

(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

1 Chapter1: MODELING OF THE SOLAR SOURCE	5
Example 1.1	6
Example 1.2	7
Example 1.3	10
Example 1.4	12
Example 1.5	13
Example 1.6	15
Example 1.7	18
Example 1.8	20
Example 1.9	21
Example 1.10	23
2 Chapter 2: MODELING OF PHOTOVOLTAIC SOURCE	26
Example 2.1	27
Example 2.2	29
Example 2.3	30
Example 2.4	32
Example 2.5	39
3 Chapter 3: MODELING OF PV SYSTEM POWER ELECTRONIC FEATURES AND AUXILIARY POWER SOURCES	48
Example 3.1	49
Example 3.2	50
Example 3.3	51
Example 3.4	55
Example 3.5	56
Example 3.6	58
4 Chapter 4: MODELING OF PHOTOVOLTAIC SYSTEM ENERGY FLOW	62

Example 4.1	63
Example 4.2	65
Example 4.3	67
Example 4.4	71
5 Chapter 5: PV SYSTEMS IN THE ELECTRICAL POWER SYSTEM	77
Example 5.1	78
6 Chapter 6: PV SYSTEM SIZE OPTIMIZATION	80
Example 6.1	81
Example 6.2	87
Example 6.3	89

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.

```
1 import math
2 import numpy as np
3
4 # Location Kuala Lumpur, Malaysia, L = (3.12), LOD = (101.7)
5 L = 3.12
6 LOD = 101.7
7 N = 183
8 T_GMT = 8
9 LMT_minutes = 792 # LMT in minutes (13:12 = 792 minutes)
10
11
12 Ds = 23.45 * math.sin((360 * (N - 81) / 365) * (math.pi / 180))
13
14
15 B = (360 * (N - 81)) / 364
16 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))
17
18
19 Lzt = 15 * T_GMT
20
21
22 if LOD >= 0:
23     Ts_correction = (-4 * (Lzt - LOD)) + EoT
24 else:
25     Ts_correction = (4 * (Lzt - LOD)) + EoT
26
27 # Solar time
28 Ts = LMT_minutes + Ts_correction
29
30 # Hour angle degree
31 Hs = (15 * (Ts - (12 * 60))) / 60
32
```

```

33
34 sin_Alpha = (math.sin(L * math.pi / 180) * math.sin(Ds * math.pi / 180)) +
    \
35         (math.cos(L * math.pi / 180) * math.cos(Ds * math.pi / 180) *
            math.cos(Hs * math.pi / 180))
36 Alpha = math.degrees(math.asin(sin_Alpha))
37
38 # Azimuth angle calculation
39 Sin_Theta = (math.cos(Ds * math.pi / 180) * math.sin(Hs * math.pi / 180)) /
    math.cos(Alpha * math.pi / 180)
40 Theta = math.degrees(math.asin(Sin_Theta))
41
42
43 print(f"Altitude angle (Alpha): {Alpha:.2f} degrees")
44 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

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

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

$$\frac{\omega_{s, sr}}{15^\circ} \pm 12 \text{ h} = \text{AST}_{s, sr} \quad (2)$$

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.

```

1 import math
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5
6
7 L = 3.12
8 LOD = 101.7
9 N = 183
10 T_GMT = 8
11 Step = 5
12
13 Ds = 23.45 * math.sin((360 * (N - 81) / 365) * (math.pi / 180))
14 Lzt = 15 * T_GMT
15
16 B = (360 * (N - 81)) / 364
17 EoT = (9.87 * math.sin(2 * B * math.pi / 180)) - \
18       (7.53 * math.cos(B * math.pi / 180)) - \
19       (1.5 * math.sin(B * math.pi / 180))
20
21 if LOD >= 0:
22     Ts_correction = (-4 * (Lzt - LOD)) + EoT
23 else:
24     Ts_correction = (4 * (Lzt - LOD)) + EoT
25
26 Wsr_ssi = -math.tan(Ds * math.pi / 180) * math.tan(L * math.pi / 180)
27 Wsrssr_ss = math.degrees(math.acos(Wsr_ssi))
28
29 ASTsr = abs(((Wsrssr_ss / 15) - 12) * 60)

```



```

30 ASTss = ((Wsrsr_ss / 15) + 12) * 60
31
32 Tsr = ASTsr + abs(Ts_correction)
33 Tss = ASTss + abs(Ts_correction)
34
35 Alpha = []
36 Theta = []
37 Time = []
38
39 LMT = Tsr
40 while LMT <= Tss:
41     Ts = LMT + Ts_correction
42     Hs = (15 * (Ts - (12 * 60))) / 60
43
44     sin_Alpha = (math.sin(L * math.pi / 180) * math.sin(Ds * math.pi / 180)
45                 ) + \
46                 (math.cos(L * math.pi / 180) * math.cos(Ds * math.pi / 180)
47                 * math.cos(Hs * math.pi / 180))
48
49     Alpha_i = math.degrees(math.asin(sin_Alpha))
50     Alpha.append(Alpha_i)
51
52     Sin_Theta = (math.cos(Ds * math.pi / 180) * math.sin(Hs * math.pi /
53                 180)) / math.cos(Alpha_i * math.pi / 180)
54     Theta_i = math.degrees(math.asin(Sin_Theta))
55     Theta.append(Theta_i)
56
57     Time.append(LMT)
58     LMT += Step
59
60 plt.figure(figsize=(10, 6))
61
62 plt.subplot(2, 1, 1)
63 plt.plot(Time, Alpha, label='Altitude (Alpha)')
64 plt.xlabel('Time (min)')
65 plt.ylabel('Angle (degree)')
66 plt.title('Altitude Angle vs Time')
67 plt.grid(True)
68 plt.xlim(400, 1200)
69
70 plt.subplot(2, 1, 2)
71 plt.plot(Time, Theta, color='red', label='Azimuth (Theta)')
72 plt.xlabel('Time (min)')
73 plt.ylabel('Angle (degree)')
74 plt.title('Azimuth Angle vs Time')
75 plt.grid(True)
76 plt.xlim(400, 1200)
77
78 plt.tight_layout()
79 plt.show()

```

results:

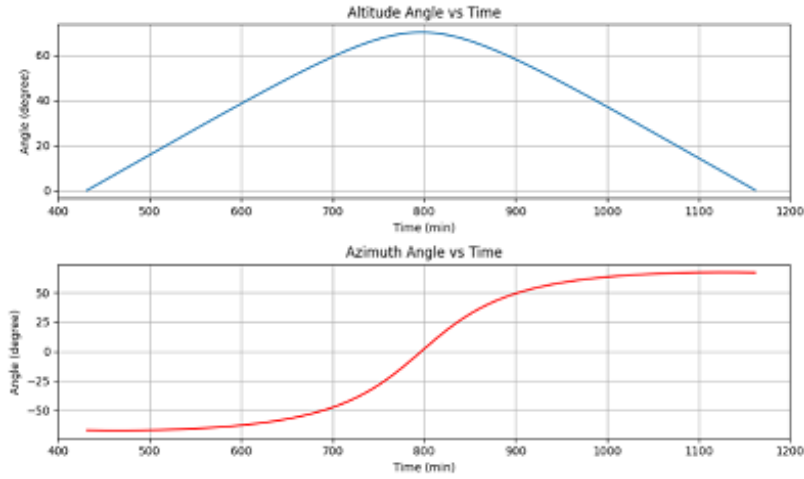


Figure 1: A day's profile of the Sun's altitude and azimuth angles (Example 1.2).

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:

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3
4
5 T = 288
6 E_lambda = []
7
8 for lamda in range(1, 61): # from 1 to 60
9     E_lambda_i = (3.74 * 10e8) / (lamda**5 * (np.exp(14400 / (lamda * T)) -
10         1))
11     E_lambda.append(E_lambda_i)
12
13 lamda_values = list(range(1, 61))
14
15 plt.plot(lamda_values, E_lambda)
16 plt.xlabel('Wavelength [~$\mu$.m]')
```

```

17 plt.ylabel('Intensity [W/m2 μm]')
18 plt.title('Spectral Energy Distribution')
19 plt.grid(True)
20 plt.show()

```

results:

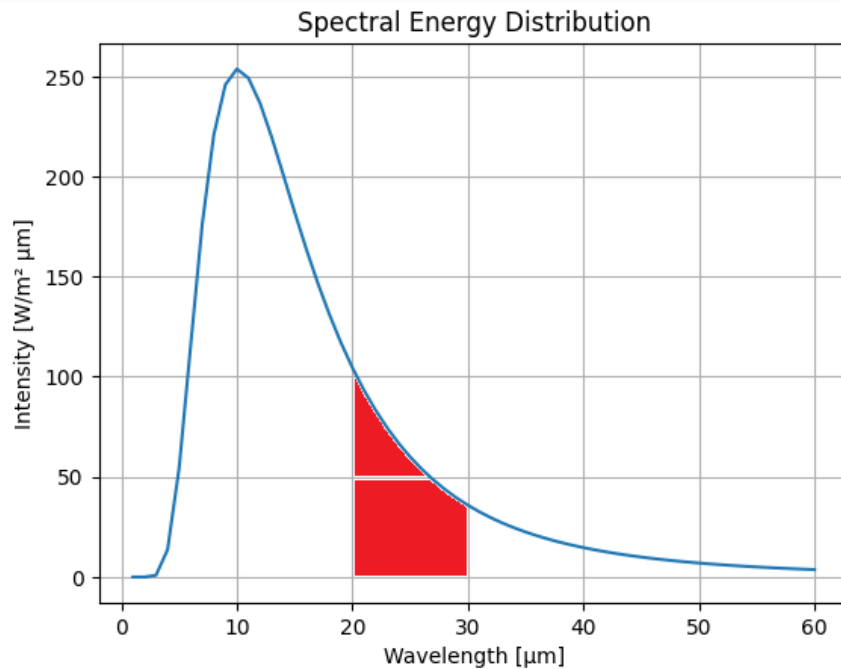


Figure 2: Spectral emissive power of a 288 K blackbody, for wavelengths in the range of (1–60) μm

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:

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

```

1  import numpy as np
2
3
4  T = 288
5  E_lambda = []
6
7  for lamda in range(20, 31): # from 20 to 30 inclusive
8      E_lambda_i = (3.74 * 10e8) / (lamda**5 * (np.exp(14400 / (lamda * T)) -
9          1))
10     E_lambda.append(E_lambda_i)
11
12 E_lambda = np.array(E_lambda)
13
14 Power = np.sum(E_lambda)
15 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:

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

```
1
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5
6
7 L = 32.22
8 LOD = 35.27
9 N = 90
10 T_GMT = +3
11 Step = 60
12
13
14 Ds = 23.45 * np.sin((360 * (N - 81) / 365) * (np.pi / 180))
15
16 # Equation of time
17 B = (360 * (N - 81)) / 364
18 EoT = (9.87 * np.sin(2 * B * np.pi / 180)) - \
19       (7.53 * np.cos(B * np.pi / 180)) - \
20       (1.5 * np.sin(B * np.pi / 180))
21
22 Lzt = 15 * T_GMT
23 Ts_correction = (-4 * (Lzt - LOD) + EoT) if LOD >= 0 else (4 * (Lzt - LOD)
24                   + EoT)
25
26 Wsr_ssi = -np.tan(np.radians(Ds)) * np.tan(np.radians(L))
27 Wsr_ssr_ss = np.degrees(np.arccos(Wsr_ssi))
28
29 ASTsr = abs(((Wsr_ssr_ss / 15) - 12) * 60)
30 ASTss = ((Wsr_ssr_ss / 15) + 12) * 60
31
32 Tsr = ASTsr + abs(Ts_correction)
33 Tss = ASTss + abs(Ts_correction)
34
35
36 LMT = np.arange(Tsr, Tss + Step, Step)
37 sin_Alpha = []
38
39 for t in LMT:
```

```

40 Ts = t + Ts_correction
41 Hs = (15 * (Ts - 12 * 60)) / 60
42 sin_Alpha_i = (np.sin(np.radians(L)) * np.sin(np.radians(Ds)) +
43               np.cos(np.radians(L)) * np.cos(np.radians(Ds)) * np.cos(
44                 np.radians(Hs)))
45 sin_Alpha.append(sin_Alpha_i)
46 sin_Alpha = np.array(sin_Alpha)
47
48
49 Go = 1367~W/m^2
50
51 Gext = Go * (1 + 0.0333 * np.cos(np.radians(360 * N / 365)))
52 GextH = Gext * sin_Alpha
53
54
55 plt.figure(figsize=(8, 5))
56 plt.plot(LMT, GextH, color='gray', linestyle='--', linewidth=1)
57 plt.scatter(LMT, GextH, color='black', s=10)
58 plt.xlabel('Time (min)', fontsize=12)
59 plt.ylabel(r'Solar radiation (W/m$^2$)', fontsize=12)
60 plt.xlim(400, 1200)
61 plt.ylim(100, 1250)
62 plt.grid(True)
63 plt.tight_layout()
64 plt.show()

```

Result:

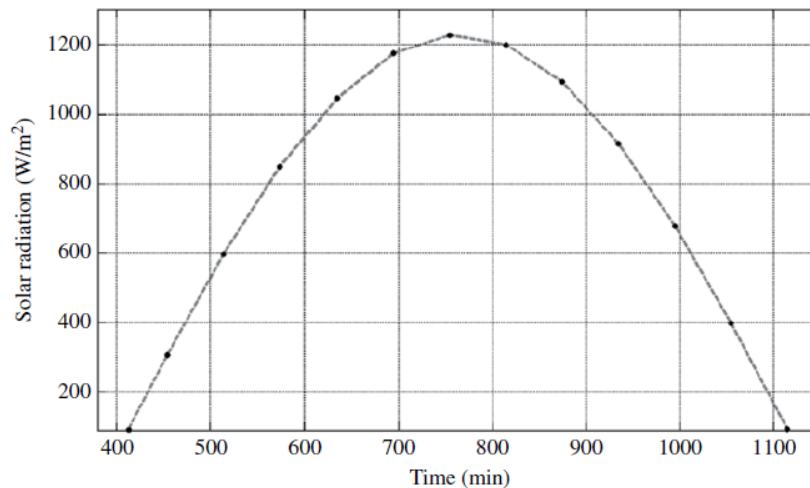


Figure 3: Daily extraterrestrial solar radiation for Nablus city

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 calcu-

lation 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:

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 # Location: Kuwait City
5 L = 29.36
6 LOD = 47.97
7 N = 122
8 T_GMT = +3
9 Step = 60
10
11
12 Ds = 23.45 * np.sin(np.radians((360 * (N - 81)) / 365))
13
14
15 B = (360 * (N - 81)) / 364
16 EoT = (9.87 * np.sin(np.radians(2 * B))) - \
17       (7.53 * np.cos(np.radians(B))) - \
18       (1.5 * np.sin(np.radians(B)))
19
20
21 Lzt = 15 * T_GMT
22 Ts_correction = (-4 * (Lzt - LOD)) + EoT if LOD >= 0 else (4 * (Lzt - LOD))
23   + EoT
24
25 Wsr_ssi = -np.tan(np.radians(Ds)) * np.tan(np.radians(L))
26 Wsr_ssr_ss = np.degrees(np.arccos(Wsr_ssi))
27
28
29 ASTsr = abs(((Wsr_ssr_ss / 15) - 12) * 60)
30 ASTss = ((Wsr_ssr_ss / 15) + 12) * 60
31 Tsr = ASTsr + abs(Ts_correction)
32 Tss = ASTss + abs(Ts_correction)
33
34
35 LMT = np.arange(Tsr, Tss + Step, Step)
36 sin_Alpha = []
37
38 for t in LMT:
39     Ts = t + Ts_correction
40     Hs = (15 * (Ts - (12 * 60))) / 60
41     sin_Alpha_i = (np.sin(np.radians(L)) * np.sin(np.radians(Ds)) +
42                   np.cos(np.radians(L)) * np.cos(np.radians(Ds)) * np.cos(
43                       np.radians(Hs)))
44     sin_Alpha.append(sin_Alpha_i)
45
46 sin_Alpha = np.array(sin_Alpha)
47
48 A = 1160 + (75 * np.sin(np.radians((360 / 365) * (N - 275))))

```

```

49 k = 0.174 + (0.035 * np.sin(np.radians((360 / 365) * (N - 100))))
50 C = 0.095 + (0.04 * np.sin(np.radians((360 / 365) * (N - 100))))
51 G_B_norm = A * np.exp(-k / sin_Alpha)
52 G_B = G_B_norm * sin_Alpha
53 G_D = C * G_B_norm
54 G_T = G_B + G_D
55
56
57 valid_mask = sin_Alpha > 0.1
58 LMT_valid = LMT[valid_mask]
59 G_T_valid = G_T[valid_mask]
60
61
62 G_A = np.array([0.000, 0.2431, 0.4422, 0.5966, 0.865, 0.976,
63                 1.031, 1.016, 0.936, 0.788, 0.5904, 0.3541, 0.1439]) * 1e3
64 G_A = G_A[:len(LMT_valid)]
65
66
67 plt.figure(figsize=(9, 5))
68 plt.plot(LMT_valid, G_T_valid, 'k--o', markersize=4, label='Developed
    method')
69 plt.plot(LMT_valid, G_A, 'k-^', markersize=4, label='Actual data')
70 plt.xlabel('Time (min)', fontsize=12)
71 plt.ylabel(r'Global solar radiation (W/m$^2$)', fontsize=12)
72 plt.xlim(350, 1100)
73 plt.ylim(100, 1050)
74 plt.grid(True)
75 plt.legend()
76 plt.tight_layout()
77 plt.show()

```

Results:

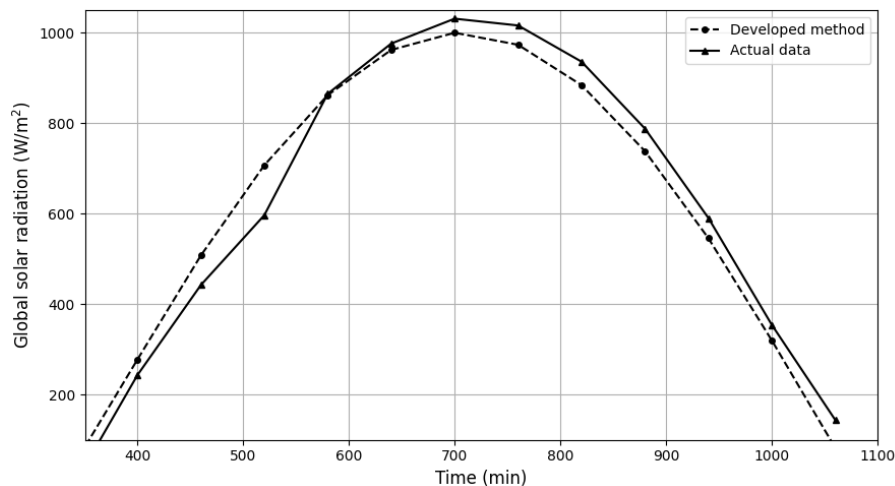


Figure 4: Global solar radiation for Kuwait City

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:

