD >= 0 else (4 * (Lzt - LOD))
    + EoT


Wsr_ssi = -np.tan(np.radians(Ds)) * np.tan(np.radians(L))
Wsrsr_ss = np.degrees(np.arccos(Wsr_ssi))

ASTsr = abs(((Wsrsr_ss / 15) - 12) * 60)
ASTss = ((Wsrsr_ss / 15) + 12) * 60
Tsr = ASTsr + abs(Ts_correction)
Tss = ASTss + abs(Ts_correction)


LMT = np.arange(Tsr, Tss, Step)
sin_Alpha = []
for t in LMT:
    Ts = t + Ts_correction
    Hs = (15 * (Ts - 12 * 60)) / 60
    sin_Alpha_i = (np.sin(np.radians(L)) * np.sin(np.radians(Ds)) +
                   np.cos(np.radians(L)) * np.cos(np.radians(Ds)) * np.cos(
                       np.radians(Hs)))
    sin_Alpha.append(sin_Alpha_i)

sin_Alpha = np.array(sin_Alpha)


A = 1160 + (75 * np.sin(np.radians((360 / 365) * (N - 275))))
k = 0.174 + (0.035 * np.sin(np.radians((360 / 365) * (N - 100))))
C = 0.095 + (0.04 * np.sin(np.radians((360 / 365) * (N - 100))))
```

```
G_B_norm = A * np.exp(-k / sin_Alpha)
G_B = G_B_norm * sin_Alpha
G_D = C * G_B_norm
G_T = G_B + G_D


Beta = L
Rb = ((np.cos(np.radians(L - Beta)) * np.cos(np.radians(Ds)) * np.sin(np.radians(Wsrsr_ss))) +
      (np.radians(Wsrsr_ss)) * np.sin(np.radians(L - Beta)) * np.sin(np.radians(Ds))) / \
     ((np.cos(np.radians(L)) * np.cos(np.radians(Ds)) * np.sin(np.radians(Wsrsr_ss))) +
      (np.radians(Wsrsr_ss)) * np.sin(np.radians(L)) * np.sin(np.radians(Ds)))

Rd = (1 + np.cos(np.radians(Beta))) / 2
Rr = (0.3 * (1 - np.cos(np.radians(Beta)))) / 2

G_B_Beta = G_B * Rb
G_D_Beta = G_D * Rd
G_R = G_T * Rr
G_T_Beta = G_B_Beta + G_D_Beta + G_R


plt.figure(figsize=(9, 5))
plt.plot(LMT, G_T, color='orange', label='Global radiation on horizontal surface')
plt.plot(LMT, G_T_Beta, 'k', label='Global radiation on tilted surface')
plt.xlabel('Time (min)')
plt.ylabel(r'Solar radiation (W/m$^2$)')
plt.xlim(min(LMT), max(LMT))
plt.ylim(200, 1100)
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()
```

Results:

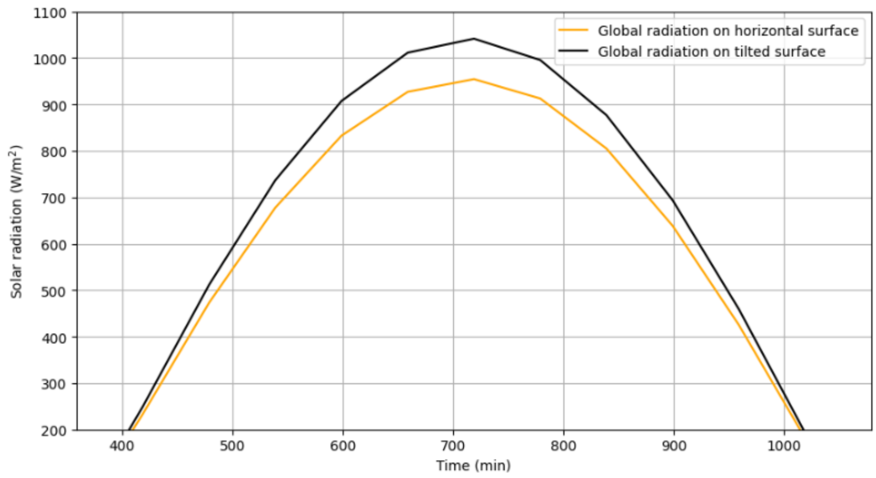


Figure 5: Global solar radiation on horizontal and tilted surfaces for Kuwait City

****

Example 1.7: Develop a linear model for a monthly average of daily solar radiation based on the data in the following table using Python. Assume that the solar time is equal to the local time.

Solution:

```
#Example 1.7
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt


file_path = 'PV Modeling Book Data Source.csv'
data = pd.read_csv(file_path)


N = pd.to_numeric(data.iloc[:, 2], errors='coerce')
LMT = pd.to_numeric(data.iloc[:, 3], errors='coerce')
G_T = pd.to_numeric(data.iloc[:, 4], errors='coerce')
S_So = pd.to_numeric(data.iloc[:, 8], errors='coerce')


valid = N.notna() & LMT.notna() & G_T.notna() & S_So.notna()
N = N[valid].to_numpy()
LMT = LMT[valid].to_numpy()
G_T = G_T[valid].to_numpy()
S_So = S_So[valid].to_numpy()


L = 3.11
Go = 1367


Ts = LMT
Ds = 23.45 * np.sin(np.radians((360 * (N - 81)) / 365))
Hs = 15 * (Ts - 12)
```

```
sin_Alpha = (np.sin(np.radians(L)) * np.sin(np.radians(Ds)) +
               np.cos(np.radians(L)) * np.cos(np.radians(Ds)) * np.cos(np.
                   radians(Hs)))

Gext = Go * (1 + 0.0333 * np.cos(np.radians(360 * N / 365)))
GextH = Gext * sin_Alpha
G_T_G_ext = G_T / GextH


valid_mask = np.isfinite(G_T_G_ext) & np.isfinite(S_So)
G_T_G_ext = G_T_G_ext[valid_mask]
S_So = S_So[valid_mask]


P_Liner = np.polyfit(S_So, G_T_G_ext, 1)
X_Liner = np.linspace(0, 1, 100)
Y_Liner = np.polyval(P_Liner, X_Liner)


plt.figure(figsize=(7, 5))
plt.plot(S_So, G_T_G_ext, 'k.', markersize=3)
plt.plot(X_Liner, Y_Liner, 'r-', linewidth=1.5)
plt.xlabel(r'$(S/S_o)$')
plt.ylabel(r'$G_T/G_{ext}$')
plt.xlim(0, 1)
plt.ylim(0, 1)
plt.grid(True, linestyle='--', alpha=0.5)
plt.tight_layout()
plt.show()
```

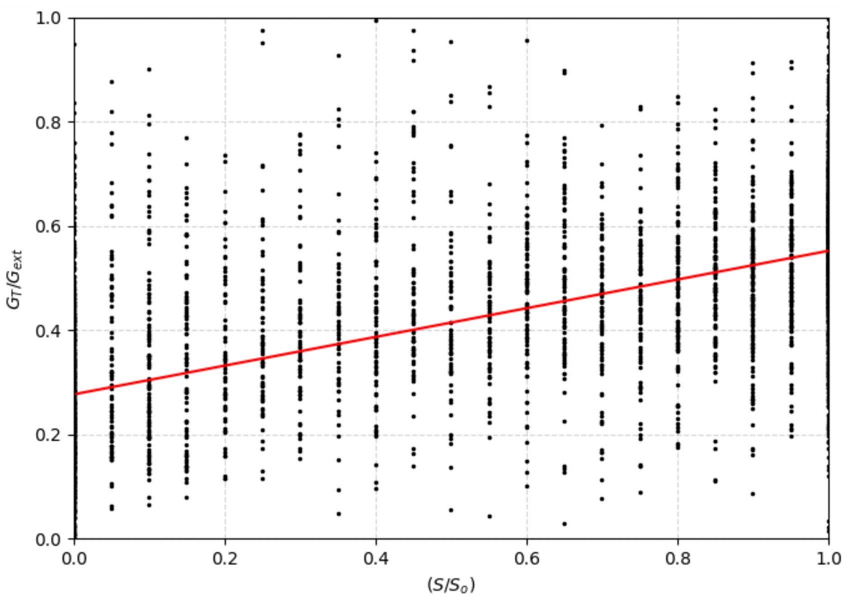


Figure 6: Modeling of global solar radiation on a horizontal surface using linear model.

Result:

$$\frac{G_T}{G_{\mathrm{ex}}} = 0.2743 + 0.2772 \cdot \frac{S}{S_o}$$

On the other hand, some authors have suggested changing the model by adding a nonlinear term to the Angström model as follows:

$$\frac{G_T}{G_{\text{ex}}} = a + b \cdot \frac{S}{S_o} + c\left(\frac{S}{S_o}\right)^2$$

****

Example 1.8: Red o Example 1.7 for diffuse solar radiation.

solution:

```python
#Example 1.8
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt


file_path = 'PV Modeling Book Data Source.csv'
data = pd.read_csv(file_path)

# Extract and ensure numeric conversion for relevant columns
N = pd.to_numeric(data.iloc[:, 2], errors='coerce')
LMT = pd.to_numeric(data.iloc[:, 3], errors='coerce')
G_T = pd.to_numeric(data.iloc[:, 4], errors='coerce')
G_D = pd.to_numeric(data.iloc[:, 5], errors='coerce')


valid = N.notna() & LMT.notna() & G_T.notna() & G_D.notna()
N = N[valid].to_numpy()
LMT = LMT[valid].to_numpy()
G_T = G_T[valid].to_numpy()
G_D = G_D[valid].to_numpy()


L = 3.11
Go = 1367


Ts = LMT
Ds = 23.45 * np.sin(np.radians((360 * (N - 81)) / 365))
Hs = 15 * (Ts - 12)

sin_Alpha = (np.sin(np.radians(L)) * np.sin(np.radians(Ds)) +
             np.cos(np.radians(L)) * np.cos(np.radians(Ds)) * np.cos(np.
                 radians(Hs)))

Gext = Go * (1 + 0.0333 * np.cos(np.radians(360 * N / 365)))
GextH = Gext * sin_Alpha
G_T_G_ext = G_T / GextH
G_D_G_T = G_D / G_T


valid_mask = np.isfinite(G_T_G_ext) & np.isfinite(G_D_G_T)
G_T_G_ext = G_T_G_ext[valid_mask]
G_D_G_T = G_D_G_T[valid_mask]

```

```

P_Liner = np.polyfit(G_T_G_ext, G_D_G_T, 1)
X_Liner = np.linspace(min(G_T_G_ext), max(G_T_G_ext), 100)
Y_Liner = np.polyval(P_Liner, X_Liner)


plt.figure(figsize=(8, 4))
plt.plot(G_T_G_ext, G_D_G_T, 'b.', markersize=3, label=r'$G_D/G_T$ vs. $K_T$')
plt.plot(X_Liner, Y_Liner, 'r-', linewidth=1, label='Linear model')
plt.xlabel(r'$K_T = G_T/G_{ext}$')
plt.ylabel(r'$G_D/G_T$')
plt.xlim(0, 1.2)
plt.ylim(0.2, 1.0)
plt.grid(True, linestyle='--', alpha=0.5)
plt.legend()
plt.tight_layout()
plt.show()
```

Result:

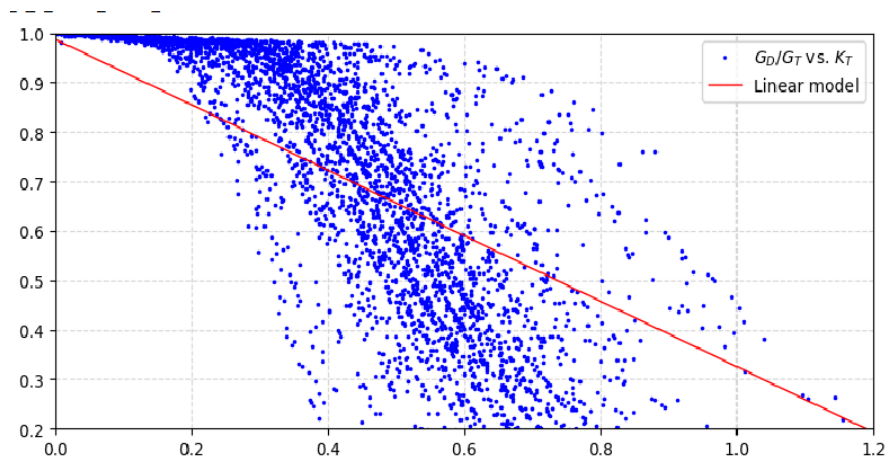


Figure 7: Modeling of diffuse solar radiation on a horizontal surface using linear model.

****

Example 1.9: Develop a FFMLP ANN model that predicts hourly global solar radiation and diffuse solar radiation as illustrated in Figure 1.16(in the book) based on the data provided in the file ”PV Modeling Book Data Source.xls”.

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neural_network import MLPRegressor
from sklearn.preprocessing import StandardScaler
```

```
from sklearn.pipeline import make_pipeline
from scipy.signal import savgol_filter


file_path = 'PV Modeling Book Data Source.xls'
sheet_name = 'Source 1'
data = pd.read_excel(file_path, sheet_name=sheet_name)


G_T = data.iloc[4:2997, 4].values
G_D = data.iloc[4:2997, 5].values
Hum = data.iloc[4:2997, 7].values
T = data.iloc[4:2997, 9].values
S = data.iloc[4:2997, 8].values
M = data.iloc[4:2997, 0].values
D = data.iloc[4:2997, 1].values
H = data.iloc[4:2997, 3].values


G_T_Test = data.iloc[2997:3640, 4].values
G_D_Test = data.iloc[2997:3640, 5].values
Hum_Test = data.iloc[2997:3640, 7].values
T_Test = data.iloc[2997:3640, 9].values
S_Test = data.iloc[2997:3640, 8].values
M_Test = data.iloc[2997:3640, 0].values
D_Test = data.iloc[2997:3640, 1].values
H_Test = data.iloc[2997:3640, 3].values


inputs = np.array([M, D, H, T, Hum, S]).T
targets = np.array([G_T, G_D]).T
test_inputs = np.array([M_Test, D_Test, H_Test, T_Test, Hum_Test, S_Test]).
    T


net = make_pipeline(
    StandardScaler(),
    MLPRegressor(hidden_layer_sizes=(10, 10), max_iter=1000, solver='adam',
        random_state=42)
)


net.fit(inputs, targets)


G_Mi = net.predict(test_inputs)
G_Tp = G_Mi[:, 0]
G_Dp = G_Mi[:, 1]


G_Dp_smooth = savgol_filter(G_Dp[:100], window_length=9, polyorder=2)


x_range = np.arange(0, 100)

plt.figure(figsize=(10, 6))


```

```python
plt.subplot(2, 1, 1)
plt.plot(x_range, G_T_Test[:100], 'b-', linewidth=1.2, label='Actual data')
plt.plot(x_range, G_Tp[:100], 'r-', linewidth=1.2, label='Predicted data')
plt.ylabel('Global solar radiation\n$(W/m^2)$')
plt.xticks(np.arange(0, 101, 10))
plt.xlim(0, 100)
plt.ylim(0, 850)
plt.grid(True)
plt.legend()


plt.subplot(2, 1, 2)
plt.plot(x_range, G_D_Test[:100], 'b-', linewidth=1.2, label='Actual data')
plt.plot(x_range, G_Dp_smooth, 'r-', linewidth=1.2, label='Predicted data')
plt.xlabel('Time (h)')
plt.ylabel('Diffuse solar radiation\n$(W/m^2)$')
plt.xticks(np.arange(0, 101, 10))
plt.xlim(0, 100)
plt.ylim(0, 500)
plt.grid(True)
plt.legend()

plt.tight_layout()
plt.show()
```

Answer:

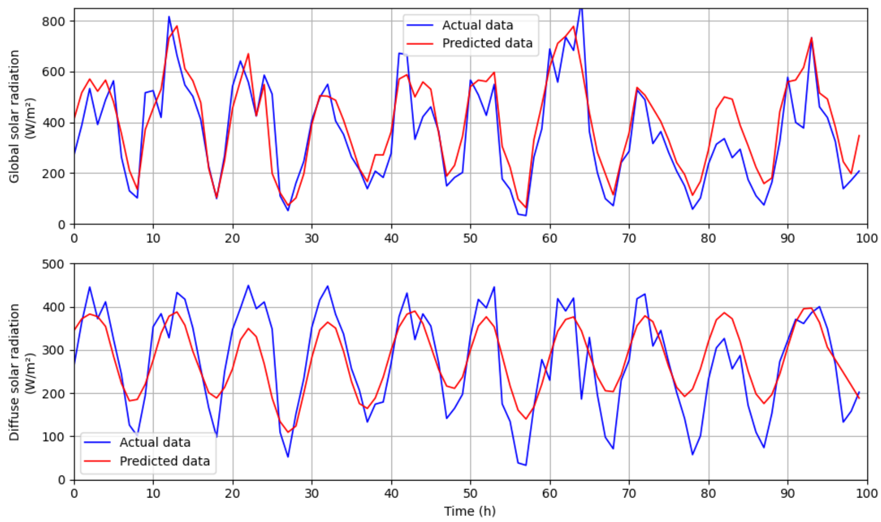


Figure 8: Prediction results of ANN model in Example 1.8

****

Example 1.10: Develop a single axis Sun tracker model using Python that tracks the Sun every 5 min.

