

USANDO A BIBLIOTECA

PVLIB

PARA A OBTENÇÃO DE CURVAS DE
GERAÇÃO FV REALÍSTICAS



```
1 import pvlib
2 from pvlib.modelchain import ModelChain
3 from pvlib.location import Location
4 from pvlib.pvsystem import PVSystem
5 from pvlib.temperature import TEMPERATURE_MODEL_PARAMETERS
6
7 import pandas as pd
8 import matplotlib.pyplot as plt
9
10 location = Location(Latitude=-0.61956112905,
11                         Longitude=-56.06485209329807,
12                         tz='America/Fortaleza',
13                         altitude=190,
14                         Name='FAET_UFMT')
15
16 sandia_models = pvlib.pvsystem.retrieve_sam('SandiaMod')
17 cec_inverters = pvlib.pvsystem.retrieve_sam('CECInverter')
18
19 module = sandia_models['Canadian_Solar_CS3W_200M_200V_']
20 inverter = cec_inverters['ABB_MICRO_0_25_I_OUTD_US_208_208V_']
21
22 temperature_parameters = TEMPERATURE_MODEL_PARAMETERS['sapm']['open_rack_glass_glass']
23
24 system = PVSystem(surface_tilt=45, surface_azimuth=180,
25                     module_parameters=module,
26                     inverter_parameters=inverter,
27                     temperature_model_parameters=temperature_parameters,
28                     modules_per_string=1,
29                     strings_per_inverter=1)
30
31
32 modelChain = ModelChain(system,location)
33
34 times = pd.date_range(start="2019-01-01",end="2019-01-02",
35                         freq="1h",
36                         tz=location.tz)
```

PREFÁCIO

A crescente adoção de fontes de energia solar pode ser atribuída a uma série de fatores. Primeiramente, a conscientização sobre a importância da sustentabilidade e a necessidade de reduzir as emissões de gases de efeito estufa. Além disso, o avanço da tecnologia de energia solar, tornando-a mais acessível e eficiente, bem como subsídios governamentais e vantagens financeiras advindas da instalação de painéis fotovoltaicos, tem contribuído para sua popularidade. No entanto, devido à natureza intermitente da energia solar, é crucial realizar simulações computacionais para otimizar o dimensionamento e o planejamento desses sistemas. Essas simulações permitem avaliar a viabilidade de projetos, determinar a capacidade necessária, prever a produção de energia, considerar fatores geográficos e climáticos, e otimizar a colocação de painéis solares.



PREFÁCIO

A biblioteca PVLIB é uma ferramenta poderosa para auxiliar na modelagem desses arranjos fotovoltaicos. Ela fornece uma ampla gama de funcionalidades e recursos que permitem simular, analisar e otimizar o desempenho de sistemas solares fotovoltaicos. Combinada com a base de dados fornecida pelo PVGIS (Photovoltaic Geographical Information System), elaborado pelo *Science Hub* da Comissão Europeia, essa ferramenta é capaz de gerar excelentes estimativas de geração solar, ao longo de um período e em uma localização desejada.

Este material tem por objetivo apresentar os primeiros passos para aqueles que desejam utilizar essa biblioteca. De maneira a complementar o aprendizado, acessando exemplos e referência úteis para esse estudo, visite o repositório público criado para esse material, disponível através do QR Code ao lado.



```
elchain import ModelChain
ation import Location
ystem import PVSystem
perature import TEMPERATURE_MODEL_PARAMETERS

as pd
lib.pyplot as plt

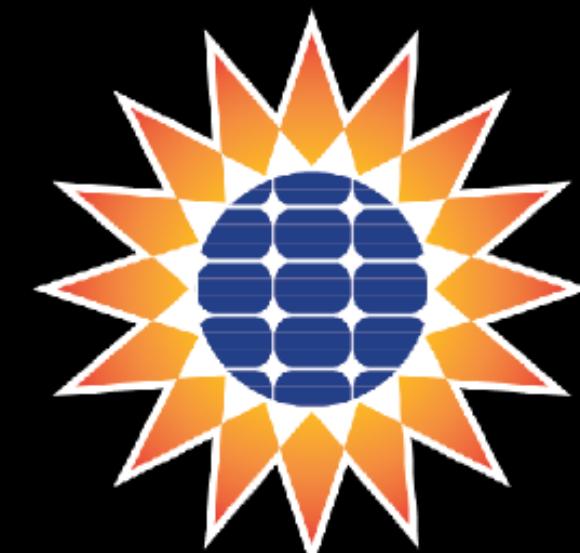
AET UFMT
ation(latitude=-15.608259564142905,
longitude=-56.06485209329807,
tz='America/Fortaleza',
altitude=190,
name='FAET UFMT')

= pvli INTRODUÇÃO AO
= pvli PVLIB
a_mode
_inver
arameters = TEMPERATURE_MODEL_PARAMETERS['sapm']['op
cem(surface_tilt=45,surface_azimuth=180,
module_parameters = module,
inverter_parameters = inverter,
temperature_model_parameters=temperature_params,
modules_per_string=1,
strings_per_inverter=1)

odelChain(system,location)
e_range(start="2019-01-01",end="2019-01-02",
freq="1h",
```

O QUE É O PVLIB?