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4
5 L = 29.36
6 LOD = 47.97
7 N = 90
8 T_GMT = +3
9 Step = 60
10
11
12 Ds = 23.45 * np.sin(np.radians((360 * (N - 81)) / 365))
13
14
15 B = (360 * (N - 81)) / 364
16 EoT = (9.87 * np.sin(np.radians(2 * B))) - \
17       (7.53 * np.cos(np.radians(B))) - \
18       (1.5 * np.sin(np.radians(B)))
19
20 # Local Meridian Standard Time
21 Lzt = 15 * T_GMT
22 Ts_correction = (-4 * (Lzt - LOD)) + EoT if LOD >= 0 else (4 * (Lzt - LOD))
23     + EoT
24
25
26 Wsr_ssi = -np.tan(np.radians(Ds)) * np.tan(np.radians(L))
27 Wsr_rsr_ss = np.degrees(np.arccos(Wsr_ssi))
28
29 ASTsr = abs(((Wsr_rsr_ss / 15) - 12) * 60)
30 ASTss = ((Wsr_rsr_ss / 15) + 12) * 60
31 Tsr = ASTsr + abs(Ts_correction)
32 Tss = ASTss + abs(Ts_correction)
33
34 LMT = np.arange(Tsr, Tss, Step)
35 sin_Alpha = []
36 for t in LMT:
37     Ts = t + Ts_correction
38     Hs = (15 * (Ts - 12 * 60)) / 60
39     sin_Alpha_i = (np.sin(np.radians(L)) * np.sin(np.radians(Ds)) +
40                   np.cos(np.radians(L)) * np.cos(np.radians(Ds)) * np.cos(
41                       np.radians(Hs)))
42     sin_Alpha.append(sin_Alpha_i)
43
44 sin_Alpha = np.array(sin_Alpha)
45
46 A = 1160 + (75 * np.sin(np.radians((360 / 365) * (N - 275))))
47 k = 0.174 + (0.035 * np.sin(np.radians((360 / 365) * (N - 100))))
48 C = 0.095 + (0.04 * np.sin(np.radians((360 / 365) * (N - 100))))
```



```

49
50 G_B_norm = A * np.exp(-k / sin_Alpha)
51 G_B = G_B_norm * sin_Alpha
52 G_D = C * G_B_norm
53 G_T = G_B + G_D
54
55
56 Beta = L
57 Rb = ((np.cos(np.radians(L - Beta)) * np.cos(np.radians(Ds)) * np.sin(np.
      radians(Wsrsr_ss))) +
58       (np.radians(Wsrsr_ss)) * np.sin(np.radians(L - Beta)) * np.sin(np.
      radians(Ds))) / \
59       ((np.cos(np.radians(L)) * np.cos(np.radians(Ds)) * np.sin(np.radians(
      Wsrsr_ss))) +
60       (np.radians(Wsrsr_ss)) * np.sin(np.radians(L)) * np.sin(np.radians(Ds
      ))))
61
62 Rd = (1 + np.cos(np.radians(Beta))) / 2
63 Rr = (0.3 * (1 - np.cos(np.radians(Beta)))) / 2
64
65 G_B_Beta = G_B * Rb
66 G_D_Beta = G_D * Rd
67 G_R = G_T * Rr
68 G_T_Beta = G_B_Beta + G_D_Beta + G_R
69
70
71 plt.figure(figsize=(9, 5))
72 plt.plot(LMT, G_T, color='orange', label='Global radiation on horizontal
      surface')
73 plt.plot(LMT, G_T_Beta, 'k', label='Global radiation on tilted surface')
74 plt.xlabel('Time (min)')
75 plt.ylabel(r'Solar radiation (W/m$^2$)')
76 plt.xlim(min(LMT), max(LMT))
77 plt.ylim(200, 1100)
78 plt.grid(True)
79 plt.legend()
80 plt.tight_layout()
81 plt.show()

```

Results:

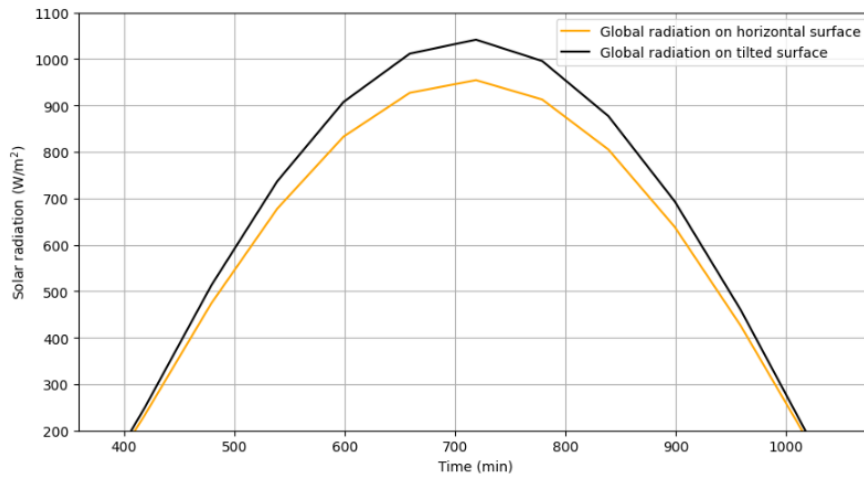


Figure 5: Global solar radiation on horizontal and tilted surfaces for Kuwait City

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:

```

1 #Example 1.7
2 import numpy as np
3 import pandas as pd
4 import matplotlib.pyplot as plt
5
6
7 file_path = 'PV Modeling Book Data Source.csv'
8 data = pd.read_csv(file_path)
9
10
11 N = pd.to_numeric(data.iloc[:, 2], errors='coerce')
12 LMT = pd.to_numeric(data.iloc[:, 3], errors='coerce')
13 G_T = pd.to_numeric(data.iloc[:, 4], errors='coerce')
14 S_So = pd.to_numeric(data.iloc[:, 8], errors='coerce')
15
16
17 valid = N.notna() & LMT.notna() & G_T.notna() & S_So.notna()
18 N = N[valid].to_numpy()
19 LMT = LMT[valid].to_numpy()
20 G_T = G_T[valid].to_numpy()
21 S_So = S_So[valid].to_numpy()
22
23
24 L = 3.11
25 Go = 1367
26
27
28 Ts = LMT
29 Ds = 23.45 * np.sin(np.radians((360 * (N - 81)) / 365))
30 Hs = 15 * (Ts - 12)

```

```

31 sin_Alpha = (np.sin(np.radians(L)) * np.sin(np.radians(Ds)) +
32              np.cos(np.radians(L)) * np.cos(np.radians(Ds)) * np.cos(np.
33              radians(Hs)))
34
35 Gext = Go * (1 + 0.0333 * np.cos(np.radians(360 * N / 365)))
36 GextH = Gext * sin_Alpha
37 G_T_G_ext = G_T / GextH
38
39
40 valid_mask = np.isfinite(G_T_G_ext) & np.isfinite(S_So)
41 G_T_G_ext = G_T_G_ext[valid_mask]
42 S_So = S_So[valid_mask]
43
44
45 P_Liner = np.polyfit(S_So, G_T_G_ext, 1)
46 X_Liner = np.linspace(0, 1, 100)
47 Y_Liner = np.polyval(P_Liner, X_Liner)
48
49
50 plt.figure(figsize=(7, 5))
51 plt.plot(S_So, G_T_G_ext, 'k.', markersize=3)
52 plt.plot(X_Liner, Y_Liner, 'r-', linewidth=1.5)
53 plt.xlabel(r'$ (S/S_o) $')
54 plt.ylabel(r'$ G_T/G_{ext} $')
55 plt.xlim(0, 1)
56 plt.ylim(0, 1)
57 plt.grid(True, linestyle='--', alpha=0.5)
58 plt.tight_layout()
59 plt.show()

```

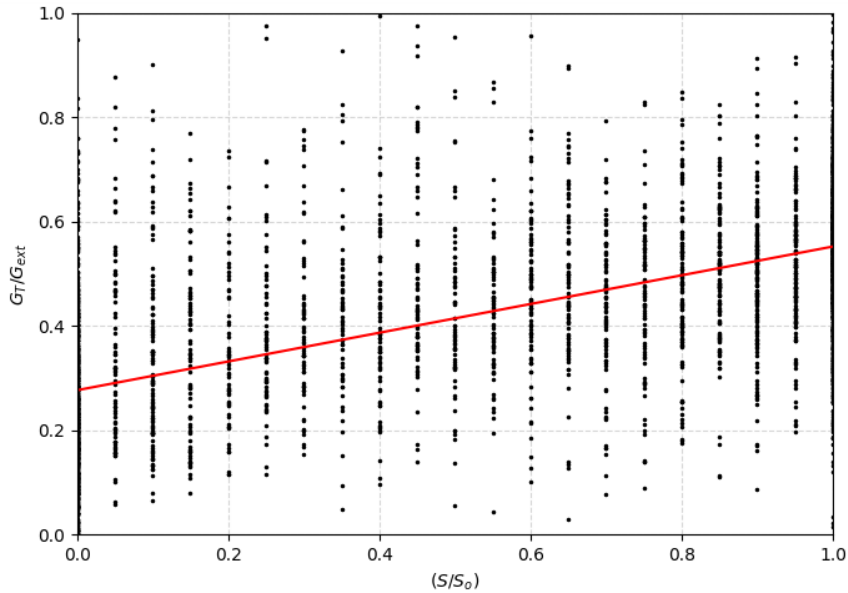


Figure 6: Modeling of global solar radiation on a horizontal surface using linear model.

Result:

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

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:

```

1 #Example 1.8
2 import numpy as np
3 import pandas as pd
4 import matplotlib.pyplot as plt
5
6
7 file_path = 'PV Modeling Book Data Source.csv'
8 data = pd.read_csv(file_path)
9
10 # Extract and ensure numeric conversion for relevant columns
11 N = pd.to_numeric(data.iloc[:, 2], errors='coerce')
12 LMT = pd.to_numeric(data.iloc[:, 3], errors='coerce')
13 G_T = pd.to_numeric(data.iloc[:, 4], errors='coerce')
14 G_D = pd.to_numeric(data.iloc[:, 5], errors='coerce')
15
16
17 valid = N.notna() & LMT.notna() & G_T.notna() & G_D.notna()
18 N = N[valid].to_numpy()
19 LMT = LMT[valid].to_numpy()
20 G_T = G_T[valid].to_numpy()
21 G_D = G_D[valid].to_numpy()
22
23
24 L = 3.11
25 Go = 1367
26
27
28 Ts = LMT
29 Ds = 23.45 * np.sin(np.radians((360 * (N - 81)) / 365))
30 Hs = 15 * (Ts - 12)
31
32 sin_Alpha = (np.sin(np.radians(L)) * np.sin(np.radians(Ds)) +
33              np.cos(np.radians(L)) * np.cos(np.radians(Ds)) * np.cos(np.
34              radians(Hs)))
35
36 Gext = Go * (1 + 0.0333 * np.cos(np.radians(360 * N / 365)))
37 GextH = Gext * sin_Alpha
38 G_T_G_ext = G_T / GextH
39 G_D_G_T = G_D / G_T
40
41 valid_mask = np.isfinite(G_T_G_ext) & np.isfinite(G_D_G_T)
42 G_T_G_ext = G_T_G_ext[valid_mask]
43 G_D_G_T = G_D_G_T[valid_mask]
44

```

```

45
46 P_Liner = np.polyfit(G_T_G_ext, G_D_G_T, 1)
47 X_Liner = np.linspace(min(G_T_G_ext), max(G_T_G_ext), 100)
48 Y_Liner = np.polyval(P_Liner, X_Liner)
49
50
51 plt.figure(figsize=(8, 4))
52 plt.plot(G_T_G_ext, G_D_G_T, 'b.', markersize=3, label=r'$G_D/G_T$ vs.
    $K_T$')
53 plt.plot(X_Liner, Y_Liner, 'r-', linewidth=1, label='Linear model')
54 plt.xlabel(r'$K_T = G_T/G_{\{ext\}}$')
55 plt.ylabel(r'$G_D/G_T$')
56 plt.xlim(0, 1.2)
57 plt.ylim(0.2, 1.0)
58 plt.grid(True, linestyle='--', alpha=0.5)
59 plt.legend()
60 plt.tight_layout()
61 plt.show()

```

Result:

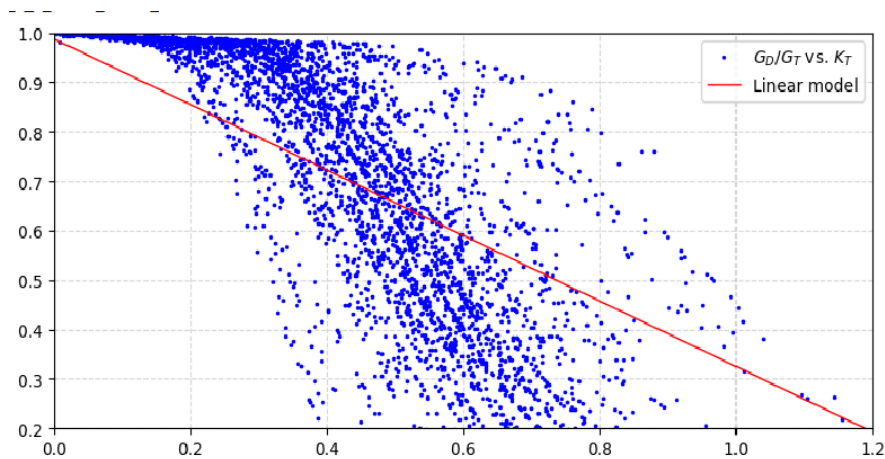


Figure 7: Modeling of diffuse solar radiation on a horizontal surface using linear model.

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".

```

1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
4 from sklearn.neural_network import MLPRegressor
5 from sklearn.preprocessing import StandardScaler

```

```

6  from sklearn.pipeline import make_pipeline
7  from scipy.signal import savgol_filter
8
9
10 file_path = 'PV Modeling Book Data Source.xls'
11 sheet_name = 'Source 1'
12 data = pd.read_excel(file_path, sheet_name=sheet_name)
13
14
15 G_T = data.iloc[4:2997, 4].values
16 G_D = data.iloc[4:2997, 5].values
17 Hum = data.iloc[4:2997, 7].values
18 T = data.iloc[4:2997, 9].values
19 S = data.iloc[4:2997, 8].values
20 M = data.iloc[4:2997, 0].values
21 D = data.iloc[4:2997, 1].values
22 H = data.iloc[4:2997, 3].values
23
24
25 G_T_Test = data.iloc[2997:3640, 4].values
26 G_D_Test = data.iloc[2997:3640, 5].values
27 Hum_Test = data.iloc[2997:3640, 7].values
28 T_Test = data.iloc[2997:3640, 9].values
29 S_Test = data.iloc[2997:3640, 8].values
30 M_Test = data.iloc[2997:3640, 0].values
31 D_Test = data.iloc[2997:3640, 1].values
32 H_Test = data.iloc[2997:3640, 3].values
33
34
35 inputs = np.array([M, D, H, T, Hum, S]).T
36 targets = np.array([G_T, G_D]).T
37 test_inputs = np.array([M_Test, D_Test, H_Test, T_Test, Hum_Test, S_Test]).
    T
38
39
40 net = make_pipeline(
41     StandardScaler(),
42     MLPRegressor(hidden_layer_sizes=(10, 10), max_iter=1000, solver='adam',
        random_state=42)
43 )
44
45
46 net.fit(inputs, targets)
47
48
49 G_Mi = net.predict(test_inputs)
50 G_Tp = G_Mi[:, 0]
51 G_Dp = G_Mi[:, 1]
52
53
54 G_Dp_smooth = savgol_filter(G_Dp[:100], window_length=9, polyorder=2)
55
56
57 x_range = np.arange(0, 100)
58
59 plt.figure(figsize=(10, 6))
60
61

```

```

62 plt.subplot(2, 1, 1)
63 plt.plot(x_range, G_T_Test[:100], 'b-', linewidth=1.2, label='Actual data')
64 plt.plot(x_range, G_Tp[:100], 'r-', linewidth=1.2, label='Predicted data')
65 plt.ylabel('Global solar radiation\n$(W/m^2)$')
66 plt.xticks(np.arange(0, 101, 10))
67 plt.xlim(0, 100)
68 plt.ylim(0, 850)
69 plt.grid(True)
70 plt.legend()
71
72
73 plt.subplot(2, 1, 2)
74 plt.plot(x_range, G_D_Test[:100], 'b-', linewidth=1.2, label='Actual data')
75 plt.plot(x_range, G_Dp_smooth, 'r-', linewidth=1.2, label='Predicted data')
76 plt.xlabel('Time (h)')
77 plt.ylabel('Diffuse solar radiation\n$(W/m^2)$')
78 plt.xticks(np.arange(0, 101, 10))
79 plt.xlim(0, 100)
80 plt.ylim(0, 500)
81 plt.grid(True)
82 plt.legend()
83
84 plt.tight_layout()
85 plt.show()

```

Answer:

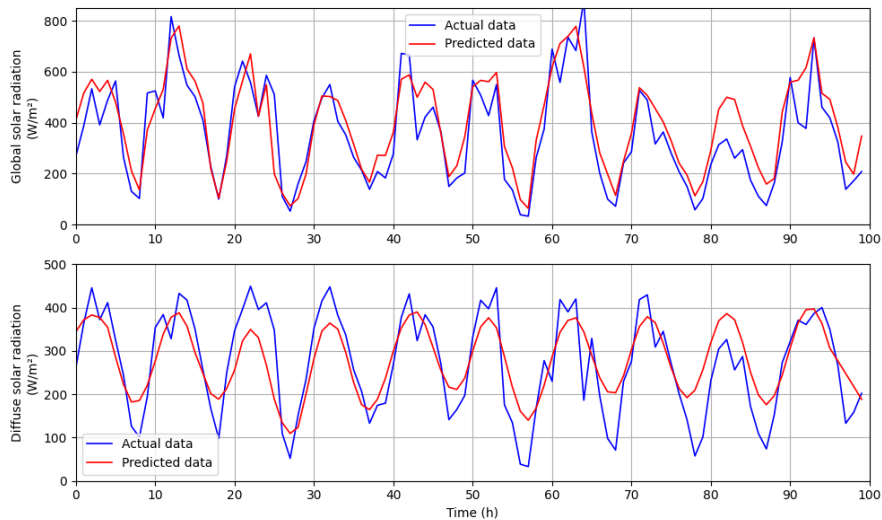


Figure 8: Prediction results of ANN model in Example 1.8

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

Solution:

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 L = 3.12 # Latitude for Kuala Lumpur
5 LOD = 101.7 # Longitude
6 T_GMT = 8 # Time difference from GMT
7 Step = 5 # Time step in minutes
8
9 BetaT = []
10
11
12 for N in range(1, 5):
13     Ds = 23.45 * np.sin(np.radians((360 * (N - 81)) / 365))
14     B = (360 * (N - 81)) / 364
15     EoT = (9.87 * np.sin(np.radians(2 * B))) - (7.53 * np.cos(np.radians(B)
16         )) - (1.5 * np.sin(np.radians(B)))
17     Lzt = 15 * T_GMT
18     Ts_correction = (-4 * (Lzt - LOD) + EoT) if LOD >= 0 else (4 * (Lzt -
19         LOD) + EoT)
20
21     Wsr_ssi = -np.tan(np.radians(Ds)) * np.tan(np.radians(L))
22     Wsr_ssr_ss = np.degrees(np.arccos(Wsr_ssi))
23     ASTsr = abs(((Wsr_ssr_ss / 15) - 12) * 60)
24     ASTss = ((Wsr_ssr_ss / 15) + 12) * 60
25     Tsr = ASTsr + abs(Ts_correction)
26     Tss = ASTss + abs(Ts_correction)
27
28     Alpha = []
29
30     for LMT in np.arange(Tsr, Tss + Step, Step):
31         Ts = LMT + Ts_correction
32         Hs = (15 * (Ts - (12 * 60))) / 60
33         sin_Alpha = (np.sin(np.radians(L)) * np.sin(np.radians(Ds)) +
34             np.cos(np.radians(L)) * np.cos(np.radians(Ds)) * np.
35             cos(np.radians(Hs)))
36         Alpha_i = np.degrees(np.arcsin(sin_Alpha))
37         Alpha.append(Alpha_i)
38
39     Alpha = np.array(Alpha)
40     Beta = 90 - Alpha
41     BetaT.append(Beta)
42
43 BetaT = np.array(BetaT).T
44
45 Beta1 = BetaT[:, 0]
46 Beta2 = BetaT[:, 1]
47 Beta3 = BetaT[:, 2]
48 Beta4 = BetaT[:, 3]
49
50 fig, axs = plt.subplots(2, 2, figsize=(10, 8))
51
52 axs[0, 0].plot(Beta1, color='orange')
53 axs[0, 0].set_ylim(20, 100)
54