Solution:

```python
import numpy as np
import matplotlib.pyplot as plt

L = 3.12  # Latitude for Kuala Lumpur
LOD = 101.7  # Longitude
T_GMT = 8  # Time difference from GMT
Step = 5  # Time step in minutes

BetaT = []


for N in range(1, 5):
    Ds = 23.45 * np.sin(np.radians((360 * (N - 81)) / 365))
    B = (360 * (N - 81)) / 364
    EoT = (9.87 * np.sin(np.radians(2 * B))) - (7.53 * np.cos(np.radians(B))) - (1.5 * np.sin(np.radians(B)))
    Lzt = 15 * T_GMT
    Ts_correction = (-4 * (Lzt - LOD) + EoT) if LOD >= 0 else (4 * (Lzt - LOD) + EoT)

    Wsr_ssi = -np.tan(np.radians(Ds)) * np.tan(np.radians(L))
    Wsrsr_ss = np.degrees(np.arccos(Wsr_ssi))
    ASTsr = abs(((Wsrsr_ss / 15) - 12) * 60)
    ASTss = ((Wsrsr_ss / 15) + 12) * 60
    Tsr = ASTsr + abs(Ts_correction)
    Tss = ASTss + abs(Ts_correction)

    Alpha = []

    for LMT in np.arange(Tsr, Tss + Step, Step):
        Ts = LMT + Ts_correction
        Hs = (15 * (Ts - (12 * 60))) / 60
        sin_Alpha = (np.sin(np.radians(L)) * np.sin(np.radians(Ds)) +
                     np.cos(np.radians(L)) * np.cos(np.radians(Ds)) * np.cos(np.radians(Hs)))
        Alpha_i = np.degrees(np.arcsin(sin_Alpha))
        Alpha.append(Alpha_i)

    Alpha = np.array(Alpha)
    Beta = 90 - Alpha
    BetaT.append(Beta)


BetaT = np.array(BetaT).T


Beta1 = BetaT[:, 0]
Beta2 = BetaT[:, 1]
Beta3 = BetaT[:, 2]
Beta4 = BetaT[:, 3]


fig, axs = plt.subplots(2, 2, figsize=(10, 8))

axs[0, 0].plot(Beta1, color='orange')
axs[0, 0].set_ylim(20, 100)

```

```
axs[0, 1].plot(Beta2, color='orange')
axs[0, 1].set_ylim(20, 100)

axs[1, 0].plot(Beta3, color='orange')
axs[1, 0].set_ylim(20, 100)

axs[1, 1].plot(Beta4, color='orange')
axs[1, 1].set_ylim(20, 100)

for ax in axs.flat:
    ax.set_ylabel('Tilt angle')
    ax.set_xlabel('Time(min)')
    ax.grid(True)

plt.tight_layout()
plt.show()
```

Results:

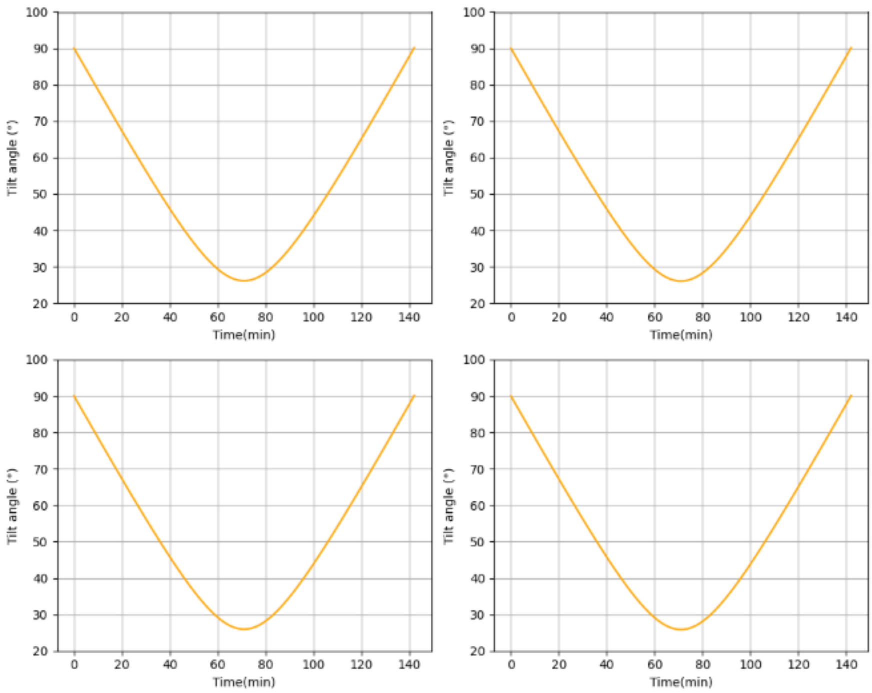


Figure 9: Optimum tilt angle results

.

2 Chapter 2: MODELING OF PHOTOVOLTAIC SOURCE

Example 2.1: Develop a Python code that predicts the I–V and P–V characteristic of the solar cell described in Table 2.1.

TABLE 2.1 PV Module Datasheet

■ Specifications

■ Electrical performance under standard test conditions(*STC)	
Maximum power(Pmax)	200 W(+10%/–5%)
Maximum power voltage (Vmpp)	26.3 V
Maximum power current (Impp)	7.16 A
Open circuit voltage(Voc)	32.9 V
Short circuit current(Isc)	8.21 A
Max system voltage	600 V
Temperature coefficient of Voc	-1.23×10^{-1} V/°C
Temperature coefficient of Isc	3.18×10^{-3} A/°C

*STC: Irradiance 1000 W/m², AM1.5 spectrum, module temperature 25°C

■ Electrical performance at 800 W/m², NOCT, AM1.5	
Maximum power(Pmax)	142 W
Maximum power voltage (Vmpp)	23.2 V
Maximum power current(Impp)	6.13 A
Open circuit voltage(Voc)	29.9 V
Short circuit current(Isc)	6.62 A

Nominal operating cell temperature (NOCT): 47°C

■ Cells	
Number per module	54

■ Module characteristics	
Length × width × depth	1425 mm(56.2in)×990 mm(39.0in)×36 mm(1.4in)
Weight	18.5 kg(40.7 lbs)
Cable	(+)720 mm(28.3in), (–)1800 mm(70.9in)

■ Junction box characteristics	
Length × width × depth	113.6 mm(4.5in)×76 mm(3.0in)×9 mm(0.4in)
IP code	IP65

■ Reduction of efficiency under low Irradiance	
Reduction	7.8%

Reduction of efficiency from an irrandiance of 1000 to 200 W/m² (module temperature 25°C)

From Kyocera 200W manufacturer.

Solution:

```python
import numpy as np
import matplotlib.pyplot as plt

def PV_model(Va, Suns, TaC):

    k = 1.38e-23
    q = 1.60e-19
    A = 1.2
    Vg = 1.12
    Ns = 54


    T1 = 273 + 25
    T2 = 273 + 75
    TaK = 273 + TaC


    Voc_T1 = 32.9 / Ns
    Isc_T1 = 8.21
    Voc_T2 = 29.9 / Ns
    Isc_T2 = 6.62


    Iph_T1 = Isc_T1 * Suns
    a = (Isc_T2 - Isc_T1) / Isc_T1 / (T2 - T1)
    Iph = Iph_T1 * (1 + a * (TaK - T1))

    # Thermal voltage and reverse saturation current
    Vt_T1 = k * T1 / q
    Ir_T1 = Isc_T1 / (np.exp(Voc_T1 / (A * Vt_T1)) - 1)
    Ir_T2 = Isc_T2 / (np.exp(Voc_T2 / (A * Vt_T1)) - 1)
```

```
    b = Vg * q / (A * k)
    Ir = Ir_T1 * (TaK / T1) ** (3 / A) * np.exp(-b * (1 / TaK - 1 / T1))


    X2v = Ir_T1 / (A * Vt_T1) * np.exp(Voc_T1 / (A * Vt_T1))
    dVdI_Voc = -1.15 / Ns / 2
    Rs = -dVdI_Voc - 1 / X2v


    Vt_Ta = A * k * TaK / q
    Vc = Va / Ns
    Ia = np.zeros_like(Vc)


    for _ in range(5):
        exp_term = np.exp((Vc + Ia * Rs) / Vt_Ta)
        Ia = Ia - (Iph - Ia - Ir * (exp_term - 1)) / (-1 - Ir * exp_term *
            Rs / Vt_Ta)

    return Ia


Suns = 1
TaC = 25
Va = np.arange(0, 34, 1)
Ia = PV_model(Va, Suns, TaC)
P = Va * Ia


plt.figure(figsize=(10, 5))
plt.subplot(2, 1, 1)
plt.plot(Va, Ia)
plt.title('Current vs Voltage')

plt.subplot(2, 1, 2)
plt.plot(Va, P)
plt.title('Power vs Voltage')
plt.xlabel('Voltage (V)')
plt.tight_layout()
plt.show()
```

This code must be called from another file as follows:
Suns=1; TaC=25; Va=0:1:33; Ia=Bookexample21(Va,Suns,TaC); P=Va.*Ia;
subplot(2,1,1) plot(Va,Ia) subplot(2,1,2) plot (Va,P)

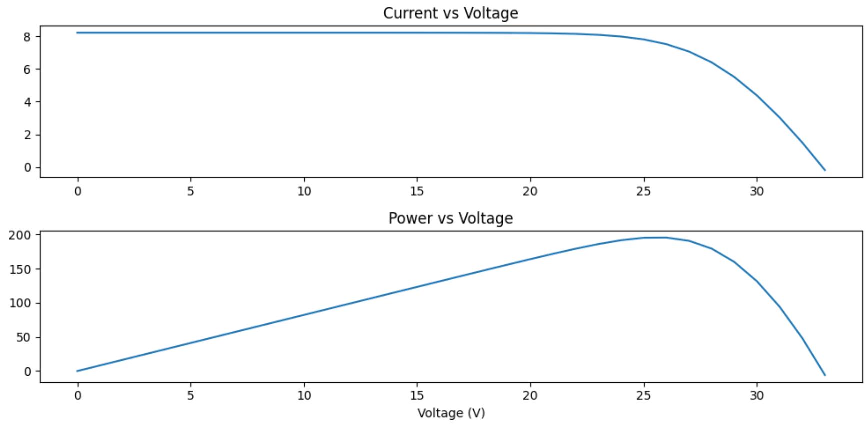


Figure 10: I–V and P–V characteristic PV module at 1000 W/m2 and 25°C

****

Example 2.2: Develop a linear regression model for PV output current provided in book source sheet's name "Source 2."

```

#Example 2.2
import numpy as np
import pandas as pd
from google.colab import files
file_path = '/content/PV Modeling Book Data Source.xls'
sheet_name = 'Source 2'


df = pd.read_excel(file_path, sheet_name=sheetName, usecols="A:C", skiprows
    =1)

G = df.iloc[:, 0].values  # Global solar radiation
Temp = df.iloc[:, 1].values  # Ambient temperature
I_PV = df.iloc[:, 2].values  # PV actual current

# ---------------Modeling of global solar energy--------
N_Liner = 1  # Order of the function
mask = ~np.isnan(G) & ~np.isnan(I_PV) & ~np.isinf(G) & ~np.isinf(I_PV)
G_clean = G[mask]
I_PV_clean = I_PV[mask]
P_Liner = np.polyfit(G_clean, I_PV_clean, N_Liner)


X_Liner = I_PV
Y_Liner = np.zeros_like(X_Liner)

for i in range(1, N_Liner + 2):
    Y_Liner += P_Liner[i - 1] * X_Liner**(N_Liner - i + 1)
```

Answer:

$$I_{PV} = (-0.861) + (0.0075 \times G) + (0.05 \times T) \tag{3}$$

****

Example 2.3: Develop a Python model that compares the aforementioned three models and test the results provided in book data source "Source 2" and compare the results to the empirical and statistical models.

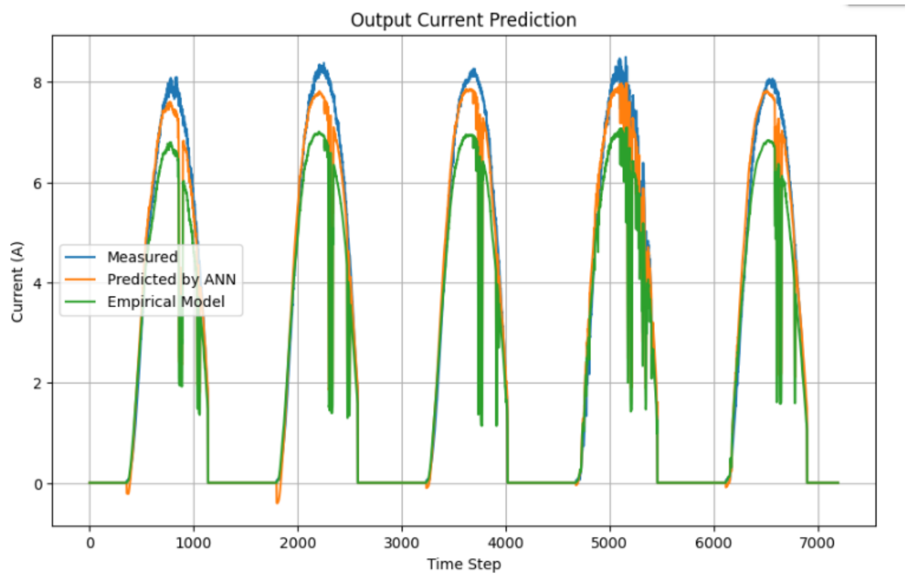
Answer:

```python
# Example 2.3
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.neural_network import MLPRegressor
from sklearn.linear_model import Ridge
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, mean_absolute_percentage_error

# Load Excel data
file_path = '/content/PV Modeling Book Data Source.xls'
sheet_name = 'Source 2'

# Read training data
df = pd.read_excel(file_path, sheet_name='Source 2', usecols="A:C", skiprows=1)

# Remove any rows with NaN values
df.dropna(inplace=True)

G = df.iloc[:, 0].values  # Global solar radiation
Temp = df.iloc[:, 1].values  # Ambient temperature
I_PV = df.iloc[:, 2].values  # PV actual current


# Read test data
df_test = pd.read_excel(file_path, sheet_name=sheet_name, usecols="A:C", skiprows=36002)
G_Test = df_test.iloc[:, 0].values
Temp_Test = df_test.iloc[:, 1].values
I_PV_Test = df_test.iloc[:, 2].values

# Prepare inputs
inputs = np.column_stack((G, Temp))
test_inputs = np.column_stack((G_Test, Temp_Test))

# Ask user for network type
print("Choose the network type:\n1. FFANN\n2. GRNN Simulation")
k = input("Enter your choice (1 or 2): ")

# Scale inputs
scaler = StandardScaler()
inputs_scaled = scaler.fit_transform(inputs)
test_scaled = scaler.transform(test_inputs)
```

```
# Select and train network
if k == "1":
    net = MLPRegressor(hidden_layer_sizes=(5,), solver='lbfgs', max_iter
        =1000, random_state=1)
elif k == "2":
    net = Ridge(alpha=1.0)  # Simulated GRNN
else:
    raise ValueError("Invalid choice")

net.fit(inputs_scaled, I_PV)
C_ANN = net.predict(test_scaled)

# Theoretical current
C_th = (G_Test / 1000) * 7.91

# Regression model (empirical)
C_Reg = -1.17112 + 0.009 * G + 0.055 * Temp

# Plot results
plt.figure(figsize=(10, 6))
plt.plot(I_PV_Test, label='Measured')
plt.plot(C_ANN, label='Predicted by ANN')
plt.plot(C_th, label='Empirical Model')
plt.legend()
plt.title("Output Current Prediction")
plt.xlabel("Time Step")
plt.ylabel("Current (A)")
plt.grid(True)
plt.show()

# Compute evaluation metrics over averaged steps (each hour = 360 samples
    at 10s steps)
steps = 5760
def averaged(arr, step):
    return np.array([np.mean(arr[i:i+step]) for i in range(0, len(arr) -
        step + 1, step)])

AV_C_Test = averaged(I_PV_Test, steps)
AV_C_ANN = averaged(C_ANN, steps)

E3_Hour = AV_C_Test - AV_C_ANN
ANN_MAPE = np.abs(E3_Hour / AV_C_Test)
ANN_meanMAPE = np.mean(ANN_MAPE) * 100
ANN_RMSE = np.sqrt(np.mean((AV_C_ANN - AV_C_Test)**2))
ANN_MBE = np.mean(AV_C_ANN - AV_C_Test)
SUM = np.mean(AV_C_ANN)
ANN_RMSE_Percentage = (ANN_RMSE / SUM) * 100
ANN_MBE_Percentage = (ANN_MBE / SUM) * 100

import pandas as pd


metrics = pd.DataFrame({
    'ANN_meanMAPE(%)': [ANN_meanMAPE],
    'ANN_RMSE': [ANN_RMSE],
    'ANN_MBE': [ANN_MBE],
    'ANN_RMSE_Percentage(%)': [ANN_RMSE_Percentage],
```

```
    'ANN_MBE_Percentage(%)': [ANN_MBE_Percentage]
})
```

Answer:



****

Example 2.4: Predict PV output current in Example 2.3 using RF model and compare it to an ANN-based model.