PVLIB é uma biblioteca *open-source* para python que fornece um conjunto de funções e classes para simular o desempenho de arranjos fotovoltaicos.



Inicialmente, essa biblioteca fora desenvolvida para o MATLAB pelo Sandia National Laboratories, laboratório de pesquisa e desenvolvimento vinculado ao governo dos Estados Unidos, e implementa muitos dos modelos e métodos desenvolvidos no laboratório.

A biblioteca possui ferramentas poderosas, tais como a geração de curvas de irradiância baseada em coordenadas geográficas, modelos de atmosféricos e de massa de ar, posição solar, além de avaliação de rendimento de arranjos fotovoltaicos com ou sem sistema de *tracking*, permitindo que a simulação seja realizada com modelos realísticos de equipamentos disponíveis no mercado, detalhadamente modelados nos bancos de dados disponíveis.

Quer saber como começar a utilizar essa biblioteca? Siga os 3 passos listados a seguir.

PRIMEIRO PASSO: INSTALANDO A BIBLIOTECA PVLIB

O primeiro passo é verificar se você possui o Python instalado em seu computador. Isso pode ser feito digitando o código abaixo no terminal ou prompt de comando de seu computador.

```
Longitude=-56.06485209329807,
```



```
1 python --version
```

Caso já possua o Python instalado em seu computador, esse comando retornará a versão instalada. Caso não possuir retornará um erro.

Nesse caso, ou caso deseje atualizar para uma versão mais recente, baixe-o de acordo com seu sistema operacional no site oficial

```
modules_per_inverter=1,
```

```
inverter_parameters = inverter,
```

```
temperature_model_parameters=temperature_parameters,
```

```
modules_per_inverter=1,
```

```
strings_per_inverter=1)
```

```
modelChain(system,location)
```

```
date_range(start="2019-01-01",end="2019-01-02",
```

```
freq="1h",
```

PRIMEIRO PASSO: INSTALANDO A BIBLIOTECA PVLIB

Após isso, verifique se seu computador possui o pip instalado. O pip é o gerenciador de pacotes padrão do Python. Você pode verificar se o possui instalado em sua máquina digitando o comando abaixo no prompt de comando.

```
● ● ●  
1 pip --version
```

Em geral, o pip é instalado juntamente com o Python, automaticamente. Caso não possua em seu computador, poderá baixá-lo no site oficial do Python, seguindo as instruções fornecidas.

O pip será responsável por baixar e instalar a biblioteca pvlib e todas as suas dependências.

Para baixar a biblioteca execute o seguinte comando no terminal:

```
● ● ●  
1 pip install pvlib
```

PRIMEIRO PASSO: INSTALANDO A BIBLIOTECA PVLIB

Feito isso, se tudo correu bem, a biblioteca pvlib já está instalada em seu computador. Você poderá testá-la digitando o seguinte comando na IDE que irá realizar o desenvolvimento (Visual Studio Code, PyCharm, ou outra de sua preferência):

```
● ● ●  
1 import pvlib
```

Se não houver mensagens de erro a importação da biblioteca foi bem-sucedida e você está pronto para começar a usar a biblioteca pvlib.

A documentação completa da biblioteca pode ser encontrada no site oficial do pvlib:

<https://pvlib-python.readthedocs.io>

SEGUNDO PASSO: CONHECENDO A BIBLIOTECA

A biblioteca se utiliza de uma classe chamada **ModelChain**, que fornece uma interface de alto nível para padronizar a modelagem de sistemas fotovoltaicos (FV). Essa classe automatiza muitos processos de modelagem, provendo flexibilidade na implementação. Uma ModelChain possuirá três componentes:

- Um objeto do tipo **PVSystem**, que representa uma coleção de módulos e inversores;
- Um objeto **Location**, representando a localização geográfica;
- Valores de atributos que especificam o modelo de referência utilizado no processo de modelagem FV.

Para utilizá-la, importe-a da biblioteca pvlib em seu algoritmo por meio do código abaixo:

```
1 import pvlib
2 from pvlib.modelchain import ModelChain
```

SEGUNDO PASSO: CONHECENDO A BIBLIOTECA

Antes de criarmos um objeto da classe ModelChain precisamos definir ao menos dois atributos que o formarão: um objeto do tipo *system* e *location*, conforme exposto anteriormente. Assim, importa-se Location da biblioteca e define-se um objeto cujos atribuidos são as coordenadas da Faculdade de Engenharia e Arquitetura da UFMT, para uma altitude do arranjo FV de 190 metros, considerando o fuso horário brasileiro. Você pode ainda adicionar um nome para o objeto, através do atributo *name*.

```
● ● ●
1 import pvlib
2 from pvlib.modelchain import ModelChain
3 from pvlib.location import Location
4
5 #Location of FAET UFMT
6 location = Location(latitude=-15.608259564142905,
7                      longitude=-56.06485209329807,
8                      tz='America/Fortaleza',
9                      altitude=190,
10                     name='FAET UFMT')
```