```



```

55 axs[0, 1].plot(Beta2, color='orange')
56 axs[0, 1].set_ylim(20, 100)
57
58 axs[1, 0].plot(Beta3, color='orange')
59 axs[1, 0].set_ylim(20, 100)
60
61 axs[1, 1].plot(Beta4, color='orange')
62 axs[1, 1].set_ylim(20, 100)
63
64 for ax in axs.flat:
65     ax.set_ylabel('Tilt angle (°)')
66     ax.set_xlabel('Time(min)')
67     ax.grid(True)
68
69 plt.tight_layout()
70 plt.show()

```

Results:

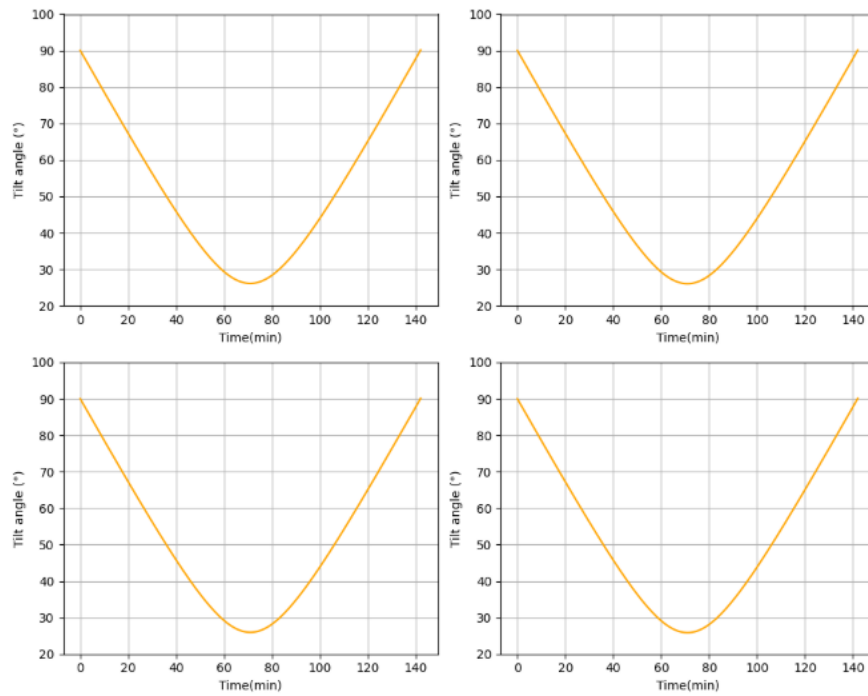


Figure 9: Optimum tilt angle results

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)		■ Cells	
Maximum power(Pmax)	200 W(+10%/-5%)	Number per module	54
Maximum power voltage (Vmpp)	26.3 V		
Maximum power current (Impp)	7.16 A	■ Module characteristics	
Open circuit voltage(Voc)	32.9 V	Length × width × depth	1425 mm(56.2in)×990 mm(39.0in)×36 mm(1.4in)
Short circuit current(Isc)	8.21 A	Weight	18.5 kg(40.7 lbs)
Max system voltage	600 V	Cable	(+)720 mm(28.3in), (-)1800 mm(70.9in)
Temperature coefficient of Voc	-1.23 × 10 ⁻³ V/°C	■ Junction box characteristics	
Temperature coefficient of Isc	3.18 × 10 ⁻³ A/°C	Length × width × depth	113.6 mm(4.5in)×76 mm(3.0in)×9 mm(0.4in)
*STC: Irradiance 1000 W/m ² , AM1.5 spectrum, module temperature 25°C		IP code	IP65
■ Electrical performance at 800 W/m ² , NOCT, AM1.5		■ Reduction of efficiency under low Irradiance	
Maximum power(Pmax)	142 W	Reduction	7.8%
Maximum power voltage (Vmpp)	23.2 V	Reduction of efficiency from an irradiance of 1000 to 200 W/m ² (module temperature 25°C)	
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			

From Kyocera 200W manufacturer.

Solution:

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 def PV_model(Va, Suns, TaC):
5
6     k = 1.38e-23
7     q = 1.60e-19
8     A = 1.2
9     Vg = 1.12
10    Ns = 54
11
12
13    T1 = 273 + 25
14    T2 = 273 + 75
15    TaK = 273 + TaC
16
17
18    Voc_T1 = 32.9 / Ns
19    Isc_T1 = 8.21
20    Voc_T2 = 29.9 / Ns
21    Isc_T2 = 6.62
22
23
24    Iph_T1 = Isc_T1 * Suns
25    a = (Isc_T2 - Isc_T1) / Isc_T1 / (T2 - T1)
26    Iph = Iph_T1 * (1 + a * (TaK - T1))
27
28    # Thermal voltage and reverse saturation current
29    Vt_T1 = k * T1 / q
30    Ir_T1 = Isc_T1 / (np.exp(Voc_T1 / (A * Vt_T1)) - 1)
31    Ir_T2 = Isc_T2 / (np.exp(Voc_T2 / (A * Vt_T1)) - 1)

```

```

32 b = Vg * q / (A * k)
33 Ir = Ir_T1 * (TaK / T1) ** (3 / A) * np.exp(-b * (1 / TaK - 1 / T1))
34
35
36 X2v = Ir_T1 / (A * Vt_T1) * np.exp(Voc_T1 / (A * Vt_T1))
37 dVdI_Voc = -1.15 / Ns / 2
38 Rs = -dVdI_Voc - 1 / X2v
39
40
41 Vt_Ta = A * k * TaK / q
42 Vc = Va / Ns
43 Ia = np.zeros_like(Vc)
44
45
46 for _ in range(5):
47     exp_term = np.exp((Vc + Ia * Rs) / Vt_Ta)
48     Ia = Ia - (Iph - Ia - Ir * (exp_term - 1)) / (-1 - Ir * exp_term *
49         Rs / Vt_Ta)
50
51 return Ia
52
53 Suns = 1
54 TaC = 25
55 Va = np.arange(0, 34, 1)
56 Ia = PV_model(Va, Suns, TaC)
57 P = Va * Ia
58
59
60 plt.figure(figsize=(10, 5))
61 plt.subplot(2, 1, 1)
62 plt.plot(Va, Ia)
63 plt.title('Current vs Voltage')
64
65 plt.subplot(2, 1, 2)
66 plt.plot(Va, P)
67 plt.title('Power vs Voltage')
68 plt.xlabel('Voltage (V)')
69 plt.tight_layout()
70 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)

```

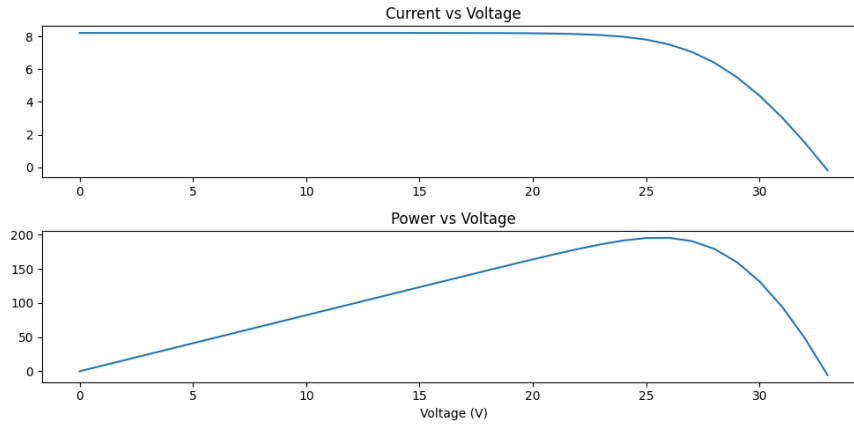


Figure 10: I–V and P–V characteristic PV module at 1000 W/m² and 25°C

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

```

1
2 #Example 2.2
3 import numpy as np
4 import pandas as pd
5 from google.colab import files
6 file_path = '/content/PV Modeling Book Data Source.xls'
7 sheet_name = 'Source 2'
8
9
10 df = pd.read_excel(file_path, sheet_name=sheetName, usecols="A:C", skiprows
    =1)
11
12 G = df.iloc[:, 0].values # Global solar radiation
13 Temp = df.iloc[:, 1].values # Ambient temperature
14 I_PV = df.iloc[:, 2].values # PV actual current
15
16 # -----Modeling of global solar energy-----
17 N_Liner = 1 # Order of the function
18 mask = ~np.isnan(G) & ~np.isnan(I_PV) & ~np.isinf(G) & ~np.isinf(I_PV)
19 G_clean = G[mask]
20 I_PV_clean = I_PV[mask]
21 P_Liner = np.polyfit(G_clean, I_PV_clean, N_Liner)
22
23
24 X_Liner = I_PV
25 Y_Liner = np.zeros_like(X_Liner)
26
27 for i in range(1, N_Liner + 2):
28     Y_Liner += P_Liner[i - 1] * X_Liner**(N_Liner - i + 1)

```

Answer:

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

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:

```
1 # Example 2.3
2 import numpy as np
3 import pandas as pd
4 import matplotlib.pyplot as plt
5 from sklearn.neural_network import MLPRegressor
6 from sklearn.linear_model import Ridge
7 from sklearn.preprocessing import StandardScaler
8 from sklearn.metrics import mean_squared_error,
   mean_absolute_percentage_error
9
10 # Load Excel data
11 file_path = '/content/PV Modeling Book Data Source.xls'
12 sheet_name = 'Source 2'
13
14 # Read training data
15 df = pd.read_excel(file_path, sheet_name='Source 2', usecols="A:C",
   skiprows=1)
16
17 # Remove any rows with NaN values
18 df.dropna(inplace=True)
19
20 G = df.iloc[:, 0].values # Global solar radiation
21 Temp = df.iloc[:, 1].values # Ambient temperature
22 I_PV = df.iloc[:, 2].values # PV actual current
23
24
25 # Read test data
26 df_test = pd.read_excel(file_path, sheet_name=sheet_name, usecols="A:C",
   skiprows=36002)
27 G_Test = df_test.iloc[:, 0].values
28 Temp_Test = df_test.iloc[:, 1].values
29 I_PV_Test = df_test.iloc[:, 2].values
30
31 # Prepare inputs
32 inputs = np.column_stack((G, Temp))
33 test_inputs = np.column_stack((G_Test, Temp_Test))
34
35 # Ask user for network type
36 print("Choose the network type:\n1. FFANN\n2. GRNN Simulation")
37 k = input("Enter your choice (1 or 2): ")
38
39 # Scale inputs
40 scaler = StandardScaler()
41 inputs_scaled = scaler.fit_transform(inputs)
42 test_scaled = scaler.transform(test_inputs)
```

```

43
44 # Select and train network
45 if k == "1":
46     net = MLPRegressor(hidden_layer_sizes=(5,), solver='lbfgs', max_iter
47                         =1000, random_state=1)
48 elif k == "2":
49     net = Ridge(alpha=1.0) # Simulated GRNN
50 else:
51     raise ValueError("Invalid choice")
52
53 net.fit(inputs_scaled, I_PV)
54 C_ANN = net.predict(test_scaled)
55
56 # Theoretical current
57 C_th = (G_Test / 1000) * 7.91
58
59 # Regression model (empirical)
60 C_Reg = -1.17112 + 0.009 * G + 0.055 * Temp
61
62 # Plot results
63 plt.figure(figsize=(10, 6))
64 plt.plot(I_PV_Test, label='Measured')
65 plt.plot(C_ANN, label='Predicted by ANN')
66 plt.plot(C_th, label='Empirical Model')
67 plt.legend()
68 plt.title("Output Current Prediction")
69 plt.xlabel("Time Step")
70 plt.ylabel("Current (A)")
71 plt.grid(True)
72 plt.show()
73
74 # Compute evaluation metrics over averaged steps (each hour = 360 samples
75 # at 10s steps)
76 steps = 5760
77 def averaged(arr, step):
78     return np.array([np.mean(arr[i:i+step]) for i in range(0, len(arr) -
79                     step + 1, step)])
80
81 AV_C_Test = averaged(I_PV_Test, steps)
82 AV_C_ANN = averaged(C_ANN, steps)
83
84 E3_Hour = AV_C_Test - AV_C_ANN
85 ANN_MAPE = np.abs(E3_Hour / AV_C_Test)
86 ANN_meanMAPE = np.mean(ANN_MAPE) * 100
87 ANN_RMSE = np.sqrt(np.mean((AV_C_ANN - AV_C_Test)**2))
88 ANN_MBE = np.mean(AV_C_ANN - AV_C_Test)
89 SUM = np.mean(AV_C_ANN)
90 ANN_RMSE_Percentage = (ANN_RMSE / SUM) * 100
91 ANN_MBE_Percentage = (ANN_MBE / SUM) * 100
92
93 import pandas as pd
94
95 metrics = pd.DataFrame({
96     'ANN_meanMAPE (%)': [ANN_meanMAPE],
97     'ANN_RMSE': [ANN_RMSE],
98     'ANN_MBE': [ANN_MBE],
99     'ANN_RMSE_Percentage (%)': [ANN_RMSE_Percentage],

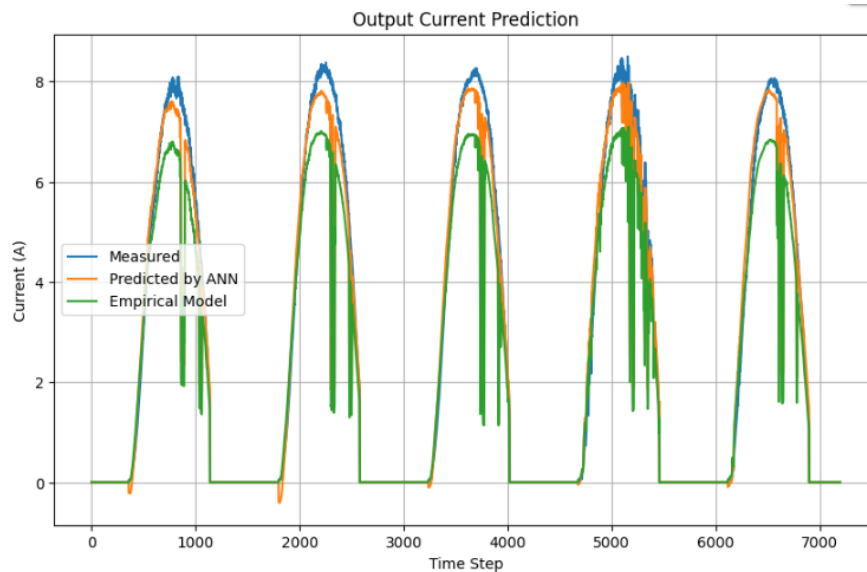
```

```

98     'ANN_MBE_Percentage (%)': [ANN_MBE_Percentage]
99 })

```

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.

```

1  #exercise 2.4
2  import pandas as pd
3  import numpy as np
4  import matplotlib.pyplot as plt
5  from sklearn.ensemble import RandomForestRegressor
6  from sklearn.metrics import mean_squared_error, mean_absolute_error
7  from sklearn.manifold import MDS
8  import time
9  import warnings
10 warnings.filterwarnings('ignore')
11
12
13
14 fileName = 'PV Modeling Book Data Source.xls'

```



```

15 sheetName = 'Source 3'
16
17
18 df = pd.read_excel(fileName, sheet_name=sheetName)
19
20 DN = df.iloc[2:1884, 1].values # Day Number (B3:B1884)
21 H = df.iloc[2:1884, 2].values # Hour (C3:C1884)
22 T = df.iloc[2:1884, 3].values # Ambient Temp ( C ) (D3:D1884)
23 S = df.iloc[2:1884, 4].values # Solar Radiation (E3:E1884)
24 La = df.iloc[2:1884, 5].values # Latitude (F3:F1884)
25 Lo = df.iloc[2:1884, 6].values # Longitude (G3:G1884)
26 NPV = df.iloc[2:1884, 7].values # Number of PV Modules (H3:H1884)
27 I = df.iloc[2:1884, 9].values # PV DC Current (A) (J3:J1884)
28
29 start_time = time.time()
30 Y = I
31 X = np.column_stack([DN, H, T, S, La, Lo, NPV])
32 t = 125
33
34 B = RandomForestRegressor(n_estimators=t, oob_score=True, random_state=42)
35 B.fit(X, Y)
36
37
38 oob_errors = []
39 for i in range(1, t+1):
40     temp_rf = RandomForestRegressor(n_estimators=i, oob_score=True,
41                                     random_state=42)
42     temp_rf.fit(X, Y)
43     oob_errors.append(1 - temp_rf.oob_score_)
44
45 plt.figure(1)
46 plt.plot(range(1, t+1), oob_errors)
47 plt.xlabel('Number of Grown Trees')
48 plt.ylabel('Out-of-Bag Mean Squared Error')
49 plt.show()
50
51 feature_importance = B.feature_importances_
52 plt.figure(2)
53 plt.bar(range(1, len(feature_importance)+1), feature_importance)
54 plt.title('Variable Importance')
55 plt.xlabel('Variable Number')
56 plt.ylabel('Out-of-Bag Variable Importance')
57 plt.legend(['1: Day Number, 2: Hour, 3: Ambient Temp, 4: Solar Radiation,
58             5: Latitude, 6: Longitude, 7: # of PV Modules'], loc='upper right')
59 plt.show()
60
61
62
63 nidx = np.where(feature_importance < 0.65)[0]
64
65
66 finbag = np.linspace(0.6, 0.9, t)
67 plt.figure(3)
68 plt.plot(range(1, t+1), finbag)
69 plt.xlabel('Number of Grown Trees')
70 plt.ylabel('Fraction of in-Bag Observations')
71 plt.show()
72
73 predictions = B.predict(X)

```

```

71 residuals = Y - predictions
72 outlier_measure = np.abs(residuals)
73
74 plt.figure(4)
75 plt.hist(outlier_measure, bins=30)
76 plt.title('The Outliers')
77 plt.xlabel('Outlier Measure')
78 plt.ylabel('Number of Observations')
79 plt.show()
80
81
82 mds = MDS(n_components=2, random_state=42)
83 X_mds = mds.fit_transform(X)
84
85 plt.figure(5)
86 plt.scatter(X_mds[:, 0], X_mds[:, 1], c='k', alpha=0.6)
87 plt.title('Cluster Analysis')
88 plt.xlabel('1st Scaled Coordinate')
89 plt.ylabel('2nd Scaled Coordinate')
90 plt.show()
91
92
93 eigenvalues = np.random.exponential(scale=2, size=20)
94 eigenvalues = np.sort(eigenvalues)[::-1]
95
96 plt.figure(6)
97 plt.bar(range(1, 21), eigenvalues)
98 plt.xlabel('Scaled Coordinate Index')
99 plt.ylabel('Eigen Value')
100 plt.show()
101
102
103 filename = 'Data.xlsx'
104 sheet = 0
105
106 df_test = pd.read_excel(filename, sheet_name=sheet)
107
108 DN_t = df_test.iloc[1884:2690, 1].values
109 H_t = df_test.iloc[1884:2690, 2].values
110 T_t = df_test.iloc[1884:2690, 3].values
111 S_t = df_test.iloc[1884:2690, 4].values
112 La_t = df_test.iloc[1884:2690, 5].values
113 Lo_t = df_test.iloc[1884:2690, 6].values
114 NPV_t = df_test.iloc[1884:2690, 7].values
115 I_t = df_test.iloc[1884:2690, 9].values
116
117
118 Xdata = np.column_stack([DN_t, H_t, T_t[:len(DN_t)], S_t, La_t, Lo_t, NPV_t
119 ])
119 Yfit = B.predict(Xdata)
120
121 plt.figure(7)
122 plt.plot(Yfit, label='I Predicted')
123 plt.plot(I_t, 'red', label='I Actual')
124 plt.xlabel('Time (H)')
125 plt.ylabel('Current (A)')
126 plt.legend(loc='upper right')
127 plt.title('I Predicted Vs I Actual')

```