Answer: RF code starts by defining algorithm's variables such as day number, number of hours per day, ambient temperature, latitude, longitude, and number of PV modules as inputs and PV output current as an output. In the first stage, RF code searched for the most important variables that affect the output. In the meanwhile, the outliers and clusters in the data utilized are detected. The number of trees is assumed to be equal to the half number of the total observations.

```

#exercise 2.4
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.manifold import MDS
import time
import warnings
warnings.filterwarnings('ignore')


fileName = 'PV Modeling Book Data Source.xls'
```

```python
sheetName = 'Source 3'


df = pd.read_excel(fileName, sheet_name=sheetName)

DN = df.iloc[2:1884, 1].values  # Day Number (B3:B1884)
H = df.iloc[2:1884, 2].values   # Hour (C3:C1884)
T = df.iloc[2:1884, 3].values   # Ambient Temp ( C ) (D3:D1884)
S = df.iloc[2:1884, 4].values   # Solar Radiation (E3:E1884)
La = df.iloc[2:1884, 5].values  # Latitude (F3:F1884)
Lo = df.iloc[2:1884, 6].values  # Longitude (G3:G1884)
NPV = df.iloc[2:1884, 7].values # Number of PV Modules (H3:H1884)
I = df.iloc[2:1884, 9].values   # PV DC Current (A) (J3:J1884)

start_time = time.time()
Y = I
X = np.column_stack([DN, H, T, S, La, Lo, NPV])
t = 125

B = RandomForestRegressor(n_estimators=t, oob_score=True, random_state=42)
B.fit(X, Y)


oob_errors = []
for i in range(1, t+1):
    temp_rf = RandomForestRegressor(n_estimators=i, oob_score=True,
       random_state=42)
    temp_rf.fit(X, Y)
    oob_errors.append(1 - temp_rf.oob_score_)

plt.figure(1)
plt.plot(range(1, t+1), oob_errors)
plt.xlabel('Number of Grown Trees')
plt.ylabel('Out-of-Bag Mean Squared Error')
plt.show()


feature_importance = B.feature_importances_
plt.figure(2)
plt.bar(range(1, len(feature_importance)+1), feature_importance)
plt.title('Variable Importance')
plt.xlabel('Variable Number')
plt.ylabel('Out-of-Bag Variable Importance')
plt.legend(['1: Day Number, 2: Hour, 3: Ambient Temp, 4: Solar Radiation,
   5: Latitude, 6: Longitude, 7: # of PV Modules'], loc='upper right')
plt.show()

nidx = np.where(feature_importance < 0.65)[0]


finbag = np.linspace(0.6, 0.9, t)
plt.figure(3)
plt.plot(range(1, t+1), finbag)
plt.xlabel('Number of Grown Trees')
plt.ylabel('Fraction of in-Bag Observations')
plt.show()

predictions = B.predict(X)
```

```
residuals = Y - predictions
outlier_measure = np.abs(residuals)

plt.figure(4)
plt.hist(outlier_measure, bins=30)
plt.title('The Outliers')
plt.xlabel('Outlier Measure')
plt.ylabel('Number of Observations')
plt.show()


mds = MDS(n_components=2, random_state=42)
X_mds = mds.fit_transform(X)

plt.figure(5)
plt.scatter(X_mds[:, 0], X_mds[:, 1], c='k', alpha=0.6)
plt.title('Cluster Analysis')
plt.xlabel('1st Scaled Coordinate')
plt.ylabel('2nd Scaled Coordinate')
plt.show()


eigenvalues = np.random.exponential(scale=2, size=20)
eigenvalues = np.sort(eigenvalues)[::-1]

plt.figure(6)
plt.bar(range(1, 21), eigenvalues)
plt.xlabel('Scaled Coordinate Index')
plt.ylabel('Eigen Value')
plt.show()


filename = 'Data.xlsx'
sheet = 0

df_test = pd.read_excel(filename, sheet_name=sheet)

DN_t = df_test.iloc[1884:2690, 1].values
H_t = df_test.iloc[1884:2690, 2].values
T_t = df_test.iloc[1884:12690, 3].values
S_t = df_test.iloc[1884:2690, 4].values
La_t = df_test.iloc[1884:2690, 5].values
Lo_t = df_test.iloc[1884:2690, 6].values
NPV_t = df_test.iloc[1884:2690, 7].values
I_t = df_test.iloc[1884:2690, 9].values


Xdata = np.column_stack([DN_t, H_t, T_t[:len(DN_t)], S_t, La_t, Lo_t, NPV_t
    ])
Yfit = B.predict(Xdata)

plt.figure(7)
plt.plot(Yfit, label='I Predicted')
plt.plot(I_t, 'red', label='I Actual')
plt.xlabel('Time (H)')
plt.ylabel('Current (A)')
plt.legend(loc='upper right')
plt.title('I Predicted Vs I Actual')
```

```
plt.show()

plt.figure(8)
E = I_t - Yfit
plt.plot(E)
plt.xlabel('Time (H)')
plt.ylabel('Magnitude (A)')
plt.title('Error')
plt.show()

elapsed_time = time.time() - start_time


MBE = np.sum(I_t - Yfit) / len(I_t)

if MBE < 0:
    F = 'Over forecasted'
elif MBE > 0:
    F = 'Under Forecasted'
elif MBE == 0:
    F = 'Ideal Forecasted'


MAPE = (np.abs(np.sum((I_t - Yfit) / I_t)) / len(I_t)) * 100

# Root Mean Square Error (RMSE)
RMSE = np.sum((I_t - Yfit)**2) / len(I_t)

# Outputs
n1 = f'Mean Bias Error (MBE): {MBE}(A) {{Average Deviation Indicator}}'
n2 = f'Forecasting Status: {F}'
n3 = f'Mean Absolute Percentage Error (MAPE): {MAPE}% {{Accuracy Indicator
    }}'
n4 = f'Root Mean Square Error (RMSE): {RMSE}(A) {{Efficiency Indicator}}'

print(n1)
print(n2)
print(n3)
print(n4)


filename = 'Data.xlsx'
sheet = 0

df_train = pd.read_excel(filename, sheet_name=sheet)

DN = df_train.iloc[2:1884, 1].values
H = df_train.iloc[2:1884, 2].values
T = df_train.iloc[2:1884, 3].values
S = df_train.iloc[2:1884, 4].values
I = df_train.iloc[2:1884, 9].values


DN_t = df_train.iloc[1884:2690, 1].values
H_t = df_train.iloc[1884:2690, 2].values
T_t = df_train.iloc[1884:12690, 3].values
S_t = df_train.iloc[1884:2690, 4].values
I_t = df_train.iloc[1884:2690, 9].values
```

```python


Y = I
X = np.column_stack([DN, H, T, S])
Xdata = np.column_stack([DN_t, H_t, T_t[:len(DN_t)], S_t])

MBE_results = []
MAPE_results = []
RMSE_results = []
time_results = []

for t in range(1, 501):
    for l in range(1, 101):
        start_time = time.time()
        B = RandomForestRegressor(n_estimators=t, min_samples_leaf=l,
            oob_score=True, random_state=42)
        B.fit(X, Y)

        # Testing
        Yfit = B.predict(Xdata)
        elapsed = time.time() - start_time
        time_results.append(elapsed)

        E = I_t - Yfit

        # Performance metrics
        MBE_val = np.sum(I_t - Yfit) / len(I_t)
        MBE_results.append(MBE_val)

        if MBE_val < 0:
            F = 'Over forecasted'
        elif MBE_val > 0:
            F = 'Under Forecasted'
        elif MBE_val == 0:
            F = 'Ideal Forecasted'

        MAPE_val = (np.sum(np.abs(E)) / np.sum(I_t)) * 100
        MAPE_results.append(MAPE_val)

        RMSE_val = np.sum((I_t - Yfit)**2) / len(I_t)
        RMSE_results.append(RMSE_val)

filename = 'RF - NTrees - 5.xlsx'
sheet = 0


df_results = pd.read_excel(filename, sheet_name=sheet)

NT = df_results.iloc[1:22, 1].values    # Trees Number
ET = df_results.iloc[1:22, 3].values    # Elapsed time (Sec.)
MBE = df_results.iloc[1:22, 4].values   # Mean Bias Error (MBE) (A)
MAPE = df_results.iloc[1:22, 6].values  # Mean Absolute Percentage Error (
    MAPE) (%)
RMSE = df_results.iloc[1:22, 8].values  # Root Mean Square Error (RMSE) (A)
OOB = df_results.iloc[1:22, 9].values   # Out of Bag (OOB)

for i in range(len(NT)):
    plt.figure(9)
```

```
    plt.plot(NT, RMSE, '*-')
    plt.xlabel('Number of Trees')
    plt.ylabel('Root Mean Squared Error (A)')

    plt.figure(10)
    plt.plot(NT, ET, '*-')
    plt.xlabel('Number of Trees')
    plt.ylabel('Elapsed time (Sec.)')

    plt.figure(11)
    plt.plot(NT, MBE, '*-')
    plt.xlabel('Number of Trees')
    plt.ylabel('Mean Bias Error (MBE) (A)')

    plt.figure(12)
    plt.plot(NT, MAPE, '*-')
    plt.xlabel('Number of Trees')
    plt.ylabel('Mean Absolute Percentage Error (MAPE) (%)')

    plt.figure(13)
    plt.plot(NT, OOB, '*-')
    plt.xlabel('Number of Trees')
    plt.ylabel('Out of Bag (OOB))')

plt.show()

M1, I1 = np.min(RMSE), np.argmin(RMSE)
M2, I2 = np.min(ET), np.argmin(ET)
M3, I3 = np.min(MAPE), np.argmin(MAPE)
M4, I4 = np.min(MBE), np.argmin(MBE)
M5, I5 = np.min(OOB), np.argmin(OOB)

n1 = f'Min. Root Mean Squared Error : {M1}(A) @ index : {I1}'
n2 = f'Min. Elapsed time : {M2}(Sec.) @ index: {I2}'
n3 = f'Min. Mean Absolute Percentage Error (MAPE) : {M3}(%) @ index : {I3}'
n4 = f'Min. Mean Bias Error (MBE) : {M4}(A) @ index : {I4}'
n5 = f'Min. Out of Bag (OOB) data : {M5}(A) @ index : {I5}'

print(n1)
print(n2)
print(n3)
print(n4)
print(n5)

filename = 'Data.xlsx'
sheet = 0


df_final = pd.read_excel(filename, sheet_name=sheet)

DN = df_final.iloc[2:1884, 1].values
H = df_final.iloc[2:1884, 2].values
T = df_final.iloc[2:1884, 3].values
S = df_final.iloc[2:1884, 4].values
I = df_final.iloc[2:1884, 9].values

# RF_Training Code
Y = I
```

```
X = np.column_stack([DN, H, T, S])
t = 65
i = 1

B = RandomForestRegressor(n_estimators=t, min_samples_leaf=i, oob_score=
    True, random_state=42)
B.fit(X, Y)

filename = 'Data.xlsx'
sheet = 0

# Testing Data - Excel File
DN_t = df_final.iloc[1884:2690, 1].values
H_t = df_final.iloc[1884:2690, 2].values
T_t = df_final.iloc[1884:12690, 3].values
S_t = df_final.iloc[1884:2690, 4].values
I_t = df_final.iloc[1884:2690, 9].values

# RF_Testing Code
Xdata = np.column_stack([DN_t, H_t, T_t[:len(DN_t)], S_t])
Yfit = B.predict(Xdata)

plt.figure(14)
plt.plot(Yfit, label='I Predicted')
plt.plot(I_t, 'red', label='I Actual')
plt.xlabel('Time (H)')
plt.ylabel('Current (A)')
plt.legend(loc='upper right')
plt.title('I Predicted Vs I Actual')
plt.show()

plt.figure(15)
E = I_t - Yfit
plt.plot(E)
plt.xlabel('Time (H)')
plt.ylabel('Magnitude (A)')
plt.title('Error')
plt.show()


MBE = np.sum(I_t - Yfit) / len(I_t)

if MBE < 0:
    F = 'Over forecasted'
elif MBE > 0:
    F = 'Under Forecasted'
elif MBE == 0:
    F = 'Ideal Forecasted'

# Mean Absolute Percentage Error (MAPE)
MAPE = (np.abs(np.sum((I_t - Yfit) / I_t)) / len(I_t)) * 100
MAPE = (np.sum(np.abs(E)) / np.sum(I_t)) * 100

# Root Mean Square Error (RMSE)
RMSE = np.sum((I_t - Yfit)**2) / len(I_t)

# Outputs
n1 = f'Mean Bias Error(MBE): {MBE}(A) {{Average Deviation Indicator}}'
```

```
n2 = f'Forecasting Status: {F}'
n3 = f'Mean Absolute Percentage Error (MAPE): {MAPE}% {{Accuracy Indicator
    }}'
n4 = f'Root Mean Square Error (RMSE): {RMSE}(A) {{Efficiency Indicator}}'

print(n1)
print(n2)
print(n3)
print(n4)

# ANN Vs RF
filename = 'Data.xlsx'
sheet = 0


df_comparison = pd.read_excel(filename, sheet_name=sheet)

I = df_comparison.iloc[2:808, 11].values
Iann = df_comparison.iloc[2:808, 13].values
Irf = df_comparison.iloc[2:808, 14].values

plt.plot(I, label='I Actual')
plt.plot(Iann, ':ks', label='I ANNs Model')
plt.plot(Irf, '--ro', label='I Random Forests Model')
plt.xlabel('Time (H)')
plt.ylabel('Current (A)')
plt.legend(loc='upper right')
plt.grid(True)
plt.show()
```

****

Example 2.5: Develop a Python code that optimally characterizes the PV module described in Table 2.1 using DE algorithm and utilize the data provided in book data source “Source 4.”

Answer: To solve this drill, four MATLAB m files are needed; the first one is used to get the value of the five parameters. In this program, solar radiation and cell temperature values are obtained. Furthermore, the experimental current and voltage (for I–V curve) are obtained as well. After that, the second program is called as a MATLAB function to implement the DE algorithm and returns the optimal five-parameter values at specific weather condition. Then the I–V and P–V characteristics for specific weather condition are obtained using NR method. After that, the fitness function of the DE algorithms at each generation is applied.

```