```

128 plt.show()
129
130 plt.figure(8)
131 E = I_t - Yfit
132 plt.plot(E)
133 plt.xlabel('Time (H)')
134 plt.ylabel('Magnitude (A)')
135 plt.title('Error')
136 plt.show()
137
138 elapsed_time = time.time() - start_time
139
140
141 MBE = np.sum(I_t - Yfit) / len(I_t)
142
143 if MBE < 0:
144     F = 'Over forecasted'
145 elif MBE > 0:
146     F = 'Under Forecasted'
147 elif MBE == 0:
148     F = 'Ideal Forecasted'
149
150
151 MAPE = (np.abs(np.sum((I_t - Yfit) / I_t)) / len(I_t)) * 100
152
153 # Root Mean Square Error (RMSE)
154 RMSE = np.sum((I_t - Yfit)**2) / len(I_t)
155
156 # Outputs
157 n1 = f'Mean Bias Error (MBE): {MBE} (A) {{Average Deviation Indicator}}'
158 n2 = f'Forecasting Status: {F}'
159 n3 = f'Mean Absolute Percentage Error (MAPE): {MAPE}% {{Accuracy Indicator}}'
160 n4 = f'Root Mean Square Error (RMSE): {RMSE} (A) {{Efficiency Indicator}}'
161
162 print(n1)
163 print(n2)
164 print(n3)
165 print(n4)
166
167
168 filename = 'Data.xlsx'
169 sheet = 0
170
171 df_train = pd.read_excel(filename, sheet_name=sheet)
172
173 DN = df_train.iloc[2:1884, 1].values
174 H = df_train.iloc[2:1884, 2].values
175 T = df_train.iloc[2:1884, 3].values
176 S = df_train.iloc[2:1884, 4].values
177 I = df_train.iloc[2:1884, 9].values
178
179
180 DN_t = df_train.iloc[1884:2690, 1].values
181 H_t = df_train.iloc[1884:2690, 2].values
182 T_t = df_train.iloc[1884:2690, 3].values
183 S_t = df_train.iloc[1884:2690, 4].values
184 I_t = df_train.iloc[1884:2690, 9].values

```

```

185
186
187 Y = I
188 X = np.column_stack([DN, H, T, S])
189 Xdata = np.column_stack([DN_t, H_t, T_t[:len(DN_t)], S_t])
190
191 MBE_results = []
192 MAPE_results = []
193 RMSE_results = []
194 time_results = []
195
196 for t in range(1, 501):
197     for l in range(1, 101):
198         start_time = time.time()
199         B = RandomForestRegressor(n_estimators=t, min_samples_leaf=1,
200                                 oob_score=True, random_state=42)
201         B.fit(X, Y)
202
203         # Testing
204         Yfit = B.predict(Xdata)
205         elapsed = time.time() - start_time
206         time_results.append(elapsed)
207
208         E = I_t - Yfit
209
210         # Performance metrics
211         MBE_val = np.sum(I_t - Yfit) / len(I_t)
212         MBE_results.append(MBE_val)
213
214         if MBE_val < 0:
215             F = 'Over forecasted'
216         elif MBE_val > 0:
217             F = 'Under Forecasted'
218         elif MBE_val == 0:
219             F = 'Ideal Forecasted'
220
221         MAPE_val = (np.sum(np.abs(E)) / np.sum(I_t)) * 100
222         MAPE_results.append(MAPE_val)
223
224         RMSE_val = np.sum((I_t - Yfit)**2) / len(I_t)
225         RMSE_results.append(RMSE_val)
226
227 filename = 'RF - NTrees - 5.xlsx'
228 sheet = 0
229
230 df_results = pd.read_excel(filename, sheet_name=sheet)
231
232 NT = df_results.iloc[1:22, 1].values # Trees Number
233 ET = df_results.iloc[1:22, 3].values # Elapsed time (Sec.)
234 MBE = df_results.iloc[1:22, 4].values # Mean Bias Error (MBE) (A)
235 MAPE = df_results.iloc[1:22, 6].values # Mean Absolute Percentage Error (
236     MAPE) (%)
237 RMSE = df_results.iloc[1:22, 8].values # Root Mean Square Error (RMSE) (A)
238 OOB = df_results.iloc[1:22, 9].values # Out of Bag (OOB)
239
240 for i in range(len(NT)):
241     plt.figure(9)

```

```

241 plt.plot(NT, RMSE, '*-')
242 plt.xlabel('Number of Trees')
243 plt.ylabel('Root Mean Squared Error (A)')
244
245 plt.figure(10)
246 plt.plot(NT, ET, '*-')
247 plt.xlabel('Number of Trees')
248 plt.ylabel('Elapsed time (Sec.)')
249
250 plt.figure(11)
251 plt.plot(NT, MBE, '*-')
252 plt.xlabel('Number of Trees')
253 plt.ylabel('Mean Bias Error (MBE) (A)')
254
255 plt.figure(12)
256 plt.plot(NT, MAPE, '*-')
257 plt.xlabel('Number of Trees')
258 plt.ylabel('Mean Absolute Percentage Error (MAPE) (%)')
259
260 plt.figure(13)
261 plt.plot(NT, OOB, '*-')
262 plt.xlabel('Number of Trees')
263 plt.ylabel('Out of Bag (OOB)')
264
265 plt.show()
266
267 M1, I1 = np.min(RMSE), np.argmin(RMSE)
268 M2, I2 = np.min(ET), np.argmin(ET)
269 M3, I3 = np.min(MAPE), np.argmin(MAPE)
270 M4, I4 = np.min(MBE), np.argmin(MBE)
271 M5, I5 = np.min(OOB), np.argmin(OOB)
272
273 n1 = f'Min. Root Mean Squared Error : {M1}(A) @ index : {I1}'
274 n2 = f'Min. Elapsed time : {M2}(Sec.) @ index: {I2}'
275 n3 = f'Min. Mean Absolute Percentage Error (MAPE) : {M3}(%) @ index : {I3}'
276 n4 = f'Min. Mean Bias Error (MBE) : {M4}(A) @ index : {I4}'
277 n5 = f'Min. Out of Bag (OOB) data : {M5}(A) @ index : {I5}'
278
279 print(n1)
280 print(n2)
281 print(n3)
282 print(n4)
283 print(n5)
284
285 filename = 'Data.xlsx'
286 sheet = 0
287
288
289 df_final = pd.read_excel(filename, sheet_name=sheet)
290
291 DN = df_final.iloc[2:1884, 1].values
292 H = df_final.iloc[2:1884, 2].values
293 T = df_final.iloc[2:1884, 3].values
294 S = df_final.iloc[2:1884, 4].values
295 I = df_final.iloc[2:1884, 9].values
296
297 # RF_Training Code
298 Y = I

```

```

299 X = np.column_stack([DN, H, T, S])
300 t = 65
301 i = 1
302
303 B = RandomForestRegressor(n_estimators=t, min_samples_leaf=i, oob_score=
    True, random_state=42)
304 B.fit(X, Y)
305
306 filename = 'Data.xlsx'
307 sheet = 0
308
309 # Testing Data - Excel File
310 DN_t = df_final.iloc[1884:2690, 1].values
311 H_t = df_final.iloc[1884:2690, 2].values
312 T_t = df_final.iloc[1884:2690, 3].values
313 S_t = df_final.iloc[1884:2690, 4].values
314 I_t = df_final.iloc[1884:2690, 9].values
315
316 # RF_Testing Code
317 Xdata = np.column_stack([DN_t, H_t, T_t[:len(DN_t)], S_t])
318 Yfit = B.predict(Xdata)
319
320 plt.figure(14)
321 plt.plot(Yfit, label='I Predicted')
322 plt.plot(I_t, 'red', label='I Actual')
323 plt.xlabel('Time (H)')
324 plt.ylabel('Current (A)')
325 plt.legend(loc='upper right')
326 plt.title('I Predicted Vs I Actual')
327 plt.show()
328
329 plt.figure(15)
330 E = I_t - Yfit
331 plt.plot(E)
332 plt.xlabel('Time (H)')
333 plt.ylabel('Magnitude (A)')
334 plt.title('Error')
335 plt.show()
336
337
338 MBE = np.sum(I_t - Yfit) / len(I_t)
339
340 if MBE < 0:
341     F = 'Over forecasted'
342 elif MBE > 0:
343     F = 'Under Forecasted'
344 elif MBE == 0:
345     F = 'Ideal Forecasted'
346
347 # Mean Absolute Percentage Error (MAPE)
348 MAPE = (np.abs(np.sum((I_t - Yfit) / I_t)) / len(I_t)) * 100
349 MAPE = (np.sum(np.abs(E)) / np.sum(I_t)) * 100
350
351 # Root Mean Square Error (RMSE)
352 RMSE = np.sum((I_t - Yfit)**2) / len(I_t)
353
354 # Outputs
355 n1 = f'Mean Bias Error(MBE): {MBE} (A) {{Average Deviation Indicator}}'

```

```

356 n2 = f'Forecasting Status: {F}'
357 n3 = f'Mean Absolute Percentage Error (MAPE): {MAPE}% {{Accuracy Indicator
    }}'
358 n4 = f'Root Mean Square Error (RMSE): {RMSE}(A) {{Efficiency Indicator}}'
359
360 print(n1)
361 print(n2)
362 print(n3)
363 print(n4)
364
365 # ANN Vs RF
366 filename = 'Data.xlsx'
367 sheet = 0
368
369
370 df_comparison = pd.read_excel(filename, sheet_name=sheet)
371
372 I = df_comparison.iloc[2:808, 11].values
373 Iann = df_comparison.iloc[2:808, 13].values
374 Irf = df_comparison.iloc[2:808, 14].values
375
376 plt.plot(I, label='I Actual')
377 plt.plot(Iann, ':ks', label='I ANNs Model')
378 plt.plot(Irf, '--ro', label='I Random Forests Model')
379 plt.xlabel('Time (H)')
380 plt.ylabel('Current (A)')
381 plt.legend(loc='upper right')
382 plt.grid(True)
383 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.

```

2 #Example 2.5
3 import numpy as np
4 import pandas as pd
5 import matplotlib.pyplot as plt
6 import time
7 import pickle
8 import os
9 from openpyxl import Workbook
10 from openpyxl import load_workbook
11
12
13 def main():
14     t = time.time()
15     radiation = [978]
16     cell_temperature = [328.56]
17     sheet = 7
18
19
20     df = pd.read_excel('PV Modeling Book Data Source.xls', sheet_name='
        Source 4')
21
22     Ve = df.iloc[2:104, 6].values
23     Ie = df.iloc[2:104, 7].values
24     Vp = Ve.copy()
25
26
27     Ve = np.array(Ve, dtype=float)
28     Ie = np.array(Ie, dtype=float)
29     Vp = np.array(Vp, dtype=float)
30
31
32     with open('var_fitness_function.pkl', 'wb') as f:
33         pickle.dump({'Vp': Vp, 'Ie': Ie}, f)
34
35     solar_radiation = np.array(radiation, dtype=float) / 1000
36     G = solar_radiation
37     Tc = np.array(cell_temperature, dtype=float)
38
39     f_bestt, a_bestt, Rs_bestt, Rp_bestt, Iph_bestt, Io_bestt =
        PV_MODELING_BASED_DE_ALGORITHM(G, Tc)
40
41     a_best = a_bestt
42     Rs_best = Rs_bestt
43     Rp_best = Rp_bestt
44     Iph_best = Iph_bestt
45     Io_best = Io_bestt
46     f_best = f_bestt
47
48
49     Nsc = 36
50     k = 1.3806503e-23
51     q = 1.60217646e-19
52     VT = (Nsc * k * Tc[0]) / q
53
54     Ip = np.zeros_like(Vp, dtype=float)
55     for h in range(5):
56
57         exp_term = np.exp((Vp + Ip * Rs_best) / (a_best * VT))

```



```

58     numerator = Iph_best - Ip - Io_best * (exp_term - 1) - ((Vp + Ip *
        Rs_best) / Rp_best)
59     denominator = -1 - Io_best * (Rs_best / (a_best * VT)) * exp_term -
        (Rs_best / Rp_best)
60     Ip = Ip - (numerator / denominator)
61
62     # Plotting I-V Characteristics
63     plt.figure()
64     plt.plot(Vp, Ip, label='Calculated')
65     plt.plot(Ve, Ie, 'o', label='Experimental')
66     plt.title('I-V characteristics of PV module with solar radiation and
        ambient temperature variations')
67     plt.xlabel('Module voltage (V)')
68     plt.ylabel('Module current (A)')
69     plt.legend()
70     plt.text(0.5, 0.5, '978 W/m^2, 328.56 K', transform=plt.gca().transAxes
        )
71     plt.show()
72
73     # Computing PV Module Power
74     Pp = Vp * Ip
75     Pe = Ve * Ie
76
77     # Plotting P-V Characteristics
78     plt.figure()
79     plt.plot(Vp, Pp, label='Calculated Power')
80     plt.plot(Vp, Pe, 'o', label='Experimental Power')
81     plt.title('P-V characteristics of PV module')
82     plt.xlabel('Module voltage (V)')
83     plt.ylabel('Module power (W)')
84     plt.legend()
85     plt.text(0.5, 0.5, '978 W/m^2, 328.56 K', transform=plt.gca().transAxes
        )
86     plt.show()
87
88
89     Iee = np.mean(Ie)
90     RMSE = np.sqrt((1/len(Vp)) * np.sum((Ip - Ie)**2))
91     MBE = (1/len(Vp)) * np.sum((Ip - Ie)**2)
92     RR = 1 - ((np.sum((Ie - Ip)**2)) / (np.sum((Ie - Iee)**2)))
93
94     print(f"RMSE: {RMSE}")
95     print(f"MBE: {MBE}")
96     print(f"RR: {RR}")
97
98 def PV_MODELING_BASED_DE_ALGORITHM(G, Tc):
99     EP = 0.054
100     D = 5
101     Np = 10 * D
102     F = 0.85
103     CR = 0.6
104     GEN_max = 500
105     sheet = 7
106     file_name = 'Results_of_Radiation_7_temperature_7.pkl'
107
108
109     params = {'Np': Np, 'GEN_max': GEN_max, 'F': F, 'CR': CR}
110     with open(file_name, 'wb') as f:

```

```

111     pickle.dump(params, f)
112
113
114     excel_file = 'PVM4_Try_3.xlsx'
115     if not os.path.exists(excel_file):
116         wb = Workbook()
117         wb.save(excel_file)
118
119     # Write parameters to Excel
120     write_to_excel(excel_file, sheet, 'P5', Np)
121     write_to_excel(excel_file, sheet, 'Q5', GEN_max)
122     write_to_excel(excel_file, sheet, 'R5', F)
123     write_to_excel(excel_file, sheet, 'S5', CR)
124     write_to_excel(excel_file, sheet, 'U5', EP)
125
126     Rs_l = 0.1; Rs_h = 2
127     Rp_l = 100; Rp_h = 5000
128     a_l = 1; a_h = 2
129     Iph_l = 1; Iph_h = 8
130     Io_l = 1e-12; Io_h = 1e-5
131
132     L = np.array([a_l, Rs_l, Rp_l, Iph_l, Io_l], dtype=float)
133     H = np.array([a_h, Rs_h, Rp_h, Iph_h, Io_h], dtype=float)
134
135     rr = 1
136     a_average = np.zeros(rr, dtype=float)
137     Rs_average = np.zeros(rr, dtype=float)
138     Rp_average = np.zeros(rr, dtype=float)
139     Iph_average = np.zeros(rr, dtype=float)
140     Io_average = np.zeros(rr, dtype=float)
141     f_average = np.zeros(rr, dtype=float)
142
143     for b in range(rr):
144         x = np.zeros(D, dtype=float)
145         pop = np.zeros((D, Np), dtype=float)
146         Fit = np.zeros(Np, dtype=float)
147         r = np.zeros(3, dtype=int)
148
149
150         for j in range(Np):
151             for i in range(D):
152                 pop[i, j] = L[i] + (H[i] - L[i]) * np.random.rand()
153
154                 a = float(pop[0, j])
155                 Rs = float(pop[1, j])
156                 Rp = float(pop[2, j])
157                 Iph = float(pop[3, j])
158                 Io = float(pop[4, j])
159                 f = fitness_function(a, Rs, Rp, Iph, Io, G, Tc)
160                 Fit[j] = f
161
162         Aa = np.zeros(GEN_max, dtype=float)
163         ARs = np.zeros(GEN_max, dtype=float)
164         ARp = np.zeros(GEN_max, dtype=float)
165         AIph = np.zeros(GEN_max, dtype=float)
166         AIo = np.zeros(GEN_max, dtype=float)
167         Af = np.zeros(GEN_max, dtype=float)
168

```

```

169     for g in range(GEN_max):
170         for j in range(Np):
171
172             r[0] = int(np.floor(np.random.rand() * Np))
173             while r[0] == j:
174                 r[0] = int(np.floor(np.random.rand() * Np))
175
176             r[1] = int(np.floor(np.random.rand() * Np))
177             while r[1] == j or r[1] == r[0]:
178                 r[1] = int(np.floor(np.random.rand() * Np))
179
180             r[2] = int(np.floor(np.random.rand() * Np))
181             while r[2] == j or r[2] == r[0] or r[2] == r[1]:
182                 r[2] = int(np.floor(np.random.rand() * Np))
183
184
185             w = pop[:, r[2]] + F * (pop[:, r[0]] - pop[:, r[1]])
186
187
188             Rnd = int(np.floor(np.random.rand() * D))
189             for i in range(D):
190                 if np.random.rand() < CR or Rnd == i:
191                     x[i] = w[i]
192                 else:
193                     x[i] = pop[i, j]
194
195             # Boundary constraint
196             for i in range(D):
197                 if x[i] < L[i] or x[i] > H[i]:
198                     x[i] = L[i] + (H[i] - L[i]) * np.random.rand()
199
200             # Selection
201             a = float(x[0])
202             Rs = float(x[1])
203             Rp = float(x[2])
204             Iph = float(x[3])
205             Io = float(x[4])
206             f = fitness_function(a, Rs, Rp, Iph, Io, G, Tc)
207
208             if f <= Fit[j]:
209                 pop[:, j] = x.copy()
210                 Fit[j] = f
211
212
213             n = np.min(np.abs(Fit))
214             iBest = np.argmin(np.abs(Fit))
215             Aa[g] = pop[0, iBest]
216             ARs[g] = pop[1, iBest]
217             ARp[g] = pop[2, iBest]
218             AIph[g] = pop[3, iBest]
219             AIo[g] = pop[4, iBest]
220             Af[g] = Fit[iBest]
221
222             if Fit[iBest] <= EP:
223                 FEV = g
224                 break
225
226