```

```python
#Example 2.5
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import time
import pickle
import os
from openpyxl import Workbook
from openpyxl import load_workbook


def main():
    t = time.time()
    radiation = [978]
    cell_temperature = [328.56]
    sheet = 7


    df = pd.read_excel('PV Modeling Book Data Source.xls', sheet_name='Source 4')

    Ve = df.iloc[2:104, 6].values
    Ie = df.iloc[2:104, 7].values
    Vp = Ve.copy()


    Ve = np.array(Ve, dtype=float)
    Ie = np.array(Ie, dtype=float)
    Vp = np.array(Vp, dtype=float)


    with open('var_fitness_function.pkl', 'wb') as f:
        pickle.dump({'Vp': Vp, 'Ie': Ie}, f)

    solar_radiation = np.array(radiation, dtype=float) / 1000
    G = solar_radiation
    Tc = np.array(cell_temperature, dtype=float)

    f_bestt, a_bestt, Rs_bestt, Rp_bestt, Iph_bestt, Io_bestt = PV_MODELING_BASED_DE_ALGORITHM(G, Tc)

    a_best = a_bestt
    Rs_best = Rs_bestt
    Rp_best = Rp_bestt
    Iph_best = Iph_bestt
    Io_best = Io_bestt
    f_best = f_bestt


    Nsc = 36
    k = 1.3806503e-23
    q = 1.60217646e-19
    VT = (Nsc * k * Tc[0]) / q

    Ip = np.zeros_like(Vp, dtype=float)
    for h in range(5):

        exp_term = np.exp((Vp + Ip * Rs_best) / (a_best * VT))
```

```python
        numerator = Iph_best - Ip - Io_best * (exp_term - 1) - ((Vp + Ip * Rs_best) / Rp_best)
        denominator = -1 - Io_best * (Rs_best / (a_best * VT)) * exp_term - (Rs_best / Rp_best)
        Ip = Ip - (numerator / denominator)

    # Plotting I-V Characteristics
    plt.figure()
    plt.plot(Vp, Ip, label='Calculated')
    plt.plot(Ve, Ie, 'o', label='Experimental')
    plt.title('I-V characteristics of PV module with solar radiation and ambient temperature variations')
    plt.xlabel('Module voltage (V)')
    plt.ylabel('Module current (A)')
    plt.legend()
    plt.text(0.5, 0.5, '978 W/m^2, 328.56 K', transform=plt.gca().transAxes)
    plt.show()

    # Computing PV Module Power
    Pp = Vp * Ip
    Pe = Vp * Ie

    # Plotting P-V Characteristics
    plt.figure()
    plt.plot(Vp, Pp, label='Calculated Power')
    plt.plot(Vp, Pe, 'o', label='Experimental Power')
    plt.title('P-V characteristics of PV module')
    plt.xlabel('Module voltage (V)')
    plt.ylabel('Module power (W)')
    plt.legend()
    plt.text(0.5, 0.5, '978 W/m^2, 328.56 K', transform=plt.gca().transAxes)
    plt.show()


    Iee = np.mean(Ie)
    RMSE = np.sqrt((1/len(Vp)) * np.sum((Ip - Ie)**2))
    MBE = (1/len(Vp)) * np.sum((Ip - Ie)**2)
    RR = 1 - ((np.sum((Ie - Ip)**2)) / (np.sum((Ie - Iee)**2)))

    print(f"RMSE: {RMSE}")
    print(f"MBE: {MBE}")
    print(f"RR: {RR}")

def PV_MODELING_BASED_DE_ALGORITHM(G, Tc):
    EP = 0.054
    D = 5
    Np = 10 * D
    F = 0.85
    CR = 0.6
    GEN_max = 500
    sheet = 7
    file_name = 'Results_of_Radiation_7_temperature_7.pkl'


    params = {'Np': Np, 'GEN_max': GEN_max, 'F': F, 'CR': CR}
    with open(file_name, 'wb') as f:
```

```
        pickle.dump(params, f)


    excel_file = 'PVM4_Try_3.xlsx'
    if not os.path.exists(excel_file):
        wb = Workbook()
        wb.save(excel_file)

    # Write parameters to Excel
    write_to_excel(excel_file, sheet, 'P5', Np)
    write_to_excel(excel_file, sheet, 'Q5', GEN_max)
    write_to_excel(excel_file, sheet, 'R5', F)
    write_to_excel(excel_file, sheet, 'S5', CR)
    write_to_excel(excel_file, sheet, 'U5', EP)

    Rs_l = 0.1;  Rs_h = 2
    Rp_l = 100;  Rp_h = 5000
    a_l = 1;     a_h = 2
    Iph_l = 1;   Iph_h = 8
    Io_l = 1e-12; Io_h = 1e-5

    L = np.array([a_l, Rs_l, Rp_l, Iph_l, Io_l], dtype=float)
    H = np.array([a_h, Rs_h, Rp_h, Iph_h, Io_h], dtype=float)

    rr = 1
    a_average = np.zeros(rr, dtype=float)
    Rs_average = np.zeros(rr, dtype=float)
    Rp_average = np.zeros(rr, dtype=float)
    Iph_average = np.zeros(rr, dtype=float)
    Io_average = np.zeros(rr, dtype=float)
    f_average = np.zeros(rr, dtype=float)

    for b in range(rr):
        x = np.zeros(D, dtype=float)
        pop = np.zeros((D, Np), dtype=float)
        Fit = np.zeros(Np, dtype=float)
        r = np.zeros(3, dtype=int)


        for j in range(Np):
            for i in range(D):
                pop[i, j] = L[i] + (H[i] - L[i]) * np.random.rand()

            a = float(pop[0, j])
            Rs = float(pop[1, j])
            Rp = float(pop[2, j])
            Iph = float(pop[3, j])
            Io = float(pop[4, j])
            f = fitness_function(a, Rs, Rp, Iph, Io, G, Tc)
            Fit[j] = f

        Aa = np.zeros(GEN_max, dtype=float)
        ARs = np.zeros(GEN_max, dtype=float)
        ARp = np.zeros(GEN_max, dtype=float)
        AIph = np.zeros(GEN_max, dtype=float)
        AIo = np.zeros(GEN_max, dtype=float)
        Af = np.zeros(GEN_max, dtype=float)

```

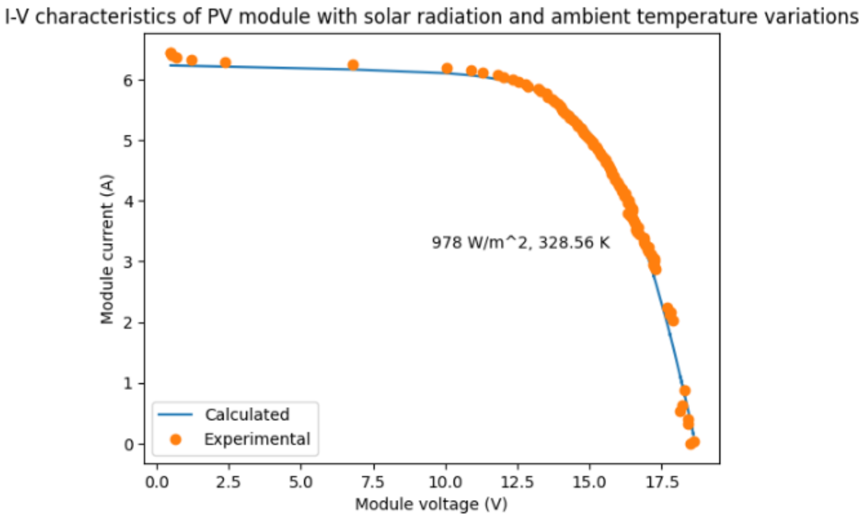
```
        for g in range(GEN_max):
            for j in range(Np):

                r[0] = int(np.floor(np.random.rand() * Np))
                while r[0] == j:
                    r[0] = int(np.floor(np.random.rand() * Np))

                r[1] = int(np.floor(np.random.rand() * Np))
                while r[1] == j or r[1] == r[0]:
                    r[1] = int(np.floor(np.random.rand() * Np))

                r[2] = int(np.floor(np.random.rand() * Np))
                while r[2] == j or r[2] == r[0] or r[2] == r[1]:
                    r[2] = int(np.floor(np.random.rand() * Np))


                w = pop[:, r[2]] + F * (pop[:, r[0]] - pop[:, r[1]])


                Rnd = int(np.floor(np.random.rand() * D))
                for i in range(D):
                    if np.random.rand() < CR or Rnd == i:
                        x[i] = w[i]
                    else:
                        x[i] = pop[i, j]

                # Boundary constraint
                for i in range(D):
                    if x[i] < L[i] or x[i] > H[i]:
                        x[i] = L[i] + (H[i] - L[i]) * np.random.rand()

                # Selection
                a = float(x[0])
                Rs = float(x[1])
                Rp = float(x[2])
                Iph = float(x[3])
                Io = float(x[4])
                f = fitness_function(a, Rs, Rp, Iph, Io, G, Tc)

                if f <= Fit[j]:
                    pop[:, j] = x.copy()
                    Fit[j] = f


            n = np.min(np.abs(Fit))
            iBest = np.argmin(np.abs(Fit))
            Aa[g] = pop[0, iBest]
            ARs[g] = pop[1, iBest]
            ARp[g] = pop[2, iBest]
            AIph[g] = pop[3, iBest]
            AIo[g] = pop[4, iBest]
            Af[g] = Fit[iBest]

            if Fit[iBest] <= EP:
                FEV = g
                break


```

```
        nn = np.min(np.abs(Af))
        Ibest = np.argmin(np.abs(Af))
        a_average[b] = Aa[Ibest]
        Rs_average[b] = ARs[Ibest]
        Rp_average[b] = ARp[Ibest]
        Iph_average[b] = AIph[Ibest]
        Io_average[b] = AIo[Ibest]
        f_average[b] = Af[Ibest]


        AF = Af.reshape(-1, 1)
        write_array_to_excel(excel_file, sheet, 'Z5', AF)


    f_bestt = np.sum(f_average) / rr
    a_bestt = np.sum(a_average) / rr
    Rs_bestt = np.sum(Rs_average) / rr
    Rp_bestt = np.sum(Rp_average) / rr
    Iph_bestt = np.sum(Iph_average) / rr
    Io_bestt = np.sum(Io_average) / rr


    results = {'f_average': f_average}
    with open(file_name, 'wb') as f:
        pickle.dump(results, f)


    write_array_to_excel(excel_file, sheet, 'A108', f_average.reshape(1,
        -1))

    return f_bestt, a_bestt, Rs_bestt, Rp_bestt, Iph_bestt, Io_bestt

def fitness_function(a, Rs, Rp, Iph, Io, G, Tc):
    Nsc = 36
    k = 1.3806503e-23
    q = 1.60217646e-19
    VT = (Nsc * k * Tc[0]) / q


    with open('var_fitness_function.pkl', 'rb') as f:
        data = pickle.load(f)
        Vp = np.array(data['Vp'], dtype=float)
        Ie = np.array(data['Ie'], dtype=float)

    Ip = np.zeros_like(Vp, dtype=float)


    a = float(a)
    Rs = float(Rs)
    Rp = float(Rp)
    Iph = float(Iph)
    Io = float(Io)
    VT = float(VT)

    for h in range(5):

        exp_arg = (Vp + Ip * Rs) / (a * VT)
        exp_term = np.exp(exp_arg)
```

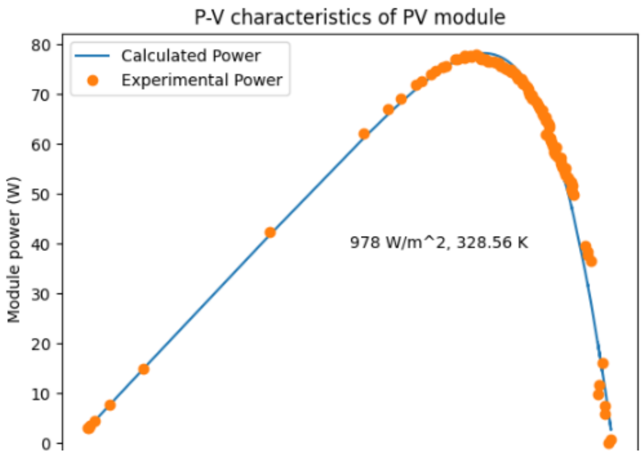
```python

        numerator = Iph - Ip - Io * (exp_term - 1) - ((Vp + Ip * Rs) / Rp)
        denominator = -1 - Io * (Rs / (a * VT)) * exp_term - (Rs / Rp)

        Ip = Ip - (numerator / denominator)

    N = len(Vp)
    f = np.sqrt((1/N) * np.sum((Ie - Ip)**2))

    return float(f)

def write_to_excel(file_path, sheet_num, cell, value):
    """Helper function to write a single value to Excel"""
    try:

        col = ord(cell[0]) - ord('A')
        row = int(cell[1:]) - 1


        try:
            df = pd.read_excel(file_path, sheet_name=f'Sheet{sheet_num}', header=None)
        except:
            df = pd.DataFrame()


        max_row = max(row + 1, len(df))
        max_col = max(col + 1, len(df.columns) if len(df.columns) > 0 else 0)

        if df.empty or df.shape[0] < max_row or df.shape[1] < max_col:
            new_df = pd.DataFrame(index=range(max_row), columns=range(max_col))
            if not df.empty:
                new_df.iloc[:df.shape[0], :df.shape[1]] = df.values
            df = new_df


        df.iloc[row, col] = value


        with pd.ExcelWriter(file_path, engine='openpyxl', mode='a', if_sheet_exists='replace') as writer:
            df.to_excel(writer, sheet_name=f'Sheet{sheet_num}', index=False, header=False)
    except Exception as e:
        print(f"Error writing to Excel: {e}")

def write_array_to_excel(file_path, sheet_num, start_cell, array):
    """Helper function to write an array to Excel"""
    try:

        col = ord(start_cell[0]) - ord('A')
        row = int(start_cell[1:]) - 1


        try:
            df = pd.read_excel(file_path, sheet_name=f'Sheet{sheet_num}',
```

```
                header=None)
        except:
            df = pd.DataFrame()


        if array.ndim == 1:
            array = array.reshape(-1, 1)

        max_row = max(row + array.shape[0], len(df) if not df.empty else 0)
        max_col = max(col + array.shape[1], len(df.columns) if not df.empty and len(df.columns) > 0 else 0)

        if df.empty or df.shape[0] < max_row or df.shape[1] < max_col:
            new_df = pd.DataFrame(index=range(max_row), columns=range(max_col))
            if not df.empty:
                new_df.iloc[:df.shape[0], :df.shape[1]] = df.values
            df = new_df


        df.iloc[row:row+array.shape[0], col:col+array.shape[1]] = array


        with pd.ExcelWriter(file_path, engine='openpyxl', mode='a', if_sheet_exists='replace') as writer:
            df.to_excel(writer, sheet_name=f'Sheet{sheet_num}', index=False, header=False)
    except Exception as e:
        print(f"Error writing array to Excel: {e}")

if __name__ == "__main__":
    main()
```

Answer:



I-V characteristics of PV module with solar radiation and ambient temperature variations
Module current (A)
Module voltage (V)
978 W/m^2, 328.56 K
Calculated
Experimental
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
0
1
2
3
4
5
6



P-V characteristics of PV module
Module power (W)
978 W/m^2, 328.56 K
Calculated Power
Experimental Power
0
10
20
30
40
50
60
70
80

.

3 Chapter 3: MODELING OF PV SYSTEM POWER ELECTRONIC FEATURES AND AUXILIARY POWER SOURCES

Example 3.1: Develop a Python program implementing a P&O-based maximum power point tracker algorithm.

Answer:

```
# Example 3.1
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import interp1d


def PV_model(Va, G, TaC):
    Iph = G * 5
    Io = 1e-10
    n = 1.2
    Vt = 0.025

    # Prevent exponential overflow
    exponent = np.clip(Va / (n * Vt), -700, 700)
    return Iph - Io * (np.exp(exponent) - 1)


TaC = 25             # Cell temperature
C = 0.5              # Step size
Suns_val = 0.028     # (1 G = 1000 W/m^2)
Va = 31              # PV voltage

Ia = PV_model(Va, Suns_val, TaC)
Pa = Ia * Va         # PV output power
Vref_new = Va + C    # New reference voltage

Va_array = []
Pa_array = []

Suns_data = np.array([
    [0, 0.1], [1, 0.2], [2, 0.3], [3, 0.3], [4, 0.5],
    [5, 0.6], [6, 0.7], [7, 0.8], [8, 0.9], [9, 1.0],
    [10, 1.1], [11, 1.2], [12, 1.3], [13, 1.4]
])

x = Suns_data[:, 0]
y = Suns_data[:, 1]
xi = np.arange(1, 201)
interp_func = interp1d(x, y, kind='cubic', fill_value='extrapolate')
yi = interp_func(xi)

for i in range(14):
    Suns = yi[i]
    Va_new = Vref_new
    Ia_new = PV_model(Va, Suns, TaC)
    Pa_new = Va_new * Ia_new
    deltaPa = Pa_new - Pa

    if deltaPa > 0:
        if Va_new > Va:
            Vref_new = Va_new + C
        else:
```

```
            Vref_new = Va_new - C
    elif deltaPa < 0:
        if Va_new > Va:
            Vref_new = Va_new - C
        else:
            Vref_new = Va_new + C
    else:
        V_ref = Va_new

    Va = Va_new
    Pa = Pa_new

    Va_array.append(Va)
    Pa_array.append(Pa)
```

****

Example 3.2: Develop a Python code for implementing an IC-based maximum power point tracker algorithm.

Answer:

```
# Example 3.2
import numpy as np
from scipy.interpolate import PchipInterpolator

TaC = 25
C = 0.5
E = 0.5


Suns_initial = 0.045
Va = 31
def KYOCERA(Va, Suns, TaC):

    return Suns * (0.02 * Va + 0.5)

Ia = KYOCERA(Va, Suns_initial, TaC)
Pa = Va * Ia
Vref_new = Va + C

Va_array = []
Pa_array = []
Pmax_array = []

Suns_matrix = np.array([
    [0, 0.1], [1, 0.2], [2, 0.3], [3, 0.3], [4, 0.5], [5, 0.6], [6, 0.7],
    [7, 0.8], [8, 0.9], [9, 1.0], [10, 1.1], [11, 1.2], [12, 1.3], [13, 1.4]
])

x = Suns_matrix[:, 0]
y = Suns_matrix[:, 1]

xi = np.arange(1, 201)
interp = PchipInterpolator(x, y)
yi = interp(xi)
```

```python

for sample in range(14):

    Suns = yi[sample]


    Va_new = Vref_new
    Ia_new = KYOCERA(Va, Suns, TaC)


    deltaVa = Va_new - Va
    deltaIa = Ia_new - Ia

    if deltaVa == 0:
        if deltaIa == 0:
            Vref_new = Va_new
        elif deltaIa > 0:
            Vref_new = Va_new + C
        else:
            Vref_new = Va_new - C
    else:
        if abs(deltaIa / deltaVa + Ia_new / Va_new) <= E:
            Vref_new = Va_new
        else:
            if deltaIa / deltaVa > -Ia_new / Va_new + E:
                Vref_new = Va_new + C
            else:
                Vref_new = Va_new - C


    Va = Va_new
    Ia = Ia_new
    Pa = Va * Ia

    Va_array.append(Va)
    Pa_array.append(Pa)
```

****

Example 3.3: Develop a Python code for a PWM-based inverter model with a signal of 50 Hz output, 20 percent modulation index, 200 Hz carrier frequency and a load phase angle of 25 °.

Answer:

```python

# Example 3.3

import numpy as np
import matplotlib.pyplot as plt
from scipy.fft import fft

def main():

    print('Voltage-source inverter with Sinusoidal-Pulse Width Modulated
        output')
    print('By Tamer Khatib')
```

```
    print(' ')


    Vrin = 1
    f = float(input('The frequency of the output voltage, f = '))
    Z = 1
    ma = float(input('The modulation index, ma (0 < ma < 1), ma = '))
    phi = float(input('The phase angle of the load in degrees = '))
    fc = float(input('The frequency of the carrier signal = '))


    phi = phi * np.pi / 180
    R = Z * np.cos(phi)
    L = (Z * np.sin(phi)) / (2 * np.pi * f)


    N = int(fc / f)


    wt = np.zeros(2*N*50)
    ma1 = np.zeros(2*N*50)
    Vt = np.zeros(2*N*50)
    Vout = np.zeros(2*N*50)
    alpha = []
    beta = []


    for k in range(1, 2*N+1):
        for j in range(1, 51):
            i = j + (k-1)*50 - 1
            wt[i] = (j + (k-1)*50) * np.pi / (N * 50)


            hpf = np.sign(np.sin(wt[i]))
            if hpf == 0:
                hpf = 1

            ma1[i] = ma * abs(np.sin(wt[i]))

            if k % 2 == 0:
                Vt[i] = 0.02 * j
                if abs(Vt[i] - ma * abs(np.sin(wt[i]))) <= 0.011:
                    m = j
                    beta.append(3.6 * ((k-1)*50 + m) / N)
            else:
                Vt[i] = 1 - 0.02 * j
                if abs(Vt[i] - ma * abs(np.sin(wt[i]))) < 0.011:
                    l = j
                    alpha.append(3.6 * ((k-1)*50 + l) / N)


            if Vt[i] > ma * abs(np.sin(wt[i])):
                Vout[i] = 0
            else:
                Vout[i] = hpf * Vrin


    if beta:
```

```
        beta = beta[1:]


    alpha = np.array(alpha)
    beta = np.array(beta)


    print(' ')
    print('
       ..................................................................')
    print('alpha        beta        width')
    for i in range(min(len(alpha), len(beta))):
        print(f'{alpha[i]:.6f}   {beta[i]:.6f}   {beta[i] - alpha[i]:.6f}')


    a = np.zeros_like(wt)
    plt.figure(1)
    plt.subplot(2, 1, 1)
    plt.plot(wt, Vt, wt, ma1, wt, a)
    plt.axis([0, 2*np.pi, -2, 2])
    plt.ylabel('Vt, m(pu)')

    plt.subplot(2, 1, 2)
    plt.plot(wt, Vout, wt, a)
    plt.axis([0, 2*np.pi, -2, 2])
    plt.ylabel('Vo(pu)')
    plt.xlabel('Radian')
    plt.show()


    Vo = np.sqrt(np.mean(Vout**2))
    print('The RMS value of the output voltage = ')
    print(Vo)

    y = fft(Vout)
    y = y[1:]
    x = np.abs(y)
    x = (np.sqrt(2) / len(Vout)) * x
    print('The RMS value of the fundamental component = ')
    print(x[0])

    THDVo = np.sqrt(Vo**2 - x[0]**2) / x[0]

    m = R / (2 * np.pi * f * L)
    DT = np.pi / (N * 50)
    C = np.zeros(2000*N)
    C[0] = -10


    Vout_extended = np.zeros(2000*N)
    Vout_extended[:len(Vout)] = Vout

    for i in range(100*N, 2000*N):
        Vout_extended[i] = Vout_extended[i - 100*N]

    for i in range(1, 2000*N):
        C[i] = C[i-1] * np.exp(-m * DT) + Vout_extended[i-1] / R * (1 - np.
           exp(-m * DT))
```

```


    CO = np.zeros(100*N)
    for j4 in range(100*N):
        CO[j4] = C[j4 + 1900*N]

    CO2 = fft(CO)
    CO2 = CO2[1:]  # Remove first element
    COX = np.abs(CO2)
    COX = (np.sqrt(2) / (100*N)) * COX

    CORMS = np.sqrt(np.mean(CO**2))
    print('The RMS value of the load current =')
    print(CORMS)

    THDIo = np.sqrt(CORMS**2 - COX[0]**2) / COX[0]


    CS = np.zeros(2000*N)
    for j2 in range(1900*N, 2000*N):
        if Vout_extended[j2] != 0:
            CS[j2] = abs(C[j2])
        else:
            CS[j2] = 0


    CS1 = np.zeros(100*N)
    for j3 in range(100*N):
        CS1[j3] = abs(CS[j3 + 1900*N])

    CSRMS = np.sqrt(np.mean(CS1**2))
    print('The RMS value of the supply current is')
    print(CSRMS)

    CSAV = np.mean(CS1)
    print('The Average value of the supply current is')
    print(CSAV)

    CS2 = fft(CS1)
    CS2 = CS2[1:]
    CSX = np.abs(CS2)
    CSX = (np.sqrt(2) / (100*N)) * CSX


    print('Performance parameters are')
    print('THD of output voltage:')
    print(THDVo)
    print('THD of output current:')
    print(THDIo)

    plt.figure(2, figsize=(12, 10))

    plt.subplot(3, 2, 1)
    plt.plot(wt, Vout[:100*N], wt, a[:100*N])
    plt.title('Output Voltage')
    plt.axis([0, 2*np.pi, -1.5, 1.5])
    plt.ylabel('Vo(pu)')

```

```python
    plt.subplot(3, 2, 2)
    plt.plot(x[:100])
    plt.title('Voltage Spectrum')
    plt.axis([0, 100, 0, 0.8])

    plt.subplot(3, 2, 3)
    plt.plot(wt, C[1900*N:2000*N], wt, a[:100*N])
    plt.title('Output Current')
    plt.axis([0, 2*np.pi, -1.5, 1.5])
    plt.ylabel('Io(pu)')

    plt.subplot(3, 2, 4)
    plt.plot(COX[:100])
    plt.title('Current Spectrum')
    plt.axis([0, 100, 0, 0.8])
    plt.ylabel('Ion(pu)')

    plt.subplot(3, 2, 5)
    plt.plot(wt, CS[1900*N:2000*N], wt, a[:100*N])
    plt.axis([0, 2*np.pi, -1.5, 1.5])
    plt.ylabel('Is(pu)')
    plt.xlabel('Radian')

    plt.subplot(3, 2, 6)
    plt.plot(CSX[:100])
    plt.plot(5, CSAV, '*')
    plt.text(5, CSAV, 'Average value')
    plt.title('Supply Current Spectrum')
    plt.axis([0, 100, 0, 0.8])
    plt.ylabel('Isn(pu)')
    plt.xlabel('Harmonic Order')

    plt.tight_layout()
    plt.show()

if __name__ == "__main__":
    main()
```

****

Example 3.4: Develop a Python code for charging and discharging a battery.

Answer:

```python
# Exercise 3.4

import numpy as np
import matplotlib.pyplot as plt
from sympy import symbols, integrate
from sympy.utilities.lambdify import lambdify


Vbati = []
SOCi = []

# Outer loop
```

```
for I1 in range(5, 6):
    t1 = 7
    SOC1 = 0.2
    K = 0.8
    D = 1e-5
    SOCm = 936
    ns = 6
    SOC2 = SOC1


    for t in np.arange(0, t1 + 0.1, 0.1):
        B = SOC2
        if I1 <= 0:
            V1 = (1.926 + 0.124 * B) * ns
            R1 = (0.19 + 0.1037 / (B - 0.14)) * ns / SOCm
        else:
            V1 = (2 + 0.148 * B) * ns
            R1 = (0.758 + 0.1309 / (1.06 - B)) * ns / SOCm
            R1 = float(R1)

        # Symbolic integration
        v = symbols('v')
        f1 = K * V1 * I1 - D * SOC2 * SOCm
        ee = integrate(f1, (v, 0, t))
        SOC = SOC1 + (1 / SOCm) * ee
        SOC2 = SOC

    Vbat = V1 + I1 * R1
    Vbati.append(float(Vbat))
    SOCi.append(float(SOC))

# Plotting

plt.figure()
plt.plot(Vbati)


plt.figure()
plt.plot(SOCi)
```

****

Example 3.5: Write a Python program that optimizes the tilt angle using the data provided in book data source, "source 5."

Answer: Monthly global solar energy averages on a horizontal surface for chosen sites, and the following equation are used to calculate the global solar radiation on a tilt surface. This equation is based on the Liu and Jordan model for R_B and R_D :

```

# Exercise 3.5

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```

fileName = 'data for 3.5 .xlsx'
sheetName = 'Sheet1'


df = pd.read_excel(fileName, sheet_name=sheetName)


TestdayNumber = df.iloc[5845:6211, 11].values
Gglobal = df.iloc[5845:6211, 4].values
Gdiffused = df.iloc[5845:6211, 5].values

N = TestdayNumber
G = Gglobal
D = Gdiffused


L = 25.0

OptimumB_M = []
data_length = len(G)


days_per_month = data_length // 12

for month in range(1, 13):
    G_M = []
    D_M = []
    N_M = []


    start_idx = (month - 1) * days_per_month
    if month == 12:
        end_idx = data_length
    else:
        end_idx = month * days_per_month

    for j in range(start_idx, end_idx):
        G_M.append(G[j])
        D_M.append(D[j])
        N_M.append(N[j])


    G_M = np.array(G_M)
    D_M = np.array(D_M)
    N_M = np.array(N_M)

    days = len(G_M)
    GBAns = []

    for B in range(0, 91):

        Ds = 23.45 * np.sin((360 * (284 + N_M) / 365) * (np.pi / 180))

        cos_W = -1 * np.tan(L * (np.pi / 180)) * np.tan(Ds * (np.pi / 180))
        cos_W = np.maximum(-1, np.minimum(1, cos_W))
        W = np.arccos(cos_W) * 180 / np.pi

```

```
        denominator = (np.cos(L * (np.pi / 180)) * np.cos(Ds * (np.pi /
            180)) *
                      np.sin(W * (np.pi / 180))) + \
                     ((W * (np.pi / 180)) * np.sin(L * (np.pi / 180)) *
                      np.sin(Ds * (np.pi / 180)))


        denominator = np.where(denominator == 0, np.finfo(float).eps,
            denominator)

        Rb = ((np.cos((L - B) * (np.pi / 180)) * np.cos(Ds * (np.pi / 180))
             *
               np.sin(W * (np.pi / 180)) +
               (W * (np.pi / 180)) * np.sin((L - B) * (np.pi / 180)) *
               np.sin(Ds * (np.pi / 180)))) / denominator

        Rd = (1 + np.cos(B * (np.pi / 180))) / 2
        Rr = (0.3 * (1 - np.cos(B * (np.pi / 180)))) / 2
        F = G_M - D_M
        BB = F * Rb
        DB = D_M * Rd
        RB = G_M * Rr
        GB = BB + DB + RB

        AV_GB = np.mean(GB)
        GBAns.append(AV_GB)

    GBAns = np.array(GBAns)
    MAX_INDEX = np.argmax(GBAns)
    MAX = GBAns[MAX_INDEX]
    maximum_Solar_Radiation = MAX
    optimumB1 = MAX_INDEX
    OptimumB_M.append(optimumB1)

OptimumB_M = np.array(OptimumB_M)
print("OptimumB_M:", OptimumB_M)

Year_Months = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])

# Plotting
plt.figure()
plt.subplot(2, 2, 1)
plt.plot(Year_Months, OptimumB_M, '-kd', linewidth=2.5,
         markeredgecolor='red', markerfacecolor='red', markersize=8)
plt.xlim([1, 12])
plt.legend(['Optimum Tilt angle'], fontsize=8)
plt.xlabel('Month', fontsize=14)
plt.ylabel('Optimum Tilt angle', fontsize=14)
plt.grid(True)
plt.title('Monthly optimum tilt angle', fontsize=14)

plt.gca().tick_params(labelsize=12)
plt.show()
```

****

Example 3.6: Develop a model of a PV water pumping system and implement it in Python. Consider the characteristics of the motor pump as in Tables 3.2 and 3.3 and utilizing the data provided in source 6.

TABLE 3.2 The Characteristics of PMDC Motor

Characteristics	Value
Rated armature current (I)	16.5 A
Rated armature voltage (V)	60 V
Armature resistance (R_a)	0.8 Ω
Rated motor speed (ω)	272.3 rad/s
Motor constant (K_T)	0.175 V/(rad/s)

TABLE 3.3 The Characteristics of Pump

Characteristics	Value
Pumping capacity (Q)	4.8 m^3/h
Maximum pumping head (H)	33 m
Nominal speed (ω)	298.5 rad/s
Nominal required power (P_{pi})	750 W
Inlet impeller radius (R_1)	16.75 mm
Outlet impeller radius (R_2)	80 mm
Inclination angle of impeller blade at impeller inlet (β_1)	38°
Inclination angle of impeller blade at impeller outlet (β_2)	33°
Height of impeller blade at impeller inlet (b_1)	5.4 mm
Height of impeller blade at impeller outlet (b_2)	2.2 mm

Answer:

```

# Example 3.6

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt


file = 'PV Modeling Book Data Source.xls'
sheet = 'Source 6'
rows = list(range(9, 4389))

G = pd.read_excel(file, sheet_name=sheet, usecols="I", skiprows=rows[0]-1,
    nrows=len(rows)).squeeze()
Tc = pd.read_excel(file, sheet_name=sheet, usecols="J", skiprows=rows[0]-1,
     nrows=len(rows)).squeeze()
Vm = pd.read_excel(file, sheet_name=sheet, usecols="L", skiprows=rows[0]-1,
     nrows=len(rows)).squeeze()
Im = pd.read_excel(file, sheet_name=sheet, usecols="M", skiprows=rows[0]-1,
     nrows=len(rows)).squeeze()

Ns = 5
Np = 4
```

```python
Am = 0.9291
A = Ns * Np * Am
Va = Ns * Vm
Ia = Np * Im
Vpv = Va
Ipv = Ia
Pao = Va * Ia
Pai = A * G
effa = np.divide(Pao, Pai, out=np.zeros_like(Pao), where=Pai != 0)

# === DC Motor ===
Va = 0.95 * Va
Ia = 0.9 * Ia
Ra = 0.8
Km = 0.175
Ebb = Va - Ra * Ia
Eb = np.where(Ebb >= 0, Ebb, 0)
Ia = np.where(Ebb >= 0, Ia, 0)
Va = np.where(Ebb >= 0, Va, 0)

Tm = Km * Ia
Tm1 = np.where(Tm == 0, 1, Tm)
Pdev = Eb * Ia

# === Pump Model Constants ===
Rou = 1000
g = 9.81
d1 = 33.5e-3
d2 = 160e-3
beta1 = np.deg2rad(38)
beta2 = np.deg2rad(33)
b1 = 5.4e-3
b2 = 2.2e-3

Kp = Rou * 2 * np.pi * b1 * (d1 / 2)**2 * np.tan(beta1) * ((d2 / 2)**2 - ((b1 * (d1 / 2)**2 * np.tan(beta1)) / (b2 * np.tan(beta2))))
Omega = np.sqrt(np.divide(Km * Ia, Kp, out=np.zeros_like(Ia), where=Kp != 0))
Pmo = Pdev
Pmi = Pao * 0.9
Pmi_safe = np.where(Pmi == 0, 1, Pmi)
effm = Pmo / Pmi_safe

# === Pump ===
Tp = Tm
Eh = Tp * Omega
Ppo = np.where(Ebb >= 0, Eh, 0)
Ppi = Pmo
Ppi_safe = np.where(Ppi == 0, 1, Ppi)
effpp = Ppo / Ppi_safe
effp = np.where(effpp <= 0.95, effpp, 0)

# === Water Flow Rate ===
h1 = 20
h2 = 0.1444
QQ = []
for i in range(len(Eh)):
    r1 = h2 * 2.725
```

```
    r2 = 0
    r3 = h1 * 2.725
    r4 = -Eh[i]
    roots = np.roots([r1, r2, r3, r4])
    real_roots = [r.real for r in roots if np.isreal(r) and r.real > 0]
    QQ.append(real_roots[0] if real_roots else 0)

QQ = np.array(QQ)
Q2 = np.where(QQ == 0, 1, QQ)
H = Eh / (2.725 * Q2)


effsub = effm * effp
effoverall = effa * effsub

Q = np.insert(QQ, 0, 0)
demand = 2.5
Cn = 50
Cr = np.zeros(len(Q))
Qexcess_pv = np.zeros(len(Q))
Qexcess_s = np.zeros(len(Q))
SOC = np.zeros(len(Q))
Qdef_pv = np.zeros(len(Q))
Qdeficit_s = np.zeros(len(Q))

X = Q[1:] - demand
Qdef_pv[1:] = np.where(X < 0, -X, 0)
Qexcess_pv[1:] = np.where(X >= 0, X, 0)

for i in range(1, len(Q)):
    Cr[i] = max(0, Cr[i-1] + X[i-1])
    SOC[i] = min(1, Cr[i] / Cn)
    if SOC[i] >= 1:
        Qexcess_s[i] = Cr[i] - Cn
        Cr[i] = Cn
    Qdeficit_s[i] = max(0, -1 * (Cr[i-1] + X[i-1]))

Q = Q[1:]
Qexcess_pv = Qexcess_pv[1:]
Qexcess_s = Qexcess_s[1:]
Qdeficit_s = Qdeficit_s[1:]
Qdef_pv = Qdef_pv[1:]
Cres = Cr[1:]
SOC = SOC[1:]

D = np.full(len(Q), demand)
LLPh = Qdeficit_s / D
LLP = np.sum(Qdeficit_s) / np.sum(D)

print("Loss of Load Probability (LLP):", LLP)
```

.