```

```

227     nn = np.min(np.abs(Af))
228     Ibest = np.argmin(np.abs(Af))
229     a_average[b] = Aa[Ibest]
230     Rs_average[b] = ARs[Ibest]
231     Rp_average[b] = ARp[Ibest]
232     Iph_average[b] = AIph[Ibest]
233     Io_average[b] = AIo[Ibest]
234     f_average[b] = Af[Ibest]
235
236
237     AF = Af.reshape(-1, 1)
238     write_array_to_excel(excel_file, sheet, 'Z5', AF)
239
240
241     f_bestt = np.sum(f_average) / rr
242     a_bestt = np.sum(a_average) / rr
243     Rs_bestt = np.sum(Rs_average) / rr
244     Rp_bestt = np.sum(Rp_average) / rr
245     Iph_bestt = np.sum(Iph_average) / rr
246     Io_bestt = np.sum(Io_average) / rr
247
248
249     results = {'f_average': f_average}
250     with open(file_name, 'wb') as f:
251         pickle.dump(results, f)
252
253
254     write_array_to_excel(excel_file, sheet, 'A108', f_average.reshape(1,
255         -1))
256
257     return f_bestt, a_bestt, Rs_bestt, Rp_bestt, Iph_bestt, Io_bestt
258
259 def fitness_function(a, Rs, Rp, Iph, Io, G, Tc):
260     Nsc = 36
261     k = 1.3806503e-23
262     q = 1.60217646e-19
263     VT = (Nsc * k * Tc[0]) / q
264
265     with open('var_fitness_function.pkl', 'rb') as f:
266         data = pickle.load(f)
267         Vp = np.array(data['Vp'], dtype=float)
268         Ie = np.array(data['Ie'], dtype=float)
269
270     Ip = np.zeros_like(Vp, dtype=float)
271
272
273     a = float(a)
274     Rs = float(Rs)
275     Rp = float(Rp)
276     Iph = float(Iph)
277     Io = float(Io)
278     VT = float(VT)
279
280     for h in range(5):
281
282         exp_arg = (Vp + Ip * Rs) / (a * VT)
283         exp_term = np.exp(exp_arg)

```

```

284     numerator = Iph - Ip - Io * (exp_term - 1) - ((Vp + Ip * Rs) / Rp)
285     denominator = -1 - Io * (Rs / (a * VT)) * exp_term - (Rs / Rp)
286
287     Ip = Ip - (numerator / denominator)
288
289
290     N = len(Vp)
291     f = np.sqrt((1/N) * np.sum((Ie - Ip)**2))
292
293     return float(f)
294
295 def write_to_excel(file_path, sheet_num, cell, value):
296     """Helper function to write a single value to Excel"""
297     try:
298
299         col = ord(cell[0]) - ord('A')
300         row = int(cell[1:]) - 1
301
302
303         try:
304             df = pd.read_excel(file_path, sheet_name=f'Sheet{sheet_num}',
305                               header=None)
306         except:
307             df = pd.DataFrame()
308
309         max_row = max(row + 1, len(df))
310         max_col = max(col + 1, len(df.columns) if len(df.columns) > 0 else
311                      0)
312
313         if df.empty or df.shape[0] < max_row or df.shape[1] < max_col:
314             new_df = pd.DataFrame(index=range(max_row), columns=range(
315                                     max_col))
316             if not df.empty:
317                 new_df.iloc[:df.shape[0], :df.shape[1]] = df.values
318             df = new_df
319
320         df.iloc[row, col] = value
321
322         with pd.ExcelWriter(file_path, engine='openpyxl', mode='a',
323                             if_sheet_exists='replace') as writer:
324             df.to_excel(writer, sheet_name=f'Sheet{sheet_num}', index=False
325                          , header=False)
326     except Exception as e:
327         print(f"Error writing to Excel: {e}")
328
329 def write_array_to_excel(file_path, sheet_num, start_cell, array):
330     """Helper function to write an array to Excel"""
331     try:
332
333         col = ord(start_cell[0]) - ord('A')
334         row = int(start_cell[1:]) - 1
335
336         try:
337             df = pd.read_excel(file_path, sheet_name=f'Sheet{sheet_num}',

```

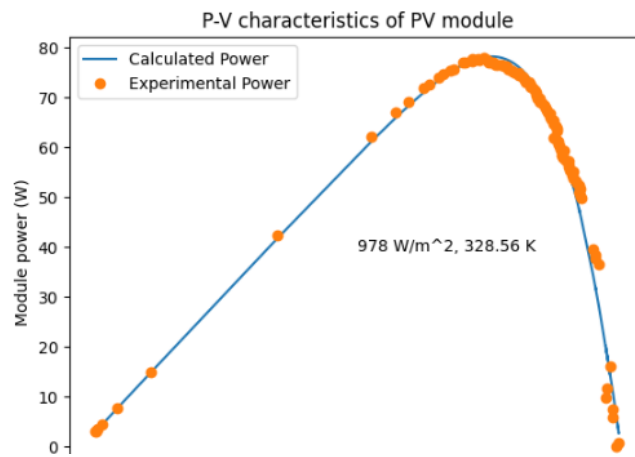
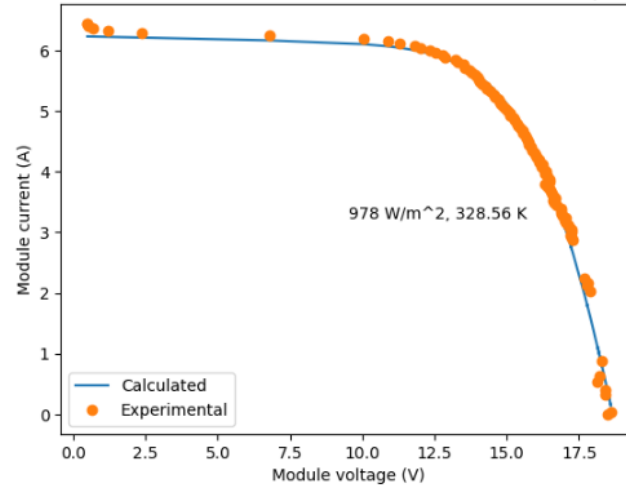
```

337         header=None)
338     except:
339         df = pd.DataFrame()
340
341     if array.ndim == 1:
342         array = array.reshape(-1, 1)
343
344     max_row = max(row + array.shape[0], len(df) if not df.empty else 0)
345     max_col = max(col + array.shape[1], len(df.columns) if not df.empty
346                  and len(df.columns) > 0 else 0)
347
348     if df.empty or df.shape[0] < max_row or df.shape[1] < max_col:
349         new_df = pd.DataFrame(index=range(max_row), columns=range(
350             max_col))
351         if not df.empty:
352             new_df.iloc[:df.shape[0], :df.shape[1]] = df.values
353         df = new_df
354
355     df.iloc[row:row+array.shape[0], col:col+array.shape[1]] = array
356
357     with pd.ExcelWriter(file_path, engine='openpyxl', mode='a',
358                        if_sheet_exists='replace') as writer:
359         df.to_excel(writer, sheet_name=f'Sheet{sheet_num}', index=False
360                    , header=False)
361     except Exception as e:
362         print(f"Error writing array to Excel: {e}")
363
364 if __name__ == "__main__":
365     main()

```

Answer:

I-V characteristics of PV module with solar radiation and ambient temperature variations



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:

```
1 # Example 3.1
2 import numpy as np
3 import matplotlib.pyplot as plt
4 from scipy.interpolate import interp1d
5
6
7 def PV_model(Va, G, TaC):
8     Iph = G * 5
9     Io = 1e-10
10    n = 1.2
11    Vt = 0.025
12
13    # Prevent exponential overflow
14    exponent = np.clip(Va / (n * Vt), -700, 700)
15    return Iph - Io * (np.exp(exponent) - 1)
16
17
18 TaC = 25          # Cell temperature
19 C = 0.5           # Step size
20 Suns_val = 0.028  # (1 G = 1000 W/m^2)
21 Va = 31          # PV voltage
22
23 Ia = PV_model(Va, Suns_val, TaC)
24 Pa = Ia * Va      # PV output power
25 Vref_new = Va + C # New reference voltage
26
27 Va_array = []
28 Pa_array = []
29
30 Suns_data = np.array([
31     [0, 0.1], [1, 0.2], [2, 0.3], [3, 0.3], [4, 0.5],
32     [5, 0.6], [6, 0.7], [7, 0.8], [8, 0.9], [9, 1.0],
33     [10, 1.1], [11, 1.2], [12, 1.3], [13, 1.4]
34 ])
35
36 x = Suns_data[:, 0]
37 y = Suns_data[:, 1]
38 xi = np.arange(1, 201)
39 interp_func = interp1d(x, y, kind='cubic', fill_value='extrapolate')
40 yi = interp_func(xi)
41
42 for i in range(14):
43     Suns = yi[i]
44     Va_new = Vref_new
45     Ia_new = PV_model(Va, Suns, TaC)
46     Pa_new = Va_new * Ia_new
47     deltaPa = Pa_new - Pa
48
49     if deltaPa > 0:
50         if Va_new > Va:
51             Vref_new = Va_new + C
52     else:
```

```

53         Vref_new = Va_new - C
54     elif deltaPa < 0:
55         if Va_new > Va:
56             Vref_new = Va_new - C
57         else:
58             Vref_new = Va_new + C
59     else:
60         V_ref = Va_new
61
62     Va = Va_new
63     Pa = Pa_new
64
65     Va_array.append(Va)
66     Pa_array.append(Pa)

```

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

Answer:

```

1  # Example 3.2
2  import numpy as np
3  from scipy.interpolate import PchipInterpolator
4
5  TaC = 25
6  C = 0.5
7  E = 0.5
8
9
10 Suns_initial = 0.045
11 Va = 31
12 def KYOCERA(Va, Suns, TaC):
13
14     return Suns * (0.02 * Va + 0.5)
15
16 Ia = KYOCERA(Va, Suns_initial, TaC)
17 Pa = Va * Ia
18 Vref_new = Va + C
19
20 Va_array = []
21 Pa_array = []
22 Pmax_array = []
23
24 Suns_matrix = np.array([
25     [0, 0.1], [1, 0.2], [2, 0.3], [3, 0.3], [4, 0.5], [5, 0.6], [6, 0.7],
26     [7, 0.8], [8, 0.9], [9, 1.0], [10, 1.1], [11, 1.2], [12, 1.3], [13,
27         1.4]
28 ])
29
30 x = Suns_matrix[:, 0]
31 y = Suns_matrix[:, 1]
32
33 xi = np.arange(1, 201)
34 interp = PchipInterpolator(x, y)
35 yi = interp(xi)

```

```

35
36 for sample in range(14):
37
38     Suns = yi[sample]
39
40
41     Va_new = Vref_new
42     Ia_new = KYOCERA(Va, Suns, TaC)
43
44
45     deltaVa = Va_new - Va
46     deltaIa = Ia_new - Ia
47
48     if deltaVa == 0:
49         if deltaIa == 0:
50             Vref_new = Va_new
51         elif deltaIa > 0:
52             Vref_new = Va_new + C
53         else:
54             Vref_new = Va_new - C
55     else:
56         if abs(deltaIa / deltaVa + Ia_new / Va_new) <= E:
57             Vref_new = Va_new
58         else:
59             if deltaIa / deltaVa > -Ia_new / Va_new + E:
60                 Vref_new = Va_new + C
61             else:
62                 Vref_new = Va_new - C
63
64
65     Va = Va_new
66     Ia = Ia_new
67     Pa = Va * Ia
68
69     Va_array.append(Va)
70     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:

```

1
2 # Example 3.3
3
4 import numpy as np
5 import matplotlib.pyplot as plt
6 from scipy.fft import fft
7
8 def main():
9
10     print('Voltage-source inverter with Sinusoidal-Pulse Width Modulated
11         output')
12     print('By Tamer Khatib')

```

```

12 print(' ')
13
14
15 Vrin = 1
16 f = float(input('The frequency of the output voltage, f = '))
17 Z = 1
18 ma = float(input('The modulation index, ma (0 < ma < 1), ma = '))
19 phi = float(input('The phase angle of the load in degrees = '))
20 fc = float(input('The frequency of the carrier signal = '))
21
22
23 phi = phi * np.pi / 180
24 R = Z * np.cos(phi)
25 L = (Z * np.sin(phi)) / (2 * np.pi * f)
26
27
28 N = int(fc / f)
29
30
31 wt = np.zeros(2*N*50)
32 ma1 = np.zeros(2*N*50)
33 Vt = np.zeros(2*N*50)
34 Vout = np.zeros(2*N*50)
35 alpha = []
36 beta = []
37
38
39 for k in range(1, 2*N+1):
40     for j in range(1, 51):
41         i = j + (k-1)*50 - 1
42         wt[i] = (j + (k-1)*50) * np.pi / (N * 50)
43
44
45         hpf = np.sign(np.sin(wt[i]))
46         if hpf == 0:
47             hpf = 1
48
49         ma1[i] = ma * abs(np.sin(wt[i]))
50
51         if k % 2 == 0:
52             Vt[i] = 0.02 * j
53             if abs(Vt[i] - ma * abs(np.sin(wt[i]))) <= 0.011:
54                 m = j
55                 beta.append(3.6 * ((k-1)*50 + m) / N)
56         else:
57             Vt[i] = 1 - 0.02 * j
58             if abs(Vt[i] - ma * abs(np.sin(wt[i]))) < 0.011:
59                 l = j
60                 alpha.append(3.6 * ((k-1)*50 + l) / N)
61
62
63         if Vt[i] > ma * abs(np.sin(wt[i])):
64             Vout[i] = 0
65         else:
66             Vout[i] = hpf * Vrin
67
68
69 if beta:

```

```

70     beta = beta[1:]
71
72
73     alpha = np.array(alpha)
74     beta = np.array(beta)
75
76
77     print(' ')
78     print('
.....')
79     print('alpha      beta      width')
80     for i in range(min(len(alpha), len(beta))):
81         print(f'{alpha[i]:.6f}    {beta[i]:.6f}    {beta[i] - alpha[i]:.6f}')
82
83
84     a = np.zeros_like(wt)
85     plt.figure(1)
86     plt.subplot(2, 1, 1)
87     plt.plot(wt, Vt, wt, ma1, wt, a)
88     plt.axis([0, 2*np.pi, -2, 2])
89     plt.ylabel('Vt, m(pu)')
90
91     plt.subplot(2, 1, 2)
92     plt.plot(wt, Vout, wt, a)
93     plt.axis([0, 2*np.pi, -2, 2])
94     plt.ylabel('Vo (pu)')
95     plt.xlabel('Radian')
96     plt.show()
97
98
99     Vo = np.sqrt(np.mean(Vout**2))
100     print('The RMS value of the output voltage = ')
101     print(Vo)
102
103     y = fft(Vout)
104     y = y[1:]
105     x = np.abs(y)
106     x = (np.sqrt(2) / len(Vout)) * x
107     print('The RMS value of the fundamental component = ')
108     print(x[0])
109
110     THDVo = np.sqrt(Vo**2 - x[0]**2) / x[0]
111
112     m = R / (2 * np.pi * f * L)
113     DT = np.pi / (N * 50)
114     C = np.zeros(2000*N)
115     C[0] = -10
116
117
118     Vout_extended = np.zeros(2000*N)
119     Vout_extended[:len(Vout)] = Vout
120
121     for i in range(100*N, 2000*N):
122         Vout_extended[i] = Vout_extended[i - 100*N]
123
124     for i in range(1, 2000*N):
125         C[i] = C[i-1] * np.exp(-m * DT) + Vout_extended[i-1] / R * (1 - np.
            exp(-m * DT))

```

```

126
127
128 CO = np.zeros(100*N)
129 for j4 in range(100*N):
130     CO[j4] = C[j4 + 1900*N]
131
132 CO2 = fft(CO)
133 CO2 = CO2[1:] # Remove first element
134 COX = np.abs(CO2)
135 COX = (np.sqrt(2) / (100*N)) * COX
136
137 CORMS = np.sqrt(np.mean(CO**2))
138 print('The RMS value of the load current =')
139 print(CORMS)
140
141 THDIO = np.sqrt(CORMS**2 - COX[0]**2) / COX[0]
142
143
144 CS = np.zeros(2000*N)
145 for j2 in range(1900*N, 2000*N):
146     if Vout_extended[j2] != 0:
147         CS[j2] = abs(C[j2])
148     else:
149         CS[j2] = 0
150
151
152 CS1 = np.zeros(100*N)
153 for j3 in range(100*N):
154     CS1[j3] = abs(CS[j3 + 1900*N])
155
156 CSRMS = np.sqrt(np.mean(CS1**2))
157 print('The RMS value of the supply current is')
158 print(CSRMS)
159
160 CSAV = np.mean(CS1)
161 print('The Average value of the supply current is')
162 print(CSAV)
163
164 CS2 = fft(CS1)
165 CS2 = CS2[1:]
166 CSX = np.abs(CS2)
167 CSX = (np.sqrt(2) / (100*N)) * CSX
168
169
170 print('Performance parameters are')
171 print('THD of output voltage:')
172 print(THDVo)
173 print('THD of output current:')
174 print(THDIO)
175
176 plt.figure(2, figsize=(12, 10))
177
178 plt.subplot(3, 2, 1)
179 plt.plot(wt, Vout[:100*N], wt, a[:100*N])
180 plt.title('Output Voltage')
181 plt.axis([0, 2*np.pi, -1.5, 1.5])
182 plt.ylabel('Vo (pu)')
183

```

```

184 plt.subplot(3, 2, 2)
185 plt.plot(x[:100])
186 plt.title('Voltage Spectrum')
187 plt.axis([0, 100, 0, 0.8])
188
189 plt.subplot(3, 2, 3)
190 plt.plot(wt, C[1900*N:2000*N], wt, a[:100*N])
191 plt.title('Output Current')
192 plt.axis([0, 2*np.pi, -1.5, 1.5])
193 plt.ylabel('Io (pu)')
194
195 plt.subplot(3, 2, 4)
196 plt.plot(COX[:100])
197 plt.title('Current Spectrum')
198 plt.axis([0, 100, 0, 0.8])
199 plt.ylabel('Ion (pu)')
200
201 plt.subplot(3, 2, 5)
202 plt.plot(wt, CS[1900*N:2000*N], wt, a[:100*N])
203 plt.axis([0, 2*np.pi, -1.5, 1.5])
204 plt.ylabel('Is (pu)')
205 plt.xlabel('Radian')
206
207 plt.subplot(3, 2, 6)
208 plt.plot(CSX[:100])
209 plt.plot(5, CSAV, '*')
210 plt.text(5, CSAV, 'Average value')
211 plt.title('Supply Current Spectrum')
212 plt.axis([0, 100, 0, 0.8])
213 plt.ylabel('Isn (pu)')
214 plt.xlabel('Harmonic Order')
215
216 plt.tight_layout()
217 plt.show()
218
219 if __name__ == "__main__":
220     main()

```

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

Answer:

```

1 # Exercise 3.4
2
3 import numpy as np
4 import matplotlib.pyplot as plt
5 from sympy import symbols, integrate
6 from sympy.utilities.lambdify import lambdify
7
8
9 Vbati = []
10 SOCi = []
11
12 # Outer loop

```

```

13 for I1 in range(5, 6):
14     t1 = 7
15     SOC1 = 0.2
16     K = 0.8
17     D = 1e-5
18     SOCm = 936
19     ns = 6
20     SOC2 = SOC1
21
22
23     for t in np.arange(0, t1 + 0.1, 0.1):
24         B = SOC2
25         if I1 <= 0:
26             V1 = (1.926 + 0.124 * B) * ns
27             R1 = (0.19 + 0.1037 / (B - 0.14)) * ns / SOCm
28         else:
29             V1 = (2 + 0.148 * B) * ns
30             R1 = (0.758 + 0.1309 / (1.06 - B)) * ns / SOCm
31             R1 = float(R1)
32
33         # Symbolic integration
34         v = symbols('v')
35         f1 = K * V1 * I1 - D * SOC2 * SOCm
36         ee = integrate(f1, (v, 0, t))
37         SOC = SOC1 + (1 / SOCm) * ee
38         SOC2 = SOC
39
40         Vbat = V1 + I1 * R1
41         Vbati.append(float(Vbat))
42         SOCi.append(float(SOC))
43
44     # Plotting
45
46     plt.figure()
47     plt.plot(Vbati)
48
49
50     plt.figure()
51     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 :

```

1
2 # Exercise 3.5
3
4 import numpy as np
5 import pandas as pd
6 import matplotlib.pyplot as plt