4 Chapter 4: MODELING OF PHOTOVOLTAIC SYSTEM ENERGY FLOW

Example 4.1: Develop a Python model for a 2.5 kW, 117 Ah/12 SAPV system utilizing the data in Source 7.

Answer

The first step in writing a Python code for an SAPV is to define the source file and the variables such as hourly solar radiation (G), hourly ambient temperature (T), and the hourly load demand (L). In addition to that, some specification of the system needs to be defined such as the capacity of the PV array, the capacity of the storage battery, inverter rated power, the efficiency the PV module, the allowable depth of charge, the charging efficiency, and the discharging efficiency (Routine 2). The simulation process starts by calculating the produced energy by the PV array, and then the net energy (Enet) is calculated. The maximum state of charge (SOC) of the battery is given to the variable SOCi as an initial value. In addition to that, matrices are defined so as to contain the results of battery state of charge (SOCf), damped energy (Dampf), and energy deficits (Deff) being initiated and defined. At this point a "for loop" is initiated to search the values of the Enet array. Then the energy difference is added to the variable SOCi. Here, if the result SOC is higher than SOCmax, the damped energy is calculated and stored in the "Dampf" array. Meanwhile the ED is set zero and the battery SOC does not change. This condition represents the case of the energy produced by the PV array and is higher than the energy demand. The battery is fully charged from the previous step. The second condition represents the case that the energy produced by the PV array and the battery together is lower than the energy required. Here the battery must stop supplying energy at the defined depth of discharge (DOD) level, while the ED equals the uncovered load demand. In addition to that, the damped energy here is equal to zero. The last condition represents the case that the energy produced by the PV array is lower than the load demand but the battery can cover the remaining load demand. In this case there is no damped nor deficit energy, while the battery SOC equals the difference between the maximum SOC and supplied energy. Finally, battery SOC values and deficit and damped energy values are recalled and the loss of load probability is calculated:

```python
#Exercise 4.1
import numpy as np
```

```
import pandas as pd


file_name = 'PV Modeling Book Data Source.xls'
sheet_name = 'Source 7'


G = pd.read_excel(file_name, sheet_name=sheet_name, usecols="A", nrows=8761)
T = pd.read_excel(file_name, sheet_name=sheet_name, usecols="B", nrows=8761)
L = pd.read_excel(file_name, sheet_name=sheet_name, usecols="D", nrows=8761)

# Convert to numeric values safely
G = pd.to_numeric(G.squeeze(), errors='coerce')
T = pd.to_numeric(T.squeeze(), errors='coerce')
L = pd.to_numeric(L.squeeze(), errors='coerce')


# (2) System specifications
PV_Wp = 2500                    # PV array capacity in Watts
Battery_SOCmax = 1400          # Battery max energy in Wh
PV_eff = 0.16
V_B = 12
Inv_RP = 2500
DOD = 0.8
Charge_eff = 0.8
Alpha = 0.05
Wire_eff = 0.98

SOCmax = Battery_SOCmax
SOCmin = SOCmax * (1 - DOD)

# (3.1) Simulation of the SAPV system
P_Ratio = (PV_Wp * (G / 1000)) / Inv_RP
Inv_eff = 97.644 - (P_Ratio * 1.995) - (0.445 / P_Ratio.replace(0, np.nan))
Inv_eff = Inv_eff.fillna(0)  # Replace NaNs from divide-by-zero with 0

# E_PV = ((PV_Wp*(G/1000)) - (Alpha*(T-25))) * Wire_eff * Inv_eff
E_PV = ((PV_Wp * (G / 1000)) - (Alpha * (T - 25))) * Wire_eff * Inv_eff

E_net = E_PV - L


SOCi = SOCmax
SOCf = []
Deff = []
Dampf = []


for ED in E_net:
    SOC = ED + SOCi

    if SOC > SOCmax:
        Dampi = SOC - SOCmax
        Defi = 0
        SOCi = SOCmax
```

```python

    elif SOC < SOCmin:
        SOCi = SOCmin
        Defi = SOC - SOCmin
        Dampi = 0

    else:
        Defi = 0
        Dampi = 0
        SOCi = SOC

    SOCf.append(SOCi)
    Deff.append(Defi)
    Dampf.append(Dampi)

# outputs
SOCf = np.array(SOCf)
Deff = np.array(Deff)
Dampf = np.array(Dampf)
SOC_per = SOCf / SOCmax
LLP_calculated = abs(np.sum(Deff)) / np.sum(L)

print("Loss of Load Probability (LLP):", LLP_calculated)
```

****

Example 4.2: Develop a Python code for a PV/diesel system utilizing the data in(source) 2500 Wp PV array, 3 kVA diesel generator, and a 580 Ah/12 V battery at 1 percent loss of load probability.

Answer: The system first supplies the load as there is no diesel generator in the system. Meanwhile the diesel generator will be operated in the time that the energy produced by the PV array and the battery is not enough to cover the load demand. The first and second parts of the energy flow model code are like the one for the SAPV system, but the capacity of the used diesel generator (kWh/day) must be added to the first part. After that, a “for loop” is initiated to search the array of the “energy net” (Enet) values. The following part represents the case that the energy generated by the PV array is more than the load demand and consequently there is no generated energy neither by the diesel generator nor the battery. In addition, there is no energy deficit in this case, while the energy to be damped equals to the difference between the energy generated by PV and the load demand. The second case is when the energy generated by the PV array and the battery is less than the energy demand. In this case, the diesel generator must cover the load demand that is not covered by the PV array and the battery. In addition to that, the diesel generator is used to charge the battery. At this point, there are three scenarios:
1. The first is that the diesel generator produced the maximum possible

energy to supply the load and to charge the battery, while the battery state of charge (SOC) is less than or equal maximum SOC.
2. The second is that the battery SOC reaches the maximum value and the diesel generator at this point must stop chagrining the battery.
3. In the third, the diesel generator could not cover all the demanded energy by the load, and it is consequently not able to charge the battery.
Finally, the following code represents the case that the battery is able to cover the load demand alone. Here the diesel generator is used to charge the battery as well. The diesel generator is supposed to keep the battery fully charged to be ready for deficit times. This is because the fact that the use of the energy stored in a battery is easier than operating a diesel generator since the diesel generator needs a start-up time. Moreover, the frequent on/off states of a diesel generator affects its lifetime negatively. However, in this part also the SOC of the battery must be controlled in order not to exceed the allowable SOC. Eventually, four calculated values are stored in arrays. These values are the energy deficits, damped energy, battery SOC, and energy produced by diesel generator. Here also the loss of load probability can be calculated to evaluate the reliability of the designed system:

```

# Exercise 4.2

import numpy as np

# Sample data (replace with real values)
E_net = np.random.uniform(-2000, 2000, size=8760)  # Net energy from PV (Wh)
L = np.random.uniform(1000, 2000, size=8760)       # Load profile (Wh)
SOCmax = 1400                                      # Max battery SOC (Wh)
SOCmin = SOCmax * (1 - 0.8)                        # SOCmin based on DOD = 0.8
E_Capacity = 1500                                  # Diesel generator capacity (Wh/hour)

SOCf = []
Deff = []
Dumpf = []
E_Geni = []

SOCi = SOCmax

for i in range(len(E_net)):
    SOC = E_net[i] + SOCi

    if SOC > SOCmax:
```

```
        Dumpi = SOC - SOCmax
        Defi = 0
        SOCi = SOCmax
        E_Gen = 0

    elif SOC < SOCmin:
        Old_Defi = (SOC - SOCmin) + E_Capacity

        if Old_Defi >= 0:
            SOCi = SOCmin + Old_Defi

            if SOCi <= SOCmax:
                Defi = 0
                Dumpi = 0
                E_Gen = abs(Old_Defi) + (SOCi - SOCmin)
            else:
                Defi = 0
                Dumpi = 0
                E_Gen = abs(Old_Defi) + (SOCi - SOCmin) - (SOCi - SOCmax)
                SOCi = SOCmax
        else:
            SOCi = SOCmin
            Defi = abs(Old_Defi)
            Dumpi = 0
            E_Gen = E_Capacity

    else:
        SOCi = SOC + E_Capacity
        if SOCi <= SOCmax:
            Defi = 0
            Dumpi = 0
            E_Gen = E_Capacity
        else:
            Defi = 0
            Dumpi = 0
            E_Gen = E_Capacity - (SOCi - SOCmax)
            SOCi = SOCmax

    SOCf.append(SOCi)
    Deff.append(Defi)
    Dumpf.append(Dumpi)
    E_Geni.append(E_Gen)

SOCf = np.array(SOCf)
Deff = np.array(Deff)
Dumpf = np.array(Dumpf)
E_Geni = np.array(E_Geni)

SOC_per = SOCf / SOCmax
LLP_calculated = abs(np.sum(Deff)) / np.sum(L)

print("Loss of Load Probability (LLP):", LLP_calculated)
```

****

Example 4.3: Develop a Python code for a PV/diesel/battery system (1.4 kW, 1 kVA, 83 Ah/12 V) considering meteorological data in Source 2 and load demand data in Source 8.

Answer:

```

# Exercise 4.3

import pandas as pd
import matplotlib.pyplot as plt


file_path = 'PV Modeling Book Data Source.xls'
sheet_name = 'Source 2'


df = pd.read_excel(file_path, sheet_name=sheet_name, usecols="B:C",
    skiprows=1, nrows=1440)
I_L = df.iloc[:, 0].values
I_PV = df.iloc[:, 1].values

# System specifications
I_Diesel = 5
SOCmax = 1000
SOCi = SOCmax
V_B = 2
DOD = 0.8
SOCmin = SOCmax * (1 - DOD)
t1 = 1
SOC1 = 1
SOC3 = 0.3
K = 0.8
D = 1e-5
ns = 6
SOC2 = SOC1
W = 0
A = 0.2461
T = 0.081451
n = 0

# Storage
SOCf = []
I_Loadf = []
I_Chargef = []
I_Dischargef = []
I_Batteryf = []
I_Deficitf = []
I_Dampf = []
I_Dieself = []
F_Cf = []

# Simulation
I_net = I_PV - I_L
SOC_track = []

```

```
for i in range(len(I_L)):
    if len(SOCf) == 0:
        prev_SOC = SOC1
    else:
        prev_SOC = SOCf[-1]

    if I_net[i] == 0 and n == 0:
        I_Loadi = I_PV[i]
        I_Dampi = I_Batteryi = I_Chargei = I_Deficiti = I_Dieseli =
            I_Dischargei = F_Ci = 0
        if i == 0 or W == 0:
            SOCi = SOC1
        elif W == 1:
            SOCi = prev_SOC

    elif I_net[i] > 0 and n == 0:
        I_Loadi = I_L[i]
        if i == 0 or W == 0:
            SOCi = SOC1
            I_Dampi = I_net[i]
            I_Chargei = I_Dischargei = I_Batteryi = I_Deficiti = I_Dieseli
                = F_Ci = 0
        elif W == 1:
            if prev_SOC >= SOC1:
                I_Dampi = I_net[i]
                SOCi = prev_SOC
                I_Dischargei = I_Batteryi = I_Deficiti = I_Dieseli =
                    I_Chargei = F_Ci = 0
            else:
                I_Chargei = I_net[i]
                I_Dischargei = I_Batteryi = I_Deficiti = I_Dieseli =
                    I_Dampi = F_Ci = 0
                SOCi = prev_SOC + abs(SOC1 - (SOC1 + I_Chargei * K))

    elif I_net[i] < 0 or (I_net[i] > 0 and n > 0):
        if W == 0 and n == 0:
            I_Dischargei = I_L[i] - I_PV[i]
            I_Loadi = I_PV[i] + I_Dischargei
            I_Batteryi = I_Dischargei
            I_Chargei = I_Deficiti = I_Dampi = I_Dieseli = F_Ci = 0
            SOCi = SOC1 - abs(SOC1 - (SOC1 - I_Dischargei * K))
            W += 1
        elif W == 1:
            if prev_SOC > SOC3 and n == 0:
                I_Dischargei = I_L[i] - I_PV[i]
                I_Loadi = I_L[i]
                I_Batteryi = I_Dischargei
                I_Chargei = I_Deficiti = I_Dampi = I_Dieseli = F_Ci = 0
                SOCi = prev_SOC - abs(SOC1 - (SOC1 - I_Dischargei * K))
            elif prev_SOC <= SOC3 or n > 0:
                if I_Diesel >= I_L[i]:
                    I_Loadi = I_L[i]
                    I_Dieseli = I_Loadi
                    I_Dischargei = I_Deficiti = 0
                    if prev_SOC < SOC1:
                        I_Chargei = I_PV[i]
                        I_Dampi = 0
                        SOCi = prev_SOC + abs(SOC1 - (SOC1 + I_Chargei * K)
```

```
                                 )
                          else:
                              I_Chargei = 0
                              I_Dampi = I_PV[i]
                              SOCi = prev_SOC
                          F_Ci = A * (230 * ((I_Dieseli / 60) / 1000)) + T * (230
                               * ((I_Diesel / 60) / 1000))
                          n += 1
                      elif I_Diesel < I_L[i]:
                          I_Deficiti = I_L[i]

                      if n == 4:
                          n = 0

     SOCf.append(SOCi)
     I_Loadf.append(I_Loadi)
     I_Chargef.append(I_Chargei)
     I_Dischargef.append(I_Dischargei)
     I_Batteryf.append(I_Batteryi)
     I_Deficitf.append(I_Deficiti)
     I_Dampf.append(I_Dampi)
     I_Dieself.append(I_Dieseli)
     F_Cf.append(F_Ci)

# Final results
Excess_energy = ((sum(I_Dampf) / 60) * 220) / 1000
Diesel_consumption = sum(F_Cf)
Enrgy_Deficit = ((sum(I_Deficitf) / 60) * 220) / 1000
Enrgy_Discharge = ((sum(I_Dischargef) / 60) * 220) / 1000

# Plotting
fig1, axs1 = plt.subplots(4, 1, figsize=(10, 8))
axs1[0].plot(I_L, label='Load')
axs1[0].plot(I_PV, label='PV', color='red')
axs1[0].set_ylabel('Current (Amp)')
axs1[0].legend()

axs1[1].plot(I_Dischargef)
axs1[1].set_ylabel('Discharge')

axs1[2].plot(I_Chargef)
axs1[2].set_ylabel('Charge')

axs1[3].plot(SOCf)
axs1[3].set_ylabel('SOC (%)')

fig2, axs2 = plt.subplots(3, 1, figsize=(10, 6))
axs2[0].plot(I_Dieself)
axs2[0].set_ylabel('Diesel')

axs2[1].plot(I_Deficitf, color='red')
axs2[1].set_ylabel('Deficit')

axs2[2].plot(I_Dampf)
axs2[2].set_ylabel('Damp')
axs2[2].set_xlabel('Time (min)')

plt.tight_layout()
```

****

Example 4.4: Redo Example 4.3 considering cycle charge-based operation method.