```



```

7
8 fileName = 'data for 3.5 .xlsx'
9 sheetName = 'Sheet1'
10
11
12 df = pd.read_excel(fileName, sheet_name=sheetName)
13
14
15 TestdayNumber = df.iloc[5845:6211, 11].values
16 Gglobal = df.iloc[5845:6211, 4].values
17 Gdiffused = df.iloc[5845:6211, 5].values
18
19 N = TestdayNumber
20 G = Gglobal
21 D = Gdiffused
22
23
24 L = 25.0
25
26 OptimumB_M = []
27 data_length = len(G)
28
29
30 days_per_month = data_length // 12
31
32 for month in range(1, 13):
33     G_M = []
34     D_M = []
35     N_M = []
36
37
38     start_idx = (month - 1) * days_per_month
39     if month == 12:
40         end_idx = data_length
41     else:
42         end_idx = month * days_per_month
43
44     for j in range(start_idx, end_idx):
45         G_M.append(G[j])
46         D_M.append(D[j])
47         N_M.append(N[j])
48
49
50     G_M = np.array(G_M)
51     D_M = np.array(D_M)
52     N_M = np.array(N_M)
53
54     days = len(G_M)
55     GBAns = []
56
57     for B in range(0, 91):
58
59         Ds = 23.45 * np.sin((360 * (284 + N_M) / 365) * (np.pi / 180))
60
61         cos_W = -1 * np.tan(L * (np.pi / 180)) * np.tan(Ds * (np.pi / 180))
62         cos_W = np.maximum(-1, np.minimum(1, cos_W))
63         W = np.arccos(cos_W) * 180 / np.pi
64

```

```

65     denominator = (np.cos(L * (np.pi / 180)) * np.cos(Ds * (np.pi /
66         180)) *
67         np.sin(W * (np.pi / 180))) + \
68         ((W * (np.pi / 180)) * np.sin(L * (np.pi / 180)) *
69         np.sin(Ds * (np.pi / 180)))
70
71     denominator = np.where(denominator == 0, np.finfo(float).eps,
72         denominator)
73
74     Rb = ((np.cos((L - B) * (np.pi / 180)) * np.cos(Ds * (np.pi / 180))
75         *
76         np.sin(W * (np.pi / 180)) +
77         (W * (np.pi / 180)) * np.sin((L - B) * (np.pi / 180)) *
78         np.sin(Ds * (np.pi / 180)))) / denominator
79
80     Rd = (1 + np.cos(B * (np.pi / 180))) / 2
81     Rr = (0.3 * (1 - np.cos(B * (np.pi / 180)))) / 2
82     F = G_M - D_M
83     BB = F * Rb
84     DB = D_M * Rd
85     RB = G_M * Rr
86     GB = BB + DB + RB
87
88     AV_GB = np.mean(GB)
89     GBAns.append(AV_GB)
90
91     GBAns = np.array(GBAns)
92     MAX_INDEX = np.argmax(GBAns)
93     MAX = GBAns[MAX_INDEX]
94     maximum_Solar_Radiation = MAX
95     optimumB1 = MAX_INDEX
96     OptimumB_M.append(optimumB1)
97
98     OptimumB_M = np.array(OptimumB_M)
99     print("OptimumB_M:", OptimumB_M)
100
101     Year_Months = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
102
103     # Plotting
104     plt.figure()
105     plt.subplot(2, 2, 1)
106     plt.plot(Year_Months, OptimumB_M, '-kd', linewidth=2.5,
107         markeredgecolor='red', markerfacecolor='red', markersize=8)
108     plt.xlim([1, 12])
109     plt.legend(['Optimum Tilt angle'], fontsize=8)
110     plt.xlabel('Month', fontsize=14)
111     plt.ylabel('Optimum Tilt angle', fontsize=14)
112     plt.grid(True)
113     plt.title('Monthly optimum tilt angle', fontsize=14)
114
115     plt.gca().tick_params(labelsize=12)
116     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:

```

1
2 # Example 3.6
3
4 import pandas as pd
5 import numpy as np
6 import matplotlib.pyplot as plt
7
8
9 file = 'PV Modeling Book Data Source.xls'
10 sheet = 'Source 6'
11 rows = list(range(9, 4389))
12
13 G = pd.read_excel(file, sheet_name=sheet, usecols="I", skiprows=rows[0]-1,
14                  nrows=len(rows)).squeeze()
15 Tc = pd.read_excel(file, sheet_name=sheet, usecols="J", skiprows=rows[0]-1,
16                  nrows=len(rows)).squeeze()
17 Vm = pd.read_excel(file, sheet_name=sheet, usecols="L", skiprows=rows[0]-1,
18                  nrows=len(rows)).squeeze()
19 Im = pd.read_excel(file, sheet_name=sheet, usecols="M", skiprows=rows[0]-1,
20                  nrows=len(rows)).squeeze()
21
22 Ns = 5
23 Np = 4

```

```

20 Am = 0.9291
21 A = Ns * Np * Am
22 Va = Ns * Vm
23 Ia = Np * Im
24 Vpv = Va
25 Ipv = Ia
26 Pao = Va * Ia
27 Pai = A * G
28 effa = np.divide(Pao, Pai, out=np.zeros_like(Pao), where=Pai != 0)
29
30 # === DC Motor ===
31 Va = 0.95 * Va
32 Ia = 0.9 * Ia
33 Ra = 0.8
34 Km = 0.175
35 Ebb = Va - Ra * Ia
36 Eb = np.where(Ebb >= 0, Ebb, 0)
37 Ia = np.where(Ebb >= 0, Ia, 0)
38 Va = np.where(Ebb >= 0, Va, 0)
39
40 Tm = Km * Ia
41 Tm1 = np.where(Tm == 0, 1, Tm)
42 Pdev = Eb * Ia
43
44 # === Pump Model Constants ===
45 Rou = 1000
46 g = 9.81
47 d1 = 33.5e-3
48 d2 = 160e-3
49 beta1 = np.deg2rad(38)
50 beta2 = np.deg2rad(33)
51 b1 = 5.4e-3
52 b2 = 2.2e-3
53
54 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))))
55 Omega = np.sqrt(np.divide(Km * Ia, Kp, out=np.zeros_like(Ia), where=Kp !=
    0))
56 Pmo = Pdev
57 Pmi = Pao * 0.9
58 Pmi_safe = np.where(Pmi == 0, 1, Pmi)
59 effm = Pmo / Pmi_safe
60
61 # === Pump ===
62 Tp = Tm
63 Eh = Tp * Omega
64 Ppo = np.where(Ebb >= 0, Eh, 0)
65 Ppi = Pmo
66 Ppi_safe = np.where(Ppi == 0, 1, Ppi)
67 effpp = Ppo / Ppi_safe
68 effp = np.where(effpp <= 0.95, effpp, 0)
69
70 # === Water Flow Rate ===
71 h1 = 20
72 h2 = 0.1444
73 QQ = []
74 for i in range(len(Eh)):
75     r1 = h2 * 2.725

```

```

76     r2 = 0
77     r3 = h1 * 2.725
78     r4 = -Eh[i]
79     roots = np.roots([r1, r2, r3, r4])
80     real_roots = [r.real for r in roots if np.isreal(r) and r.real > 0]
81     QQ.append(real_roots[0] if real_roots else 0)
82
83     QQ = np.array(QQ)
84     Q2 = np.where(QQ == 0, 1, QQ)
85     H = Eh / (2.725 * Q2)
86
87
88     effsub = effm * effp
89     effoverall = effa * effsub
90
91     Q = np.insert(QQ, 0, 0)
92     demand = 2.5
93     Cn = 50
94     Cr = np.zeros(len(Q))
95     Qexcess_pv = np.zeros(len(Q))
96     Qexcess_s = np.zeros(len(Q))
97     SOC = np.zeros(len(Q))
98     Qdef_pv = np.zeros(len(Q))
99     Qdeficit_s = np.zeros(len(Q))
100
101     X = Q[1:] - demand
102     Qdef_pv[1:] = np.where(X < 0, -X, 0)
103     Qexcess_pv[1:] = np.where(X >= 0, X, 0)
104
105     for i in range(1, len(Q)):
106         Cr[i] = max(0, Cr[i-1] + X[i-1])
107         SOC[i] = min(1, Cr[i] / Cn)
108         if SOC[i] >= 1:
109             Qexcess_s[i] = Cr[i] - Cn
110             Cr[i] = Cn
111         Qdeficit_s[i] = max(0, -1 * (Cr[i-1] + X[i-1]))
112
113     Q = Q[1:]
114     Qexcess_pv = Qexcess_pv[1:]
115     Qexcess_s = Qexcess_s[1:]
116     Qdeficit_s = Qdeficit_s[1:]
117     Qdef_pv = Qdef_pv[1:]
118     Cres = Cr[1:]
119     SOC = SOC[1:]
120
121     D = np.full(len(Q), demand)
122     LLPh = Qdeficit_s / D
123     LLP = np.sum(Qdeficit_s) / np.sum(D)
124
125     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 (E_{net}) is calculated. The maximum state of charge (SOC) of the battery is given to the variable SOC_i as an initial value. In addition to that, matrices are defined so as to contain the results of battery state of charge (SOC_f), 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 E_{net} array. Then the energy difference is added to the variable SOC_i . Here, if the result SOC is higher than SOC_{max} , 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:

```
1 #Exercise 4.1
2 import numpy as np
```

```

3 import pandas as pd
4
5
6 file_name = 'PV Modeling Book Data Source.xls'
7 sheet_name = 'Source 7'
8
9
10 G = pd.read_excel(file_name, sheet_name=sheet_name, usecols="A", nrows
    =8761)
11 T = pd.read_excel(file_name, sheet_name=sheet_name, usecols="B", nrows
    =8761)
12 L = pd.read_excel(file_name, sheet_name=sheet_name, usecols="D", nrows
    =8761)
13
14 # Convert to numeric values safely
15 G = pd.to_numeric(G.squeeze(), errors='coerce')
16 T = pd.to_numeric(T.squeeze(), errors='coerce')
17 L = pd.to_numeric(L.squeeze(), errors='coerce')
18
19
20 # (2) System specifications
21 PV_Wp = 2500 # PV array capacity in Watts
22 Battery_SOCmax = 1400 # Battery max energy in Wh
23 PV_eff = 0.16
24 V_B = 12
25 Inv_RP = 2500
26 DOD = 0.8
27 Charge_eff = 0.8
28 Alpha = 0.05
29 Wire_eff = 0.98
30
31 SOCmax = Battery_SOCmax
32 SOCmin = SOCmax * (1 - DOD)
33
34 # (3.1) Simulation of the SAPV system
35 P_Ratio = (PV_Wp * (G / 1000)) / Inv_RP
36 Inv_eff = 97.644 - (P_Ratio * 1.995) - (0.445 / P_Ratio.replace(0, np.nan))
37 Inv_eff = Inv_eff.fillna(0) # Replace NaNs from divide-by-zero with 0
38
39 # E_PV = ((PV_Wp*(G/1000)) - (Alpha*(T-25))) * Wire_eff * Inv_eff
40 E_PV = ((PV_Wp * (G / 1000)) - (Alpha * (T - 25))) * Wire_eff * Inv_eff
41
42 E_net = E_PV - L
43
44
45 SOCi = SOCmax
46 SOCf = []
47 Deff = []
48 Dampf = []
49
50
51 for ED in E_net:
52     SOC = ED + SOCi
53
54     if SOC > SOCmax:
55         Dampi = SOC - SOCmax
56         Defi = 0
57         SOCi = SOCmax

```



```

58
59     elif SOC < SOCmin:
60         SOCi = SOCmin
61         Defi = SOC - SOCmin
62         Dampi = 0
63
64     else:
65         Defi = 0
66         Dampi = 0
67         SOCi = SOC
68
69     SOCf.append(SOCi)
70     Deff.append(Defi)
71     Dampf.append(Dampi)
72
73 # outputs
74 SOCf = np.array(SOCf)
75 Deff = np.array(Deff)
76 Dampf = np.array(Dampf)
77 SOC_per = SOCf / SOCmax
78 LLP_calculated = abs(np.sum(Deff)) / np.sum(L)
79
80 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 charging 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:

```
1
2 # Exercise 4.2
3
4 import numpy as np
5
6 # Sample data (replace with real values)
7 E_net = np.random.uniform(-2000, 2000, size=8760) # Net energy from PV (Wh)
8
9 L = np.random.uniform(1000, 2000, size=8760) # Load profile (Wh)
10 SOCmax = 1400 # Max battery SOC (Wh)
11 SOCmin = SOCmax * (1 - 0.8) # SOCmin based on DOD = 0.8
12
13 E_Capacity = 1500 # Diesel generator
14     capacity (Wh/hour)
15
16 SOCf = []
17 Deff = []
18 Dumpf = []
19 E_Geni = []
20
21 SOCi = SOCmax
22
23 for i in range(len(E_net)):
24     SOC = E_net[i] + SOCi
25
26     if SOC > SOCmax:
```

```

24     Dumpi = SOC - SOCmax
25     Defi = 0
26     SOCi = SOCmax
27     E_Gen = 0
28
29     elif SOC < SOCmin:
30         Old_Defi = (SOC - SOCmin) + E_Capacity
31
32         if Old_Defi >= 0:
33             SOCi = SOCmin + Old_Defi
34
35             if SOCi <= SOCmax:
36                 Defi = 0
37                 Dumpi = 0
38                 E_Gen = abs(Old_Defi) + (SOCi - SOCmin)
39             else:
40                 Defi = 0
41                 Dumpi = 0
42                 E_Gen = abs(Old_Defi) + (SOCi - SOCmin) - (SOCi - SOCmax)
43                 SOCi = SOCmax
44         else:
45             SOCi = SOCmin
46             Defi = abs(Old_Defi)
47             Dumpi = 0
48             E_Gen = E_Capacity
49
50     else:
51         SOCi = SOC + E_Capacity
52         if SOCi <= SOCmax:
53             Defi = 0
54             Dumpi = 0
55             E_Gen = E_Capacity
56         else:
57             Defi = 0
58             Dumpi = 0
59             E_Gen = E_Capacity - (SOCi - SOCmax)
60             SOCi = SOCmax
61
62     SOCf.append(SOCi)
63     Deff.append(Defi)
64     Dumpf.append(Dumpi)
65     E_Geni.append(E_Gen)
66
67 SOCf = np.array(SOCf)
68 Deff = np.array(Deff)
69 Dumpf = np.array(Dumpf)
70 E_Geni = np.array(E_Geni)
71
72 SOC_per = SOCf / SOCmax
73 LLP_calculated = abs(np.sum(Deff)) / np.sum(L)
74
75 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:

```
1
2 # Exercise 4.3
3
4 import pandas as pd
5 import matplotlib.pyplot as plt
6
7
8 file_path = 'PV Modeling Book Data Source.xls'
9 sheet_name = 'Source 2'
10
11
12 df = pd.read_excel(file_path, sheet_name=sheet_name, usecols="B:C",
13                    skiprows=1, nrows=1440)
14 I_L = df.iloc[:, 0].values
15 I_PV = df.iloc[:, 1].values
16
17 # System specifications
18 I_Diesel = 5
19 SOCmax = 1000
20 SOCi = SOCmax
21 V_B = 2
22 DOD = 0.8
23 SOCmin = SOCmax * (1 - DOD)
24 t1 = 1
25 SOC1 = 1
26 SOC3 = 0.3
27 K = 0.8
28 D = 1e-5
29 ns = 6
30 SOC2 = SOC1
31 W = 0
32 A = 0.2461
33 T = 0.081451
34 n = 0
35
36 # Storage
37 SOCf = []
38 I_Loadf = []
39 I_Chargef = []
40 I_Dischargef = []
41 I_Batteryf = []
42 I_Deficitf = []
43 I_Dampf = []
44 I_Dieself = []
45 F_Cf = []
46
47 # Simulation
48 I_net = I_PV - I_L
49 SOC_track = []
```

```

50 for i in range(len(I_L)):
51     if len(SOCf) == 0:
52         prev_SOC = SOC1
53     else:
54         prev_SOC = SOCf[-1]
55
56     if I_net[i] == 0 and n == 0:
57         I_Loadi = I_PV[i]
58         I_Dampi = I_Batteryi = I_Chargei = I_Deficiti = I_Dieseli =
59             I_Dischargei = F_Ci = 0
60         if i == 0 or W == 0:
61             SOCi = SOC1
62         elif W == 1:
63             SOCi = prev_SOC
64
65     elif I_net[i] > 0 and n == 0:
66         I_Loadi = I_L[i]
67         if i == 0 or W == 0:
68             SOCi = SOC1
69             I_Dampi = I_net[i]
70             I_Chargei = I_Dischargei = I_Batteryi = I_Deficiti = I_Dieseli
71                 = F_Ci = 0
72         elif W == 1:
73             if prev_SOC >= SOC1:
74                 I_Dampi = I_net[i]
75                 SOCi = prev_SOC
76                 I_Dischargei = I_Batteryi = I_Deficiti = I_Dieseli =
77                     I_Chargei = F_Ci = 0
78             else:
79                 I_Chargei = I_net[i]
80                 I_Dischargei = I_Batteryi = I_Deficiti = I_Dieseli =
81                     I_Dampi = F_Ci = 0
82                 SOCi = prev_SOC + abs(SOC1 - (SOC1 + I_Chargei * K))
83
84     elif I_net[i] < 0 or (I_net[i] > 0 and n > 0):
85         if W == 0 and n == 0:
86             I_Dischargei = I_L[i] - I_PV[i]
87             I_Loadi = I_PV[i] + I_Dischargei
88             I_Batteryi = I_Dischargei
89             I_Chargei = I_Deficiti = I_Dampi = I_Dieseli = F_Ci = 0
90             SOCi = SOC1 - abs(SOC1 - (SOC1 - I_Dischargei * K))
91             W += 1
92         elif W == 1:
93             if prev_SOC > SOC3 and n == 0:
94                 I_Dischargei = I_L[i] - I_PV[i]
95                 I_Loadi = I_L[i]
96                 I_Batteryi = I_Dischargei
97                 I_Chargei = I_Deficiti = I_Dampi = I_Dieseli = F_Ci = 0
98                 SOCi = prev_SOC - abs(SOC1 - (SOC1 - I_Dischargei * K))
99             elif prev_SOC <= SOC3 or n > 0:
100                 if I_Diesel >= I_L[i]:
101                     I_Loadi = I_L[i]
102                     I_Dieseli = I_Loadi
103                     I_Dischargei = I_Deficiti = 0
104                 elif prev_SOC < SOC1:
105                     I_Chargei = I_PV[i]
106                     I_Dampi = 0
107                     SOCi = prev_SOC + abs(SOC1 - (SOC1 + I_Chargei * K))

```

```

104         else:
105             I_Chargei = 0
106             I_Dampi = I_PV[i]
107             SOCi = prev_SOC
108             F_Ci = A * (230 * ((I_Dieseli / 60) / 1000)) + T * (230
109                 * ((I_Diesel / 60) / 1000))
110             n += 1
111             elif I_Diesel < I_L[i]:
112                 I_Deficiti = I_L[i]
113
114             if n == 4:
115                 n = 0
116
117             SOCf.append(SOCi)
118             I_Loadf.append(I_Loadi)
119             I_Chargef.append(I_Chargei)
120             I_Dischargef.append(I_Dischargei)
121             I_Batteryf.append(I_Batteryi)
122             I_Deficitf.append(I_Deficiti)
123             I_Dampf.append(I_Dampi)
124             I_Dieself.append(I_Dieseli)
125             F_Cf.append(F_Ci)
126
127 # Final results
128 Excess_energy = ((sum(I_Dampf) / 60) * 220) / 1000
129 Diesel_consumption = sum(F_Cf)
130 Enrgy_Deficit = ((sum(I_Deficitf) / 60) * 220) / 1000
131 Enrgy_Discharge = ((sum(I_Dischargef) / 60) * 220) / 1000
132
133 # Plotting
134 fig1, axs1 = plt.subplots(4, 1, figsize=(10, 8))
135 axs1[0].plot(I_L, label='Load')
136 axs1[0].plot(I_PV, label='PV', color='red')
137 axs1[0].set_ylabel('Current (Amp)')
138 axs1[0].legend()
139
140 axs1[1].plot(I_Dischargef)
141 axs1[1].set_ylabel('Discharge')
142
143 axs1[2].plot(I_Chargef)
144 axs1[2].set_ylabel('Charge')
145
146 axs1[3].plot(SOCf)
147 axs1[3].set_ylabel('SOC (%)')
148
149 fig2, axs2 = plt.subplots(3, 1, figsize=(10, 6))
150 axs2[0].plot(I_Dieself)
151 axs2[0].set_ylabel('Diesel')
152
153 axs2[1].plot(I_Deficitf, color='red')
154 axs2[1].set_ylabel('Deficit')
155
156 axs2[2].plot(I_Dampf)
157 axs2[2].set_ylabel('Damp')
158 axs2[2].set_xlabel('Time (min)')
159
160 plt.tight_layout()

```

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

Answer:

```
1 #Exercise 4.4
2
3 import numpy as np
4 import pandas as pd
5 import matplotlib.pyplot as plt
6 from scipy import integrate
7 import sympy as sp
8
9 plt.close('all')
10
11
12 fileName = 'Load for Ammar.xls'
13 sheetName1 = 'Sheet2'
14 sheetName2 = 'Sheet3'
15
16
17 df1 = pd.read_excel(fileName, sheet_name=sheetName1, usecols='A:B',
18                     skiprows=1, nrows=1440)
19 I_PV = df1.iloc[:, 0].values
20 I_L = df1.iloc[:, 1].values
21
22 I_Diesel = 5
23 SOCmax = 1000
24 SOCi = SOCmax
25 V_B = 2
26 DOD = 0.8
27 SOCmin = SOCmax * (1 - DOD)
28 t1 = 1
29 SOC1 = 1
30 SOC3 = 0.3
31 K = 0.8
32 D = 1e-5
33 ns = 6
34 SOC2 = SOC1
35 W = 0
36 A = 0.2461
37 T = 0.081451
38 n = 0
39
40 SOCfinal = []
41 Dieselfinal = []
42 SOCf = []
43 I_Loadf = []
44 I_Chargef = []
45 I_Dischargef = []
46 I_Batteryf = []
47 I_Deficitf = []
```

```

48 I_Dampf = []
49 I_Dieself = []
50 F_Cf = []
51
52 L = len(I_L)
53 I_net = np.zeros(L)
54 SOC = np.zeros(L)
55
56 for i in range(L):
57     I_net[i] = I_PV[i] - I_L[i]
58
59     if I_net[i] == 0:
60         if n == 0:
61             I_Loadi = I_PV[i]
62             I_Dampi = 0
63             I_Batteryi = 0
64             I_Chargei = 0
65             I_Deficiti = 0
66             I_Dieseli = 0
67             I_Dischargei = 0
68             F_Ci = 0
69
70             if i == 0:
71                 SOCi = SOC1
72             elif W == 0:
73                 SOCi = SOC1
74             elif W == 1:
75                 SOCi = SOCf[i-1]
76
77
78         elif I_net[i] > 0 and n == 0:
79             I_Loadi = I_L[i]
80
81             if i == 0:
82                 SOCi = SOC1
83                 I_Dampi = I_net[i]
84                 I_Chargei = 0
85                 I_Dischargei = 0
86                 I_Batteryi = 0
87                 I_Deficiti = 0
88                 I_Dieseli = 0
89                 F_Ci = 0
90
91             elif i > 0:
92                 if W == 0:
93                     SOCi = SOC1
94                     I_Dampi = I_net[i]
95
96                 elif W == 1:
97                     if SOCf[i-1] >= SOC1:
98                         I_Dampi = I_net[i]
99                         SOCi = SOCf[i-1]
100                         I_Dischargei = 0
101                         I_Batteryi = 0
102                         I_Deficiti = 0
103                         I_Dieseli = 0
104                         I_Chargei = 0
105                         F_Ci = 0

```



```

106
107         elif SOCf[i-1] < SOC1:
108             I_Chargei = I_net[i]
109             I_Dischargei = 0
110             I_Batteryi = 0
111             I_Deficiti = 0
112             I_Dieseli = 0
113             I_Dampi = 0
114             F_Ci = 0
115
116         for t in [1]:
117             B = SOC2
118             V1 = (2 + 0.148 * B) * ns
119             R1 = (0.758 + 0.1309 / (1.06 - B)) * ns / SOCmax
120             R1 = float(R1)
121
122
123             v = sp.Symbol('v')
124             integrand = K * V1 * I_net[i] - D * SOC2 * SOCmax
125             ee = float(sp.integrate(integrand, (v, 0, t)))
126             SOC_temp = SOC1 + (1/SOCmax) * ee
127             SOC2 = SOC_temp
128
129             SOC2 = float(SOC2)
130             SOC[i] = SOCf[i-1] + abs(SOC1 - SOC2)
131             SOCi = SOC[i]
132
133     elif I_net[i] < 0 or (I_net[i] > 0 and n >= 0):
134         if W == 0:
135             if n == 0:
136                 I_Dischargei = I_L[i] - I_PV[i]
137                 I_Loadi = I_PV[i] + I_Dischargei
138                 I_Batteryi = I_Dischargei
139                 I_Chargei = 0
140                 I_Deficiti = 0
141                 I_Dampi = 0
142                 I_Dieseli = 0
143                 F_Ci = 0
144
145                 for t in [1]:
146                     B = SOC2
147                     V1 = (1.926 + 0.124 * B) * ns
148                     R1 = (0.19 + 0.1037 / (B - 0.14)) * ns / SOCmax
149
150
151                     v = sp.Symbol('v')
152                     integrand = K * V1 * I_net[i] - D * SOC2 * SOCmax
153                     ee = float(sp.integrate(integrand, (v, 0, t)))
154                     SOC_temp = SOC1 + (1/SOCmax) * ee
155                     SOC2 = SOC_temp
156
157                     SOC2 = float(SOC2)
158                     SOC[i] = SOC2
159                     SOCi = SOC[i]
160                     W = W + 1
161
162             elif W == 1:
163                 if SOCf[i-1] > SOC3 and n == 0:

```

```

164     I_Dischargei = I_L[i] - I_PV[i]
165     I_Loadi = I_L[i]
166     I_Batteryi = I_Dischargei
167     I_Chargei = 0
168     I_Deficiti = 0
169     I_Dampi = 0
170     I_Dieseli = 0
171     F_Ci = 0
172
173     for t in [1]:
174         B = SOC2
175         V1 = (1.926 + 0.124 * B) * ns
176         R1 = (0.19 + 0.1037 / (B - 0.14)) * ns / SOCmax
177
178         v = sp.Symbol('v')
179         integrand = K * V1 * I_net[i] - D * SOC2 * SOCmax
180         ee = float(sp.integrate(integrand, (v, 0, t)))
181         SOC_temp = SOC1 + (1/SOCmax) * ee
182         SOC2 = SOC_temp
183
184     SOC2 = float(SOC2)
185     SOC[i] = SOCf[i-1] - abs(SOC1 - SOC2)
186     SOCi = SOC[i]
187
188 elif SOCf[i-1] <= SOC3 or n >= 0:
189     if I_Diesel >= I_L[i]:
190         I_Loadi = I_L[i]
191         I_Dieseli = I_Diesel
192         I_Dischargei = 0
193         I_Deficiti = 0
194
195     if SOCf[i-1] < SOC1:
196         I_Chargei = I_PV[i] + I_Dieseli - I_L[i]
197         I_Dampi = 0
198
199     for t in [1]:
200         B = SOC2
201         V1 = (2 + 0.148 * B) * ns
202         R1 = (0.758 + 0.1309 / (1.06 - B)) * ns /
203             SOCmax
204         R1 = float(R1)
205
206         v = sp.Symbol('v')
207         integrand = K * V1 * I_Chargei - D * SOC2 *
208             SOCmax
209         ee = float(sp.integrate(integrand, (v, 0, t)))
210         SOC_temp = SOC1 + (1/SOCmax) * ee
211         SOC2 = SOC_temp
212
213     SOC2 = float(SOC2)
214     SOC[i] = SOCf[i-1] + abs(SOC1 - SOC2)
215     SOCi = SOC[i]
216 else:
217     I_Chargei = 0
218     I_Dampi = I_PV[i] + I_Dieseli - I_L[i]
219     SOCi = SOC1

```

```

220
221         F_C = A * (220 * ((I_Diesel / 60) / 1000)) + \
222             T * (220 * ((I_Diesel / 60) / 1000))
223         F_Ci = F_C
224         n = n + 1
225
226     elif I_Diesel < I_L[i] and SOCf[i-1] >= 0.8:
227         I_Dischargei = I_L[i] - I_Diesel
228         I_Loadi = I_L[i]
229         I_Batteryi = I_Dischargei
230         I_Chargei = 0
231         I_Deficiti = 0
232         I_Dampi = 0
233         I_Dieseli = I_Diesel
234         F_Ci = 0
235
236     for t in [1]:
237         B = SOC2
238         V1 = (1.926 + 0.124 * B) * ns
239         R1 = (0.19 + 0.1037 / (B - 0.14)) * ns / SOCmax
240
241
242         v = sp.Symbol('v')
243         integrand = K * V1 * I_Dischargei - D * SOC2 *
                SOCmax
244         ee = float(sp.integrate(integrand, (v, 0, t)))
245         SOC_temp = SOC1 + (1/SOCmax) * ee
246         SOC2 = SOC_temp
247
248         SOC2 = float(SOC2)
249         SOC[i] = SOCf[i-1] - abs(SOC1 - SOC2)
250         SOCi = SOC[i]
251
252     elif I_Diesel < I_L[i] and SOCf[i-1] <= 0.3:
253         I_Deficiti = I_L[i]
254
255     if SOC[i] >= 0.9:
256         n = 0
257
258
259     SOCf.append(SOCi)
260     I_Loadf.append(I_Loadi)
261     I_Chargef.append(I_Chargei)
262     I_Dischargef.append(I_Dischargei)
263     I_Batteryf.append(I_Batteryi)
264     I_Deficitf.append(I_Deficiti)
265     I_Dampf.append(I_Dampi)
266     I_Dieself.append(I_Dieseli)
267     F_Cf.append(F_Ci)
268
269
270 SOCf = np.array(SOCf)
271 I_Loadf = np.array(I_Loadf)
272 I_Chargef = np.array(I_Chargef)
273 I_Dischargef = np.array(I_Dischargef)
274 I_Batteryf = np.array(I_Batteryf)
275 I_Deficitf = np.array(I_Deficitf)
276 I_Dampf = np.array(I_Dampf)

```

```

277 I_Dieself = np.array(I_Dieself)
278 F_Cf = np.array(F_Cf)
279
280
281 DD = SOCf
282 SSS = np.sum(DD < 0.3)
283 LL = SSS
284
285 Excess_energy = ((np.sum(I_Dampf) / 60) * 220) / 1000
286 Diesel_consumption = np.sum(F_Cf) # Litres
287 Enrgy_Deficit = ((np.sum(I_Deficitf) / 60) * 220) / 1000 # kWh
288 Enrgy_Discharge = ((np.sum(I_Dischargef) / 60) * 220) / 1000
289
290 # Plotting
291 plt.figure(figsize=(12, 10))
292 plt.subplot(4, 1, 1)
293 plt.plot(I_L, label='Load')
294 plt.plot(I_PV, 'red', label='PV')
295 plt.ylabel('Current (Amp)')
296 plt.legend()
297
298 plt.subplot(4, 1, 2)
299 plt.plot(I_Dischargef)
300 plt.ylabel('Discharge')
301
302 plt.subplot(4, 1, 3)
303 plt.plot(I_Chargef)
304 plt.ylabel('Charge')
305
306 plt.subplot(4, 1, 4)
307 plt.plot(SOCf)
308 plt.ylabel('SOC (%)')
309
310 plt.tight_layout()
311
312 plt.figure(figsize=(12, 8))
313 plt.subplot(3, 1, 1)
314 plt.plot(I_Dieself)
315 plt.ylabel('Diesel')
316
317 plt.subplot(3, 1, 2)
318 plt.plot(I_Deficitf, 'red')
319 plt.ylabel('Deficit')
320
321 plt.subplot(3, 1, 3)
322 plt.plot(I_Dampf)
323 plt.ylabel('Damp')
324 plt.xlabel('Time (min)')
325
326 plt.tight_layout()
327 plt.show()
328
329 print(f"Excess energy: {Excess_energy} kWh")
330 print(f"Diesel consumption: {Diesel_consumption} Litres")
331 print(f"Energy Deficit: {Enrgy_Deficit} kWh")
332 print(f"Energy Discharge: {Enrgy_Discharge} kWh")
333 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
5kW	-0.2418	-1.127	96.10

Figure 11: TABLE 5.1 Inverter Models Coefficients

Answer:

```

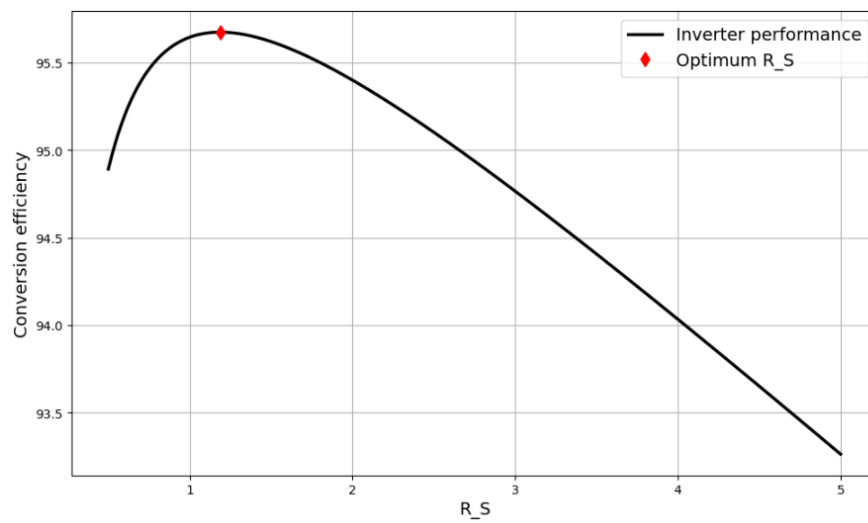
1 # Example 5.1
2
3 import pandas as pd
4 import numpy as np
5 import matplotlib.pyplot as plt
6
7 # Load Excel data
8 file_name = 'Malaysian Daily Solar Data.xls'
9 sheet_name = 'Kuala Lumpur'
10 E_Solar = pd.read_excel(file_name, sheet_name=sheet_name, usecols="E",
11                          skiprows=7306, nrows=366)
12
13 Solar_Rad = (E_Solar.values.flatten() / 12) * 1000
14
15 AV_InvEff = []
16 Rs = []
17
18 for Rsi in np.arange(0.5, 5.01, 0.01):
19     Rs.append(Rsi)
20     Pm = 2
21     InvC = Pm / Rsi
22     P_Ratio = (Pm * (Solar_Rad / 1000)) / InvC
23     InvEffi = 97.644 - (P_Ratio * 1.995) - (0.445 / P_Ratio)
24
25     P = InvEffi[InvEffi >= 0]
26     Av = np.sum(P) / len(P)
27     AV_InvEff.append(Av)
28
29 # Convert to numpy arrays for plotting
30 Rs = np.array(Rs)
31 AV_InvEff = np.array(AV_InvEff)
32
33 # Plotting
34 plt.figure(figsize=(10, 6))
35 plt.plot(Rs, AV_InvEff, '-k', linewidth=2.5)
36
37 # Find optimal point
38 max_eff = np.max(AV_InvEff)

```

```

37 max_index = np.argmax(AV_InvEff)
38 opt_rs = 0.5 + max_index * 0.01
39
40 plt.plot(opt_rs, max_eff, 'd', markerfacecolor='red', markeredgecolor='red'
41         , markersize=8)
42
43 plt.xlabel('R_S', fontsize=14)
44 plt.ylabel('Conversion efficiency', fontsize=14)
45 plt.legend(['Inverter performance', 'Optimum R_S'], fontsize=14)
46 plt.grid(True)
47 plt.tight_layout()
48 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:

```
1 # Exercise 6.1
2
3 import numpy as np
4 import pandas as pd
5 from scipy import integrate
6 import sympy as sp
7 import warnings
8 warnings.filterwarnings('ignore')
9
10
11 fileName = 'PV Modeling Book Data Source.xls'
12 sheetName = 'Source 7'
13
14
15 df = pd.read_excel(fileName, sheet_name=sheetName)
16 I_L = df.iloc[9:8769, 3].values
17 T = df.iloc[9:8769, 1].values
18 S = df.iloc[9:8769, 0].values
19
20 NOCT = 43.6
21 Alpha = 0.068
22 IPV_M = 6.89
23
24
25 T_cell = np.zeros(len(S))
26 I_PV = np.zeros(len(S))
27
28 for k in range(len(S)):
29     T_cell[k] = (T[k] + ((NOCT - 20) / 800) * S[k])
30     I_PV[k] = ((S[k] / 1000)) + (Alpha * (T_cell[k] - 25))
31
32 for zzz in range(len(I_L)):
33     if I_PV[zzz] < 0:
34         I_PV[zzz] = 0
35     else:
36         I_PV[zzz] = I_PV[zzz]
37
38 k = 0
39 zzz = 0
40
41
42 PV_eff = 0.14
43 Wire_eff = 0.98
44 INV_eff = 0.95
45 V_sys = 230
46 PSH = (np.mean(S) * 12) / 1000
47 V_B = 12
```

```

48 DCharge_eff = 0.8
49 DOD = 0.8
50 Alpha = 0.068
51 PV_WP = 120
52
53
54 P_L = np.zeros(len(I_L))
55 for i in range(len(I_L)):
56     P_L[i] = I_L[i] * 230
57
58 A_PV = 1.408 * 0.56
59
60
61 E_L = np.sum(I_L * 230)
62
63 if np.isnan(E_L) or E_L <= 0:
64     print("Error: Invalid energy consumption calculation")
65     print(f"E_L = {E_L}")
66     print(f"I_L stats: min={np.min(I_L)}, max={np.max(I_L)}, mean={np.mean(I_L)}")
67     E_L = 1000
68
69 if np.isnan(PSH) or PSH <= 0:
70     print("Error: Invalid PSH calculation")
71     print(f"PSH = {PSH}")
72     print(f"S stats: min={np.min(S)}, max={np.max(S)}, mean={np.mean(S)}")
73     PSH = 5
74
75 N_PV = np.ceil((E_L / (PV_eff * INV_eff * Wire_eff * PSH * 1000 * A_PV)))
76
77 if np.isnan(N_PV) or N_PV <= 0:
78     print("Error: Invalid N_PV calculation")
79     print(f"N_PV = {N_PV}")
80     N_PV = 10
81
82 N_PVmin = int(np.ceil(N_PV / 3))
83 N_PVmax = int(5 * N_PV)
84 nh = 2
85 C_battery = (E_L * nh) / (DOD * DCharge_eff)
86 C_batterymin = C_battery / 3
87 C_batterymax = 5 * C_battery
88
89 print(f"E_L = {E_L}")
90 print(f"PSH = {PSH}")
91 print(f"N_PV = {N_PV}")
92 print(f"N_PVmin = {N_PVmin}, N_PVmax = {N_PVmax}")
93 print(f"C_battery = {C_battery}")
94 print(f"C_batterymin = {C_batterymin}, C_batterymax = {C_batterymax}")
95
96 SOC = []
97 I_Load = []
98 I_Charge = []
99 I_Discharge = []
100 I_Deficit = []
101 I_Damp = []
102 I_Battery = []
103 t = 1
104 Vmp = 17.4