Answer:

```
#Exercise 4.4

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy import integrate
import sympy as sp

plt.close('all')


fileName = 'Load for Ammar.xls'
sheetName1 = 'Sheet2'
sheetName2 = 'Sheet3'


df1 = pd.read_excel(fileName, sheet_name=sheetName1, usecols='A:B',
    skiprows=1, nrows=1440)
I_PV = df1.iloc[:, 0].values
I_L = df1.iloc[:, 1].values


I_Diesel = 5
SOCmax = 1000
SOCi = SOCmax
V_B = 2
DOD = 0.8
SOCmin = SOCmax * (1 - DOD)
t1 = 1
SOC1 = 1
SOC3 = 0.3
K = 0.8
D = 1e-5
ns = 6
SOC2 = SOC1
W = 0
A = 0.2461
T = 0.081451
n = 0

SOCfinal = []
Dieselfinal = []
SOCf = []
I_Loadf = []
I_Chargef = []
I_Dischargef = []
I_Batteryf = []
I_Deficitf = []
```

```
I_Dampf = []
I_Dieself = []
F_Cf = []

L = len(I_L)
I_net = np.zeros(L)
SOC = np.zeros(L)

for i in range(L):
    I_net[i] = I_PV[i] - I_L[i]

    if I_net[i] == 0:
        if n == 0:
            I_Loadi = I_PV[i]
            I_Dampi = 0
            I_Batteryi = 0
            I_Chargei = 0
            I_Deficiti = 0
            I_Dieseli = 0
            I_Dischargei = 0
            F_Ci = 0

            if i == 0:
                SOCi = SOC1
            elif W == 0:
                SOCi = SOC1
            elif W == 1:
                SOCi = SOCf[i-1]


    elif I_net[i] > 0 and n == 0:
        I_Loadi = I_L[i]

        if i == 0:
            SOCi = SOC1
            I_Dampi = I_net[i]
            I_Chargei = 0
            I_Dischargei = 0
            I_Batteryi = 0
            I_Deficiti = 0
            I_Dieseli = 0
            F_Ci = 0

        elif i > 0:
            if W == 0:
                SOCi = SOC1
                I_Dampi = I_net[i]

            elif W == 1:
                if SOCf[i-1] >= SOC1:
                    I_Dampi = I_net[i]
                    SOCi = SOCf[i-1]
                    I_Dischargei = 0
                    I_Batteryi = 0
                    I_Deficiti = 0
                    I_Dieseli = 0
                    I_Chargei = 0
                    F_Ci = 0
```

```
                elif SOCf[i-1] < SOC1:
                    I_Chargei = I_net[i]
                    I_Dischargei = 0
                    I_Batteryi = 0
                    I_Deficiti = 0
                    I_Dieseli = 0
                    I_Dampi = 0
                    F_Ci = 0

                    for t in [1]:
                        B = SOC2
                        V1 = (2 + 0.148 * B) * ns
                        R1 = (0.758 + 0.1309 / (1.06 - B)) * ns / SOCmax
                        R1 = float(R1)


                        v = sp.Symbol('v')
                        integrand = K * V1 * I_net[i] - D * SOC2 * SOCmax
                        ee = float(sp.integrate(integrand, (v, 0, t)))
                        SOC_temp = SOC1 + (1/SOCmax) * ee
                        SOC2 = SOC_temp

                    SOC2 = float(SOC2)
                    SOC[i] = SOCf[i-1] + abs(SOC1 - SOC2)
                    SOCi = SOC[i]

    elif I_net[i] < 0 or (I_net[i] > 0 and n >= 0):
        if W == 0:
            if n == 0:
                I_Dischargei = I_L[i] - I_PV[i]
                I_Loadi = I_PV[i] + I_Dischargei
                I_Batteryi = I_Dischargei
                I_Chargei = 0
                I_Deficiti = 0
                I_Dampi = 0
                I_Dieseli = 0
                F_Ci = 0

                for t in [1]:
                    B = SOC2
                    V1 = (1.926 + 0.124 * B) * ns
                    R1 = (0.19 + 0.1037 / (B - 0.14)) * ns / SOCmax


                    v = sp.Symbol('v')
                    integrand = K * V1 * I_net[i] - D * SOC2 * SOCmax
                    ee = float(sp.integrate(integrand, (v, 0, t)))
                    SOC_temp = SOC1 + (1/SOCmax) * ee
                    SOC2 = SOC_temp

                SOC2 = float(SOC2)
                SOC[i] = SOC2
                SOCi = SOC[i]
                W = W + 1

        elif W == 1:
            if SOCf[i-1] > SOC3 and n == 0:
```

```
                I_Dischargei = I_L[i] - I_PV[i]
                I_Loadi = I_L[i]
                I_Batteryi = I_Dischargei
                I_Chargei = 0
                I_Deficiti = 0
                I_Dampi = 0
                I_Dieseli = 0
                F_Ci = 0

                for t in [1]:
                    B = SOC2
                    V1 = (1.926 + 0.124 * B) * ns
                    R1 = (0.19 + 0.1037 / (B - 0.14)) * ns / SOCmax


                    v = sp.Symbol('v')
                    integrand = K * V1 * I_net[i] - D * SOC2 * SOCmax
                    ee = float(sp.integrate(integrand, (v, 0, t)))
                    SOC_temp = SOC1 + (1/SOCmax) * ee
                    SOC2 = SOC_temp

                SOC2 = float(SOC2)
                SOC[i] = SOCf[i-1] - abs(SOC1 - SOC2)
                SOCi = SOC[i]

            elif SOCf[i-1] <= SOC3 or n >= 0:
                if I_Diesel >= I_L[i]:
                    I_Loadi = I_L[i]
                    I_Dieseli = I_Diesel
                    I_Dischargei = 0
                    I_Deficiti = 0

                    if SOCf[i-1] < SOC1:
                        I_Chargei = I_PV[i] + I_Dieseli - I_L[i]
                        I_Dampi = 0

                        for t in [1]:
                            B = SOC2
                            V1 = (2 + 0.148 * B) * ns
                            R1 = (0.758 + 0.1309 / (1.06 - B)) * ns / SOCmax
                            R1 = float(R1)


                            v = sp.Symbol('v')
                            integrand = K * V1 * I_Chargei - D * SOC2 * SOCmax
                            ee = float(sp.integrate(integrand, (v, 0, t)))
                            SOC_temp = SOC1 + (1/SOCmax) * ee
                            SOC2 = SOC_temp

                        SOC2 = float(SOC2)
                        SOC[i] = SOCf[i-1] + abs(SOC1 - SOC2)
                        SOCi = SOC[i]
                    else:
                        I_Chargei = 0
                        I_Dampi = I_PV[i] + I_Dieseli - I_L[i]
                        SOCi = SOC1
```

```

                    F_C = A * (220 * ((I_Diesel / 60) / 1000)) + \
                          T * (220 * ((I_Diesel / 60) / 1000))
                    F_Ci = F_C
                    n = n + 1

                elif I_Diesel < I_L[i] and SOCf[i-1] >= 0.8:
                    I_Dischargei = I_L[i] - I_Diesel
                    I_Loadi = I_L[i]
                    I_Batteryi = I_Dischargei
                    I_Chargei = 0
                    I_Deficiti = 0
                    I_Dampi = 0
                    I_Dieseli = I_Diesel
                    F_Ci = 0

                    for t in [1]:
                        B = SOC2
                        V1 = (1.926 + 0.124 * B) * ns
                        R1 = (0.19 + 0.1037 / (B - 0.14)) * ns / SOCmax


                        v = sp.Symbol('v')
                        integrand = K * V1 * I_Dischargei - D * SOC2 * SOCmax
                        ee = float(sp.integrate(integrand, (v, 0, t)))
                        SOC_temp = SOC1 + (1/SOCmax) * ee
                        SOC2 = SOC_temp

                    SOC2 = float(SOC2)
                    SOC[i] = SOCf[i-1] - abs(SOC1 - SOC2)
                    SOCi = SOC[i]

                elif I_Diesel < I_L[i] and SOCf[i-1] <= 0.3:
                    I_Deficiti = I_L[i]

                if SOC[i] >= 0.9:
                    n = 0


    SOCf.append(SOCi)
    I_Loadf.append(I_Loadi)
    I_Chargef.append(I_Chargei)
    I_Dischargef.append(I_Dischargei)
    I_Batteryf.append(I_Batteryi)
    I_Deficitf.append(I_Deficiti)
    I_Dampf.append(I_Dampi)
    I_Dieself.append(I_Dieseli)
    F_Cf.append(F_Ci)


SOCf = np.array(SOCf)
I_Loadf = np.array(I_Loadf)
I_Chargef = np.array(I_Chargef)
I_Dischargef = np.array(I_Dischargef)
I_Batteryf = np.array(I_Batteryf)
I_Deficitf = np.array(I_Deficitf)
I_Dampf = np.array(I_Dampf)
```

```python
I_Dieself = np.array(I_Dieself)
F_Cf = np.array(F_Cf)


DD = SOCf
SSS = np.sum(DD < 0.3)
LL = SSS

Excess_energy = ((np.sum(I_Dampf) / 60) * 220) / 1000
Diesel_consumption = np.sum(F_Cf)                          # Litres
Enrgy_Deficit = ((np.sum(I_Deficitf) / 60) * 220) / 1000   # kWh
Enrgy_Discharge = ((np.sum(I_Dischargef) / 60) * 220) / 1000

# Plotting
plt.figure(figsize=(12, 10))
plt.subplot(4, 1, 1)
plt.plot(I_L, label='Load')
plt.plot(I_PV, 'red', label='PV')
plt.ylabel('Current (Amp)')
plt.legend()

plt.subplot(4, 1, 2)
plt.plot(I_Dischargef)
plt.ylabel('Discharge')

plt.subplot(4, 1, 3)
plt.plot(I_Chargef)
plt.ylabel('Charge')

plt.subplot(4, 1, 4)
plt.plot(SOCf)
plt.ylabel('SOC (%)')

plt.tight_layout()

plt.figure(figsize=(12, 8))
plt.subplot(3, 1, 1)
plt.plot(I_Dieself)
plt.ylabel('Diesel')

plt.subplot(3, 1, 2)
plt.plot(I_Deficitf, 'red')
plt.ylabel('Deficit')

plt.subplot(3, 1, 3)
plt.plot(I_Dampf)
plt.ylabel('Damp')
plt.xlabel('Time (min)')

plt.tight_layout()
plt.show()

print(f"Excess energy: {Excess_energy} kWh")
print(f"Diesel consumption: {Diesel_consumption} Litres")
print(f"Energy Deficit: {Enrgy_Deficit} kWh")
print(f"Energy Discharge: {Enrgy_Discharge} kWh")
print(f"Number of times SOC < 0.3: {LL}")
```

5 Chapter 5: PV SYSTEMS IN THE ELECTRICAL POWER SYSTEM

Example 5.1: Develop a Python code that optimizes inverter size for three PV system sizes of 5 kW, respectively. Model's coefficient for 5 are provided in Table 5.1 (refer to Eq. 3.5). Utilize data in source 2.

TABLE 5.1 Inverter Models Coefficients

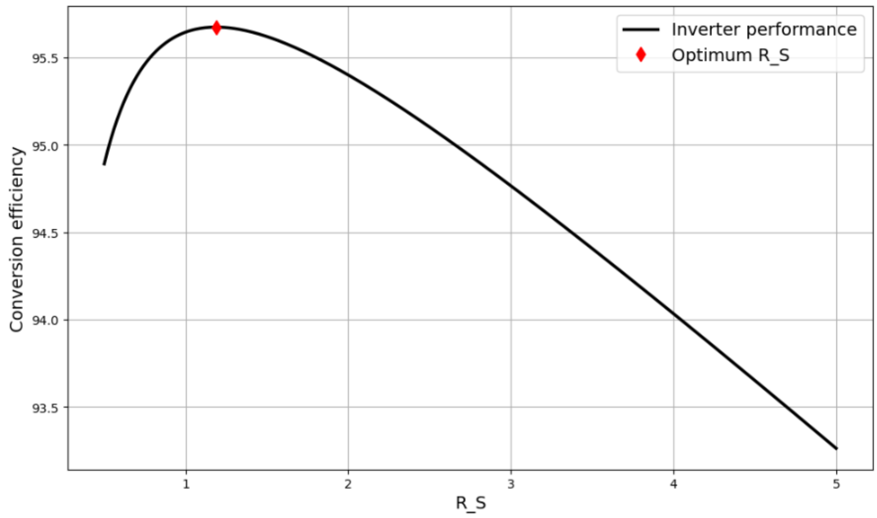
	C_1	C_2	C_3
5 kW	–0.2418	–1.127	96.10

Figure 11: TABLE 5.1 Inverter Models Coefficients

Answer:

```python
# Example 5.1

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Load Excel data
file_name = 'Malaysian Daily Solar Data.xls'
sheet_name = 'Kuala Lumpur'
E_Solar = pd.read_excel(file_name, sheet_name=sheet_name, usecols="E",
    skiprows=7306, nrows=366)
Solar_Rad = (E_Solar.values.flatten() / 12) * 1000

AV_InvEff = []
Rs = []

for Rsi in np.arange(0.5, 5.01, 0.01):
    Rs.append(Rsi)
    Pm = 2
    InvC = Pm / Rsi
    P_Ratio = (Pm * (Solar_Rad / 1000)) / InvC
    InvEffi = 97.644 - (P_Ratio * 1.995) - (0.445 / P_Ratio)

    P = InvEffi[InvEffi >= 0]
    Av = np.sum(P) / len(P)
    AV_InvEff.append(Av)

# Convert to numpy arrays for plotting
Rs = np.array(Rs)
AV_InvEff = np.array(AV_InvEff)

# Plotting
plt.figure(figsize=(10, 6))
plt.plot(Rs, AV_InvEff, '-k', linewidth=2.5)

# Find optimal point
max_eff = np.max(AV_InvEff)
```

```
max_index = np.argmax(AV_InvEff)
opt_rs = 0.5 + max_index * 0.01

plt.plot(opt_rs, max_eff, 'd', markerfacecolor='red', markeredgecolor='red'
    , markersize=8)

plt.xlabel('R_S', fontsize=14)
plt.ylabel('Conversion efficiency', fontsize=14)
plt.legend(['Inverter performance', 'Optimum R_S'], fontsize=14)
plt.grid(True)
plt.tight_layout()
plt.show()
```



6 Chapter 6: PV SYSTEM SIZE OPTIMIZATION

6.2.1.1: Developed Python Code In order to help the readers in practicing similar example, the developed Python code for this problem is shown as follows:

Answer:

```
# Exercise 6.1

import numpy as np
import pandas as pd
from scipy import integrate
import sympy as sp
import warnings
warnings.filterwarnings('ignore')


fileName = 'PV Modeling Book Data Source.xls'
sheetName = 'Source 7'


df = pd.read_excel(fileName, sheet_name=sheetName)
I_L = df.iloc[9:8769, 3].values
T   = df.iloc[9:8769, 1].values
S = df.iloc[9:8769, 0].values

NOCT = 43.6
Alpha = 0.068
IPV_M = 6.89


T_cell = np.zeros(len(S))
I_PV = np.zeros(len(S))

for k in range(len(S)):
    T_cell[k] = (T[k] + (((NOCT - 20) / 800) * S[k]))
    I_PV[k] = ((S[k] / 1000)) + (Alpha * (T_cell[k] - 25))

for zzz in range(len(I_L)):
    if I_PV[zzz] < 0:
        I_PV[zzz] = 0
    else:
        I_PV[zzz] = I_PV[zzz]

k = 0
zzz = 0


PV_eff = 0.14
Wire_eff = 0.98
INV_eff = 0.95
V_sys = 230
PSH = (np.mean(S) * 12) / 1000
V_B = 12
```

```python
DCharge_eff = 0.8
DOD = 0.8
Alpha = 0.068
PV_WP = 120


P_L = np.zeros(len(I_L))
for i in range(len(I_L)):
    P_L[i] = I_L[i] * 230

A_PV = 1.408 * 0.56


E_L = np.sum(I_L * 230)

if np.isnan(E_L) or E_L <= 0:
    print("Error: Invalid energy consumption calculation")
    print(f"E_L = {E_L}")
    print(f"I_L stats: min={np.min(I_L)}, max={np.max(I_L)}, mean={np.mean(
        I_L)}")
    E_L = 1000

if np.isnan(PSH) or PSH <= 0:
    print("Error: Invalid PSH calculation")
    print(f"PSH = {PSH}")
    print(f"S stats: min={np.min(S)}, max={np.max(S)}, mean={np.mean(S)}")
    PSH = 5

N_PV = np.ceil((E_L / (PV_eff * INV_eff * Wire_eff * PSH * 1000 * A_PV)))

if np.isnan(N_PV) or N_PV <= 0:
    print("Error: Invalid N_PV calculation")
    print(f"N_PV = {N_PV}")
    N_PV = 10

N_PVmin = int(np.ceil(N_PV / 3))
N_PVmax = int(5 * N_PV)
nh = 2
C_battery = (E_L * nh) / (DOD * DCharge_eff)
C_batterymin = C_battery / 3
C_batterymax = 5 * C_battery

print(f"E_L = {E_L}")
print(f"PSH = {PSH}")
print(f"N_PV = {N_PV}")
print(f"N_PVmin = {N_PVmin}, N_PVmax = {N_PVmax}")
print(f"C_battery = {C_battery}")
print(f"C_batterymin = {C_batterymin}, C_batterymax = {C_batterymax}")

SOC = []
I_Load = []
I_Charge = []
I_Discharge = []
I_Deficit = []
I_Damp = []
I_Battery = []
t = 1
Vmp = 17.4
```

```
NOCT = 43.6
K = 0.8
D = 1e-5
ns = 6
SOC1 = 1
SOC2 = SOC1
SOC3 = 0.2

C_batteryf = []
C_PVf = []
LLP_calculated = []

x = 0
for m in range(N_PVmin, N_PVmax + 1):
    y = 0
    C_batteryf_row = []
    C_PVf_row = []
    LLP_calculated_row = []

    for n in np.arange(C_batterymin, C_batterymax + 1, 500):
        SOCmax = n
        SOCmin = SOCmax * (1 - DOD)
        w = 0
        SOC = np.zeros(len(I_L))
        I_Load = np.zeros(len(I_L))
        I_Charge = np.zeros(len(I_L))
        I_Discharge = np.zeros(len(I_L))
        I_Deficit = np.zeros(len(I_L))
        I_Damp = np.zeros(len(I_L))
        I_Battery = np.zeros(len(I_L))
        I_net = np.zeros(len(I_L))

        for k in range(len(I_L)):
            I_net[k] = I_PV[k] - I_L[k]


            if SOCmax > 0:

                if I_net[k] == 0:
                    I_Load[k] = I_PV[k]
                    I_Damp[k] = 0
                    I_Charge[k] = 0
                    I_Deficit[k] = 0
                    I_Discharge[k] = 0
                    I_Battery[k] = 0
                    if k == 0:
                        SOC[k] = SOC1
                    elif w == 0:
                        SOC[k] = SOC1
                    elif w == 1:
                        SOC[k] = SOC[k-1]


                elif I_net[k] > 0:
                    I_Load[k] = I_L[k]
                    if k == 0:
                        SOC[k] = SOC1
                        I_Damp[k] = I_net[k]
```

```
                        I_Charge[k] = 0
                        I_Deficit[k] = 0
                        I_Discharge[k] = 0
                        I_Battery[k] = 0
                    elif k > 0:
                        if w == 0:
                            SOC[k] = SOC1
                            I_Damp[k] = I_net[k]
                        elif w == 1:
                            if SOC[k-1] >= SOC1:
                                SOC[k] = SOC[k-1]
                                I_Damp[k] = I_net[k]
                                I_Charge[k] = 0
                                I_Deficit[k] = 0
                                I_Discharge[k] = 0
                                I_Battery[k] = 0
                            elif SOC[k-1] < SOC1:
                                I_Damp[k] = 0
                                I_Charge[k] = I_net[k]
                                I_Deficit[k] = 0
                                I_Discharge[k] = 0
                                I_Battery[k] = 0

                                B = SOC2
                                V1 = (2 + (0.148 * B)) * ns
                                R1 = float(((0.758 + (0.1309 / (1.06 - B))) * ns) / SOCmax)
                                v = sp.Symbol('v')
                                ee = float(sp.integrate(((K * V1 * I_net[k]) - (D * SOC2 * SOCmax)), (v, 0, t)))
                                SOCx = SOC1 + (SOCmax**-1) * ee
                                SOC2 = SOCx
                                SOC2 = float(SOCx)
                                SOC[k] = SOC[k-1] + abs(SOC1 - SOC2)


                            I_Damp[k] = 0
                            I_Charge[k] = 0
                            I_Deficit[k] = 0
                            I_Discharge[k] = I_L[k] - I_PV[k]
                            I_Battery[k] = I_Battery[k]
                            I_Load[k] = I_L[k]
                            B = SOC2
                            V1 = (1.926 + 0.124 * B) * ns
                            R1 = (0.19 + (0.1037 / (B - 0.14))) * (ns / SOCmax)
                            v = sp.Symbol('v')
                            ee = float(sp.integrate(((K * V1 * I_net[k]) - (D * SOC2 * SOCmax)), (v, 0, t)))
                            SOCx = SOC1 + (SOCmax**-1) * ee
                            SOC2 = SOCx
                            SOC2 = float(SOCx)
                            SOC[k] = SOC[k-1] - abs(SOC1 - SOC2)
                        elif SOC[k-1] <= SOC3:
                            SOC2 = SOC3
                            SOC[k] = SOC3
                            I_Damp[k] = 0
                            I_Charge[k] = 0
```

```
                            I_Deficit[k] = I_net[k]
                            I_Discharge[k] = 0
                            I_Battery[k] = 0
                            I_Load[k] = 0


            else:

                if I_net[k] == 0:
                    SOC[k] = SOC1
                    I_Load[k] = I_PV[k]
                    I_Damp[k] = 0
                    I_Charge[k] = 0
                    I_Deficit[k] = 0
                    I_Discharge[k] = 0
                    I_Battery[k] = 0

                elif I_net[k] > 0:
                    SOC[k] = SOC1
                    I_Load[k] = I_L[k]
                    I_Damp[k] = I_net[k]
                    I_Charge[k] = 0
                    I_Deficit[k] = 0
                    I_Discharge[k] = 0
                    I_Battery[k] = 0

            LLP_ff.append(LLP_calculated[ii, jj])
            C_PV_ff.append(C_PVf[ii, jj])
            C_battery_ff.append(C_batteryf[ii, jj])
            cc = cc + 1

if len(C_PV_ff) == 0:
    print('No combinations found with LLP between 0.0095 and 0.0105')
    print(f'Minimum LLP found: {np.min(LLP_calculated)}')
    print(f'Maximum LLP found: {np.max(LLP_calculated)}')


    idx = np.argmin(np.abs(LLP_calculated - 0.01))
    row, col = np.unravel_index(idx, LLP_calculated.shape)

    LLP_ff = [LLP_calculated[row, col]]
    C_PV_ff = [C_PVf[row, col]]
    C_battery_ff = [C_batteryf[row, col]]

    print(f'Using closest LLP value: {LLP_ff[0]}')


LLP_ff = np.array(LLP_ff)
C_PV_ff = np.array(C_PV_ff)
C_battery_ff = np.array(C_battery_ff)



CC_PV = 456
MC_PV = 6.5
Ls = 25
L_PV = 25

```

```
# Battery
Ca_battery = 1200
CC_batwh = 4.8
CC_bat = CC_batwh * Ca_battery
MC_bat = 3.4
L_bat = 5
Y_bat = (Ls / L_bat) - 1
B_rep = 50
CC_B_rep = Y_bat * B_rep

# Charge Controller
CC_cc = 400
MC_cc = 0
L_cc = 25
Y_cc = (Ls / L_cc) - 1
N_cc = 1

# Inverter
CC_inv = 800
MC_inv = 0
L_inv = 25
Y_inv = (Ls / L_inv) - 1
N_inv = 1


N_CB = 4
C_CB = 25
CC_CB = N_CB * C_CB

CC_SS = 200

CC_CW = 400

CC_OC = CC_CB + CC_SS + CC_CW + CC_B_rep
ir = 0.035
fr = 0.015
ndr = ((1 + ir) / (1 + fr)) - 1

# Total life cycle cost calculation
LCCx = np.zeros(len(C_PV_ff))
CC_D = np.zeros(len(C_PV_ff))
LCC = np.zeros(len(C_PV_ff))

for kk in range(len(C_PV_ff)):
    LCCx[kk] = (CC_OC / ((((1 + ndr)**Ls) - 1) / ((ndr * ((1 + ndr)**Ls))))
        ) + ((C_PV_ff[kk] * (CC_PV + Ls * MC_PV)) / L_PV) + (((np.ceil(
        C_battery_ff[kk] / Ca_battery) * CC_bat * (1 + Y_bat)) + (MC_bat * (
        Ls - Y_bat))) / L_bat) + (((N_cc * CC_cc * (1 + Y_cc)) + (MC_cc * (
        Ls - Y_cc))) / L_cc) + (((N_inv * CC_inv * (1 + Y_inv)) + (MC_inv *
        (Ls - Y_inv))) / L_inv)
    CC_D[kk] = (C_PV_ff[kk] * (CC_PV)) + ((np.ceil(C_battery_ff[kk] /
        Ca_battery) * CC_bat * (1 + Y_bat))) + (N_cc * CC_cc * (1 + Y_cc)) +
         (N_inv * CC_inv * (1 + Y_inv)) + CC_CB + CC_SS
    LCC[kk] = LCCx[kk] - (((0.13 * (CC_D[kk]))) / ((((1 + ndr)**Ls) - 1) /
        ((ndr * ((1 + ndr)**Ls)))))

MM = np.min(LCC)
II = np.argmin(LCC)
```

```python
C_PV_best = C_PV_ff[II]
C_battery_best = C_battery_ff[II]

print(f'Best PV Modules Number is: {int(C_PV_best)}')
print(f'Best Battery Capacity is: {int(C_battery_best)}')
```

****

Design space for the proposed hybrid PV/wind/diesel system subject to 1 percent LLP:

Answer:

```python
# Exercise 6.2

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Load Excel file
fileName = 'PV Modeling Book Data Source.xls'
sheetName = 'Source 7'
llp = 0.01
LOAD = 1

# (1) Assumptions
PV_eff = 0.16
Inv_eff = 0.90
V_B = 12
DOD = 0.8
Charge_eff = 0.8
Wp_Cost = 3.7
Ah_Cost = 2.5
kWp_Cost = 1200

# (3) Energy sources size
SE = pd.read_excel(fileName, sheet_name=sheetName, usecols='K', skiprows=1,
     nrows=365).values.flatten()
WS = pd.read_excel(fileName, sheet_name=sheetName, usecols='J', skiprows=1,
     nrows=365).values.flatten()

C_A = []
C_W = []
C_D = []
LLP = []
C_B = []

for PV_A in np.arange(0, 1.05, 0.05):
    PV_E = 0.17 * SE * PV_A * 0.9
    C_Ai = (PV_A * 0.17 * 4.9 * 0.9) / LOAD

    for R in np.arange(0, 1.05, 0.05):
        Air_Density = 1.22521
```

```
        W_Ei = (Air_Density * 3.14 * (R ** 2) * 0.5 * 24 * (WS ** 3)) /
            1000
        W_E = W_Ei * 0.2
        C_Wi = np.sum(W_E) / (LOAD * 365)

        for E_diesel in np.arange(0, 1.05, 0.05):
            C_Di = E_diesel / LOAD
            C_D.append(C_Di)
            C_A.append(C_Ai)
            C_W.append(C_Wi)

            E_T = PV_E + W_E + E_diesel
            E_NET = E_T - LOAD

            EN = E_NET[E_NET < 0]
            EP = E_NET[E_NET >= 0]

            C_Bi = (np.sum(EP) * 0.8 / 365) / LOAD
            C_B.append(C_Bi)

            LLPi = (-1 * np.sum(EN)) / (LOAD * 365)
            LLP.append(LLPi)

C_A = np.array(C_A)
C_W = np.array(C_W)
C_D = np.array(C_D)
C_B = np.array(C_B)
LLP = np.array(LLP)

Array = np.column_stack((C_A, C_W, C_D, C_B, LLP))

llp_modified1 = llp + 0.0003
llp_modified2 = llp - 0.0003

Array_New = []

for j in range(len(Array)):
    if llp_modified2 <= Array[j, 4] <= llp_modified1:
        Array_New.append(Array[j])

Array_New = np.array(Array_New)

if Array_New.shape[0] > 0:
    T = np.arange(0, len(Array_New))
    x_1 = Array_New[T, 0]
    x_2 = Array_New[T, 1]
    x_3 = Array_New[T, 2]
    x_4 = Array_New[T, 3]
    x_5 = x_1 + x_2 + x_3

    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    ax.plot3D(x_1, x_2, x_3, 'b.')
    ax.set_xlabel('PV')
    ax.set_ylabel('Wind')
    ax.set_zlabel('Diesel')
    plt.title('PV-Wind-Diesel Combination')
    plt.show()
```

```python
else:
    print("No configurations found within LLP range.")
```

****

6.4 PV PUMPING SYSTEM SIZE OPTIMIZATION

Answer:

```python
# Exercise 6.3

import numpy as np
import pandas as pd
from numpy import pi, tan, abs, sqrt, real, imag
import warnings

warnings.filterwarnings("ignore")

fileName = 'PV Modeling Book Data Source.xls'
sheetName = 'Source 7'


G = pd.read_excel(fileName, sheet_name=7, usecols='I', skiprows=9, nrows=4380).values.flatten()
Tc = pd.read_excel(fileName, sheet_name=7, usecols='J', skiprows=9, nrows=4380).values.flatten()
Vm = pd.read_excel(fileName, sheet_name=7, usecols='L', skiprows=9, nrows=4380).values.flatten()
Im = pd.read_excel(fileName, sheet_name=7, usecols='M', skiprows=9, nrows=4380).values.flatten()

CNs = np.zeros(80000)
CNp = np.zeros(80000)
CCn = np.zeros(80000)
CLLP = np.ones(80000) * -0.5
CQexcess = np.ones(80000) * -0.5
CQdeficit = np.ones(80000) * -0.5
CQ = np.ones(80000) * -0.5

q = 0

for Cn in range(1, 3):
    for N in range(6, 8):
        for Ns in range(1, N + 1):
            if N % Ns == 0:
                Np = N // Ns
                h1 = 20
                h2 = 0.1444

                Va = Ns * Vm
                Ia = Np * Im
                Vpv = Va
                Ipv = Ia
                Pao = Va * Ia
                Am = 0.9291
                A = Ns * Np * Am
                Pai = A * G
                with np.errstate(divide='ignore', invalid='ignore'):
```

```
            effa = np.nan_to_num(Pao / Pai)

        Va = 0.95 * Va
        Ia = 0.90 * Ia
        Ra = 0.8
        Km = 0.175
        Ebb = Va - Ra * Ia
        Eb = (Ebb >= 0) * Ebb
        Ia = (Ebb >= 0) * Ia
        Va = (Ebb >= 0) * Va
        Tm = Km * Ia
        Tmm = (Tm == 0) * 1
        Tm1 = Tmm + Tm

        Rou = 1000
        d1 = 33.5 * 0.001
        d2 = 160 * 0.001
        beta1 = 38 * 2 * pi / 360
        beta2 = 33 * 2 * pi / 360
        b1 = 5.4 * 0.001
        b2 = 2.2 * 0.001
        Kp = Rou * 2 * pi * b1 * (d1 / 2) ** 2 * tan(beta1) * ((d2 / 2) ** 2 - ((b1 * (d1 / 2) ** 2 * tan(beta1)) / (b2 * tan(beta2))))

        Pdev = Eb * Ia
        with np.errstate(divide='ignore', invalid='ignore'):
            Omega = np.nan_to_num(sqrt((Km * Ia) / Kp))
        Pmo = Pdev
        Pmi = Pao * 0.9
        PMI1 = (Pmi == 0) * 1
        PMI2 = Pmi + PMI1
        with np.errstate(divide='ignore', invalid='ignore'):
            effm = np.nan_to_num(Pmo / PMI2)

        Tp = Tm
        Eh = Tp * Omega
        Ppo = Eh
        Ppo = (Ebb >= 0) * Ppo
        Ppi = Pmo
        PPI1 = (Ppi == 0) * 1
        PPI2 = Ppi + PPI1
        with np.errstate(divide='ignore', invalid='ignore'):
            effpp = np.nan_to_num(Ppo / PPI2)
            effp = (effpp <= 0.95) * effpp

        QQ = np.zeros(len(Eh))
        for ii in range(len(Eh)):
            coeffs = [h2 * 2.725, 0, h1 * 2.725, -Eh[ii]]
            r = np.roots(coeffs)
            QQQ = 0
            for root in r:
                if imag(root) == 0 and real(root) > 0:
                    QQQ = real(root)
                    break
            QQ[ii] = QQQ

        Q1 = (QQ == 0) * 1
```

```
                Q2 = QQ + Q1
                with np.errstate(divide='ignore', invalid='ignore'):
                    H = np.nan_to_num(Eh / (2.725 * Q2))

                effsub = effm * effp
                effoverall = effa * effm * effp
                QQ = (Ebb >= 0) * QQ
                QQ = (effpp <= 0.95) * QQ
                Q = np.concatenate(([0], QQ))
                d = 2.5
                Cr = np.zeros(len(Q))
                Qexcess_pv = np.zeros(len(Q))
                Qexcess_s = np.zeros(len(Q))
                SOC = np.zeros(len(Q))
                Qdef_pv = np.zeros(len(Q))
                Qdeficit_s = np.zeros(len(Q))
                X = Q[1:] - d

                Qdef_pv[1:] = (X < 0) * abs(X)
                Qexcess_pv[1:] = (X >= 0) * abs(X)

                C = len(Q) - 1
                for i in range(C):
                    Cr[i+1] = ((Cr[i] + X[i]) >= 0) * abs(Cr[i] + X[i])
                    SOC[i+1] = Cr[i+1] / Cn
                    if SOC[i+1] >= 1:
                        SOC[i+1] = 1
                        Qexcess_s[i+1] = Cr[i+1] - Cn
                        Cr[i+1] = Cn
                    else:
                        Qexcess_s[i+1] = 0
                    Qdeficit_s[i+1] = ((Cr[i] + X[i]) < 0) * abs(Cr[i] + X[
                        i])

                Q = Q[1:]
                sumQ = np.sum(Q)
                Qexcess = np.sum(Qexcess_s[1:])
                Qdeficit = np.sum(Qdeficit_s[1:])
                D = np.full(len(Q), d)
                with np.errstate(divide='ignore', invalid='ignore'):
                    LLP = np.sum(Qdeficit_s[1:]) / np.sum(D)

                if LLP <= 0.01:
                    CNs[q] = Ns
                    CNp[q] = Np
                    CCn[q] = Cn
                    CLLP[q] = LLP
                    CQexcess[q] = Qexcess
                    CQdeficit[q] = Qdeficit
                    CQ[q] = sumQ
                    q += 1

CNs_result = CNs[:q]
CNp_result = CNp[:q]
CCn_result = CCn[:q]
CLLP_result = CLLP[:q]
CQ_result = CQ[:q]

```

```
CNs_result, CNp_result, CCn_result, CLLP_result, CQ_result
```

.

Reference Book:

MODELING OF PHOTOVOLTAIC SYSTEMS USING MATLAB®

Tamer Khatib, Wilfried Elmenreich