```

```

105 NOCT = 43.6
106 K = 0.8
107 D = 1e-5
108 ns = 6
109 SOC1 = 1
110 SOC2 = SOC1
111 SOC3 = 0.2
112
113 C_batteryf = []
114 C_PVf = []
115 LLP_calculated = []
116
117 x = 0
118 for m in range(N_PVmin, N_PVmax + 1):
119     y = 0
120     C_batteryf_row = []
121     C_PVf_row = []
122     LLP_calculated_row = []
123
124     for n in np.arange(C_batterymin, C_batterymin + 1, 500):
125         SOCmax = n
126         SOCmin = SOCmax * (1 - DOD)
127         w = 0
128         SOC = np.zeros(len(I_L))
129         I_Load = np.zeros(len(I_L))
130         I_Charge = np.zeros(len(I_L))
131         I_Discharge = np.zeros(len(I_L))
132         I_Deficit = np.zeros(len(I_L))
133         I_Damp = np.zeros(len(I_L))
134         I_Battery = np.zeros(len(I_L))
135         I_net = np.zeros(len(I_L))
136
137         for k in range(len(I_L)):
138             I_net[k] = I_PV[k] - I_L[k]
139
140
141             if SOCmax > 0:
142
143                 if I_net[k] == 0:
144                     I_Load[k] = I_PV[k]
145                     I_Damp[k] = 0
146                     I_Charge[k] = 0
147                     I_Deficit[k] = 0
148                     I_Discharge[k] = 0
149                     I_Battery[k] = 0
150                     if k == 0:
151                         SOC[k] = SOC1
152                     elif w == 0:
153                         SOC[k] = SOC1
154                     elif w == 1:
155                         SOC[k] = SOC[k-1]
156
157                 elif I_net[k] > 0:
158                     I_Load[k] = I_L[k]
159                     if k == 0:
160                         SOC[k] = SOC1
161                     I_Damp[k] = I_net[k]

```

```

163         I_Charge[k] = 0
164         I_Deficit[k] = 0
165         I_Discharge[k] = 0
166         I_Battery[k] = 0
167     elif k > 0:
168         if w == 0:
169             SOC[k] = SOC1
170             I_Damp[k] = I_net[k]
171         elif w == 1:
172             if SOC[k-1] >= SOC1:
173                 SOC[k] = SOC[k-1]
174                 I_Damp[k] = I_net[k]
175                 I_Charge[k] = 0
176                 I_Deficit[k] = 0
177                 I_Discharge[k] = 0
178                 I_Battery[k] = 0
179             elif SOC[k-1] < SOC1:
180                 I_Damp[k] = 0
181                 I_Charge[k] = I_net[k]
182                 I_Deficit[k] = 0
183                 I_Discharge[k] = 0
184                 I_Battery[k] = 0
185
186                 B = SOC2
187                 V1 = (2 + (0.148 * B)) * ns
188                 R1 = float(((0.758 + (0.1309 / (1.06 - B)))
189                             * ns) / SOCmax)
189                 v = sp.Symbol('v')
190                 ee = float(sp.integrate(((K * V1 * I_net[k]
191                                         ] - (D * SOC2 * SOCmax))), (v, 0, t)))
191                 SOCx = SOC1 + (SOCmax**(-1) * ee
192                 SOC2 = SOCx
193                 SOC2 = float(SOCx)
194                 SOC[k] = SOC[k-1] + abs(SOC1 - SOC2)
195
196
197         I_Damp[k] = 0
198         I_Charge[k] = 0
199         I_Deficit[k] = 0
200         I_Discharge[k] = I_L[k] - I_PV[k]
201         I_Battery[k] = I_Battery[k]
202         I_Load[k] = I_L[k]
203         B = SOC2
204         V1 = (1.926 + 0.124 * B) * ns
205         R1 = (0.19 + (0.1037 / (B - 0.14))) * (ns /
206             SOCmax)
207         v = sp.Symbol('v')
208         ee = float(sp.integrate(((K * V1 * I_net[k]) -
209                                 (D * SOC2 * SOCmax))), (v, 0, t)))
210         SOCx = SOC1 + (SOCmax**(-1) * ee
211         SOC2 = SOCx
212         SOC2 = float(SOCx)
213         SOC[k] = SOC[k-1] - abs(SOC1 - SOC2)
214     elif SOC[k-1] <= SOC3:
215         SOC2 = SOC3
216         SOC[k] = SOC3
217         I_Damp[k] = 0
218         I_Charge[k] = 0

```

```

217         I_Deficit[k] = I_net[k]
218         I_Discharge[k] = 0
219         I_Battery[k] = 0
220         I_Load[k] = 0
221
222
223     else:
224
225         if I_net[k] == 0:
226             SOC[k] = SOC1
227             I_Load[k] = I_PV[k]
228             I_Damp[k] = 0
229             I_Charge[k] = 0
230             I_Deficit[k] = 0
231             I_Discharge[k] = 0
232             I_Battery[k] = 0
233
234         elif I_net[k] > 0:
235             SOC[k] = SOC1
236             I_Load[k] = I_L[k]
237             I_Damp[k] = I_net[k]
238             I_Charge[k] = 0
239             I_Deficit[k] = 0
240             I_Discharge[k] = 0
241             I_Battery[k] = 0
242
243         LLP_ff.append(LLP_calculated[ii, jj])
244         C_PV_ff.append(C_PVf[ii, jj])
245         C_battery_ff.append(C_batteryf[ii, jj])
246         cc = cc + 1
247
248 if len(C_PV_ff) == 0:
249     print('No combinations found with LLP between 0.0095 and 0.0105')
250     print(f'Minimum LLP found: {np.min(LLP_calculated)}')
251     print(f'Maximum LLP found: {np.max(LLP_calculated)}')
252
253
254     idx = np.argmin(np.abs(LLP_calculated - 0.01))
255     row, col = np.unravel_index(idx, LLP_calculated.shape)
256
257     LLP_ff = [LLP_calculated[row, col]]
258     C_PV_ff = [C_PVf[row, col]]
259     C_battery_ff = [C_batteryf[row, col]]
260
261     print(f'Using closest LLP value: {LLP_ff[0]}')
262
263
264 LLP_ff = np.array(LLP_ff)
265 C_PV_ff = np.array(C_PV_ff)
266 C_battery_ff = np.array(C_battery_ff)
267
268
269
270 CC_PV = 456
271 MC_PV = 6.5
272 Ls = 25
273 L_PV = 25
274

```

```

275 # Battery
276 Ca_battery = 1200
277 CC_batwh = 4.8
278 CC_bat = CC_batwh * Ca_battery
279 MC_bat = 3.4
280 L_bat = 5
281 Y_bat = (Ls / L_bat) - 1
282 B_rep = 50
283 CC_B_rep = Y_bat * B_rep
284
285 # Charge Controller
286 CC_cc = 400
287 MC_cc = 0
288 L_cc = 25
289 Y_cc = (Ls / L_cc) - 1
290 N_cc = 1
291
292 # Inverter
293 CC_inv = 800
294 MC_inv = 0
295 L_inv = 25
296 Y_inv = (Ls / L_inv) - 1
297 N_inv = 1
298
299
300 N_CB = 4
301 C_CB = 25
302 CC_CB = N_CB * C_CB
303
304 CC_SS = 200
305
306 CC_CW = 400
307
308 CC_OC = CC_CB + CC_SS + CC_CW + CC_B_rep
309 ir = 0.035
310 fr = 0.015
311 ndr = ((1 + ir) / (1 + fr)) - 1
312
313 # Total life cycle cost calculation
314 LCCx = np.zeros(len(C_PV_ff))
315 CC_D = np.zeros(len(C_PV_ff))
316 LCC = np.zeros(len(C_PV_ff))
317
318 for kk in range(len(C_PV_ff)):
319     LCCx[kk] = (CC_OC / (((1 + ndr)**Ls) - 1) / ((ndr * ((1 + ndr)**Ls))))
320     ) + ((C_PV_ff[kk] * (CC_PV + Ls * MC_PV)) / L_PV) + ((np.ceil(
321         C_battery_ff[kk] / Ca_battery) * CC_bat * (1 + Y_bat)) + (MC_bat * (
322         Ls - Y_bat))) / L_bat) + (((N_cc * CC_cc * (1 + Y_cc)) + (MC_cc * (
323         Ls - Y_cc))) / L_cc) + (((N_inv * CC_inv * (1 + Y_inv)) + (MC_inv *
324         (Ls - Y_inv))) / L_inv)
325     CC_D[kk] = (C_PV_ff[kk] * (CC_PV)) + ((np.ceil(C_battery_ff[kk] /
326         Ca_battery) * CC_bat * (1 + Y_bat))) + (N_cc * CC_cc * (1 + Y_cc)) +
327         (N_inv * CC_inv * (1 + Y_inv)) + CC_CB + CC_SS
328     LCC[kk] = LCCx[kk] - (((0.13 * (CC_D[kk])))) / (((1 + ndr)**Ls) - 1) /
329         ((ndr * ((1 + ndr)**Ls))))
330
331 MM = np.min(LCC)
332 II = np.argmin(LCC)

```

```

325 C_PV_best = C_PV_ff[II]
326 C_battery_best = C_battery_ff[II]
327
328 print(f'Best PV Modules Number is: {int(C_PV_best)}')
329 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:

```

1  # Exercise 6.2
2
3  import numpy as np
4  import pandas as pd
5  import matplotlib.pyplot as plt
6  from mpl_toolkits.mplot3d import Axes3D
7
8  # Load Excel file
9  fileName = 'PV Modeling Book Data Source.xls'
10 sheetName = 'Source 7'
11 llp = 0.01
12 LOAD = 1
13
14 # (1) Assumptions
15 PV_eff = 0.16
16 Inv_eff = 0.90
17 V_B = 12
18 DOD = 0.8
19 Charge_eff = 0.8
20 Wp_Cost = 3.7
21 Ah_Cost = 2.5
22 kWp_Cost = 1200
23
24 # (3) Energy sources size
25 SE = pd.read_excel(fileName, sheet_name=sheetName, usecols='K', skiprows=1,
26                      nrows=365).values.flatten()
27 WS = pd.read_excel(fileName, sheet_name=sheetName, usecols='J', skiprows=1,
28                      nrows=365).values.flatten()
29
30 C_A = []
31 C_W = []
32 C_D = []
33 LLP = []
34 C_B = []
35
36 for PV_A in np.arange(0, 1.05, 0.05):
37     PV_E = 0.17 * SE * PV_A * 0.9
38     C_Ai = (PV_A * 0.17 * 4.9 * 0.9) / LOAD
39
40     for R in np.arange(0, 1.05, 0.05):
41         Air_Density = 1.22521

```

```

40     W_Ei = (Air_Density * 3.14 * (R ** 2) * 0.5 * 24 * (WS ** 3)) /
        1000
41     W_E = W_Ei * 0.2
42     C_Wi = np.sum(W_E) / (LOAD * 365)
43
44     for E_diesel in np.arange(0, 1.05, 0.05):
45         C_Di = E_diesel / LOAD
46         C_D.append(C_Di)
47         C_A.append(C_Ai)
48         C_W.append(C_Wi)
49
50         E_T = PV_E + W_E + E_diesel
51         E_NET = E_T - LOAD
52
53         EN = E_NET[E_NET < 0]
54         EP = E_NET[E_NET >= 0]
55
56         C_Bi = (np.sum(EP) * 0.8 / 365) / LOAD
57         C_B.append(C_Bi)
58
59         LLPi = (-1 * np.sum(EN)) / (LOAD * 365)
60         LLP.append(LLPi)
61
62     C_A = np.array(C_A)
63     C_W = np.array(C_W)
64     C_D = np.array(C_D)
65     C_B = np.array(C_B)
66     LLP = np.array(LLP)
67
68     Array = np.column_stack((C_A, C_W, C_D, C_B, LLP))
69
70     llp_modified1 = llp + 0.0003
71     llp_modified2 = llp - 0.0003
72
73     Array_New = []
74
75     for j in range(len(Array)):
76         if llp_modified2 <= Array[j, 4] <= llp_modified1:
77             Array_New.append(Array[j])
78
79     Array_New = np.array(Array_New)
80
81     if Array_New.shape[0] > 0:
82         T = np.arange(0, len(Array_New))
83         x_1 = Array_New[T, 0]
84         x_2 = Array_New[T, 1]
85         x_3 = Array_New[T, 2]
86         x_4 = Array_New[T, 3]
87         x_5 = x_1 + x_2 + x_3
88
89         fig = plt.figure()
90         ax = fig.add_subplot(111, projection='3d')
91         ax.plot3D(x_1, x_2, x_3, 'b.')
92         ax.set_xlabel('PV')
93         ax.set_ylabel('Wind')
94         ax.set_zlabel('Diesel')
95         plt.title('PV-Wind-Diesel Combination')
96         plt.show()

```



```

97 else:
98     print("No configurations found within LLP range.")

```

6.4 PV PUMPING SYSTEM SIZE OPTIMIZATION

Answer:

```

1  # Exercise 6.3
2
3  import numpy as np
4  import pandas as pd
5  from numpy import pi, tan, abs, sqrt, real, imag
6  import warnings
7
8  warnings.filterwarnings("ignore")
9
10 fileName = 'PV Modeling Book Data Source.xls'
11 sheetName = 'Source 7'
12
13
14 G = pd.read_excel(fileName, sheet_name=7, usecols='I', skiprows=9, nrows
    =4380).values.flatten()
15 Tc = pd.read_excel(fileName, sheet_name=7, usecols='J', skiprows=9, nrows
    =4380).values.flatten()
16 Vm = pd.read_excel(fileName, sheet_name=7, usecols='L', skiprows=9, nrows
    =4380).values.flatten()
17 Im = pd.read_excel(fileName, sheet_name=7, usecols='M', skiprows=9, nrows
    =4380).values.flatten()
18
19 CNs = np.zeros(80000)
20 CNp = np.zeros(80000)
21 CCn = np.zeros(80000)
22 CLLP = np.ones(80000) * -0.5
23 CQexcess = np.ones(80000) * -0.5
24 CQdeficit = np.ones(80000) * -0.5
25 CQ = np.ones(80000) * -0.5
26
27 q = 0
28
29 for Cn in range(1, 3):
30     for N in range(6, 8):
31         for Ns in range(1, N + 1):
32             if N % Ns == 0:
33                 Np = N // Ns
34                 h1 = 20
35                 h2 = 0.1444
36
37                 Va = Ns * Vm
38                 Ia = Np * Im
39                 Vpv = Va
40                 Ipv = Ia
41                 Pao = Va * Ia
42                 Am = 0.9291
43                 A = Ns * Np * Am
44                 Pai = A * G
45                 with np.errstate(divide='ignore', invalid='ignore'):

```

```

46         effa = np.nan_to_num(Pao / Pai)
47
48     Va = 0.95 * Va
49     Ia = 0.90 * Ia
50     Ra = 0.8
51     Km = 0.175
52     Ebb = Va - Ra * Ia
53     Eb = (Ebb >= 0) * Ebb
54     Ia = (Ebb >= 0) * Ia
55     Va = (Ebb >= 0) * Va
56     Tm = Km * Ia
57     Tmm = (Tm == 0) * 1
58     Tml = Tmm + Tm
59
60     Rou = 1000
61     d1 = 33.5 * 0.001
62     d2 = 160 * 0.001
63     beta1 = 38 * 2 * pi / 360
64     beta2 = 33 * 2 * pi / 360
65     b1 = 5.4 * 0.001
66     b2 = 2.2 * 0.001
67     Kp = Rou * 2 * pi * b1 * (d1 / 2) ** 2 * tan(beta1) * ((d2
        / 2) ** 2 - ((b1 * (d1 / 2) ** 2 * tan(beta1)) / (b2 *
        tan(beta2))))
68
69     Pdev = Eb * Ia
70     with np.errstate(divide='ignore', invalid='ignore'):
71         Omega = np.nan_to_num(sqrt((Km * Ia) / Kp))
72     Pmo = Pdev
73     Pmi = Pao * 0.9
74     PMI1 = (Pmi == 0) * 1
75     PMI2 = Pmi + PMI1
76     with np.errstate(divide='ignore', invalid='ignore'):
77         effm = np.nan_to_num(Pmo / PMI2)
78
79     Tp = Tm
80     Eh = Tp * Omega
81     Ppo = Eh
82     Ppo = (Ebb >= 0) * Ppo
83     Ppi = Pmo
84     PPI1 = (Ppi == 0) * 1
85     PPI2 = Ppi + PPI1
86     with np.errstate(divide='ignore', invalid='ignore'):
87         effpp = np.nan_to_num(Ppo / PPI2)
88         effp = (effpp <= 0.95) * effpp
89
90     QQ = np.zeros(len(Eh))
91     for ii in range(len(Eh)):
92         coeffs = [h2 * 2.725, 0, h1 * 2.725, -Eh[ii]]
93         r = np.roots(coeffs)
94         QQQ = 0
95         for root in r:
96             if imag(root) == 0 and real(root) > 0:
97                 QQQ = real(root)
98             break
99         QQ[ii] = QQQ
100
101     Q1 = (QQ == 0) * 1

```

```

102     Q2 = QQ + Q1
103     with np.errstate(divide='ignore', invalid='ignore'):
104         H = np.nan_to_num(Eh / (2.725 * Q2))
105
106     effsub = effm * effp
107     effoverall = effa * effm * effp
108     QQ = (Ebb >= 0) * QQ
109     QQ = (effpp <= 0.95) * QQ
110     Q = np.concatenate(([0], QQ))
111     d = 2.5
112     Cr = np.zeros(len(Q))
113     Qexcess_pv = np.zeros(len(Q))
114     Qexcess_s = np.zeros(len(Q))
115     SOC = np.zeros(len(Q))
116     Qdef_pv = np.zeros(len(Q))
117     Qdeficit_s = np.zeros(len(Q))
118     X = Q[1:] - d
119
120     Qdef_pv[1:] = (X < 0) * abs(X)
121     Qexcess_pv[1:] = (X >= 0) * abs(X)
122
123     C = len(Q) - 1
124     for i in range(C):
125         Cr[i+1] = ((Cr[i] + X[i]) >= 0) * abs(Cr[i] + X[i])
126         SOC[i+1] = Cr[i+1] / Cn
127         if SOC[i+1] >= 1:
128             SOC[i+1] = 1
129             Qexcess_s[i+1] = Cr[i+1] - Cn
130             Cr[i+1] = Cn
131         else:
132             Qexcess_s[i+1] = 0
133             Qdeficit_s[i+1] = ((Cr[i] + X[i]) < 0) * abs(Cr[i] + X[i])
134
135     Q = Q[1:]
136     sumQ = np.sum(Q)
137     Qexcess = np.sum(Qexcess_s[1:])
138     Qdeficit = np.sum(Qdeficit_s[1:])
139     D = np.full(len(Q), d)
140     with np.errstate(divide='ignore', invalid='ignore'):
141         LLP = np.sum(Qdeficit_s[1:]) / np.sum(D)
142
143     if LLP <= 0.01:
144         CNs[q] = Ns
145         CNp[q] = Np
146         CCn[q] = Cn
147         CLLP[q] = LLP
148         CQexcess[q] = Qexcess
149         CQdeficit[q] = Qdeficit
150         CQ[q] = sumQ
151         q += 1
152
153 CNs_result = CNs[:q]
154 CNp_result = CNp[:q]
155 CCn_result = CCn[:q]
156 CLLP_result = CLLP[:q]
157 CQ_result = CQ[:q]
158

```

159 | CNS_result, CNp_result, CCn_result, CLLP_result, CQ_result

Reference Book:

MODELING OF PHOTOVOLTAIC
SYSTEMS USING MATLAB®

Tamer Khatib, Wilfried Elmenreich