Para saber os fusos disponíveis acesse a página da Wikipedia disponível no link abaixo e altere o argumento *tz* no objeto *location* para o valor presente na coluna "*TZ identifier*" da página.

https://en.wikipedia.org/wiki/List_of_tz_database_time_zones

SEGUNDO PASSO: CONHECENDO A BIBLIOTECA

Agora, partindo para a criação do objeto *system*, primeiramente importaremos *PVSystem* da biblioteca *pvlib*. Em seguida, definiremos um objeto *PVSystem* que receberá os atributos *surface_tilt* e *surface_azimuth*, ambos relativos a inclinação dos painéis. Também adicionamos os parâmetros do inversor e módulo utilizados, bem como o modelo de temperatura de célula que será adotado, definidos em detalhes a seguir. Por fim, definimos que o arranjo possuirá um módulo por *string*, com uma *string* por inversor

```
● ● ●  
from pvlib.pvsystem import PVSystem  
  
system = PVSystem(surface_tilt=45,surface_azimuth=180,  
                   module_parameters = module,  
                   inverter_parameters = inverter,  
                   temperature_model_parameters=temperature_parameters,  
                   modules_per_string=1,  
                   strings_per_inverter=1)
```

```
temperature_model_parameters=temperature_parameters,  
modules_per_string=1,  
strings_per_inverter=1)
```

```
modelChain(system,location)
```

```
e_range(start="2019-01-01",end="2019-01-02",  
        freq="1h",
```

SEGUNDO PASSO: CONHECENDO A BIBLIOTECA

A biblioteca pvlib utiliza de bases de dados de diferentes institutos para a modelagem dos módulos FV e inversores. No exemplo abaixo utiliza-se da base de dados *Sandia Modules* para o modelo do módulo FV e *CEC* para o modelo do inversor. Selecionam-se os equipamentos dos fabricantes *Canadian Solar* e *ABB*.

```
● ● ●  
  
sandia_models = pvlib.pvsystem.retrieve_sam('SandiaMod')  
cec_inverters = pvlib.pvsystem.retrieve_sam('CECInverter')  
  
module = sandia_models['Canadian_Solar_CS5P_220M__2009_']  
inverter = cec_inverters['ABB_MICRO_0_25_I_OUTD_US_208_208V_']
```

As bases de dados disponíveis são:

- **Módulos:**
 1. 'CECMod' - base de dados CEC
 2. 'SandiaMod' - base de dados Sandia
- **Inversores:**
 1. 'CECInverter' - base de dados CEC
 2. 'SandiaInverter' - base de dados Sandia
 3. 'ADRInverter' - base de dados ADR

Todos os modelos contemplados e atualizados podem ser encontrados no repositório abaixo:

[https://github.com/NREL/SAM/tree/
develop/deploy/libraries](https://github.com/NREL/SAM/tree/develop/deploy/libraries)

SEGUNDO PASSO: CONHECENDO A BIBLIOTECA

O modelo de temperatura irá ditar o comportamento dos módulos fotovoltaicos, em termos de desempenho, quando aquecidos. Para definir o modelo, novamente, importamos o módulo que os referencia. Nesse exemplo utilizou-se o modelo Sandia Array Performance (SAMP) 'open_rack_glass_glass'.

```
name='FAET UFMT')
```



```
from pvlib.temperature import TEMPERATURE_MODEL_PARAMETERS  
temperature_parameters = TEMPERATURE_MODEL_PARAMETERS['sapm']['open_rack_glass_glass']
```

Os modelos de temperatura de costa de células são:

- sapm

1. 'open_rack_glass_glass':

2. 'close_mount_glass_glass'

3. 'open_rack_glass_polymer'

4. 'insulated_back_glass_polymer'

- pvsyst

1. 'freestanding'

2. 'insulated'

SEGUNDO PASSO: CONHECENDO A BIBLIOTECA

Finalmente, poderemos criar nosso objeto de classe ModelChain que será formado com base nos modelos do sistema e localização definidos. O código completo deverá ficar da seguinte forma:

```
tz='America/Fortaleza',
```

```
● ● ●  
1 import pvlib  
2 from pvlib.modelchain import ModelChain  
3 from pvlib.location import Location  
4 from pvlib.pvsystem import PVSystem  
5 from pvlib.temperature import TEMPERATURE_MODEL_PARAMETERS  
6  
7 sandia_models = pvlib.pvsystem.retrieve_sam('SandiaMod')  
8 cec_inverters = pvlib.pvsystem.retrieve_sam('CECInverter')  
9  
10 module = sandia_models['Canadian_Solar_CS5P_220M__2009__']  
11 inverter = cec_inverters['ABB__MICRO_0_25_I_OUTD_US_208__208V__']  
12  
13 temperature_parameters = TEMPERATURE_MODEL_PARAMETERS['sapm']['open_rack_glass_glass']  
14  
15 system = PVSystem(surface_tilt=45,surface_azimuth=180,  
16     module_parameters = module,  
17     inverter_parameters = inverter,  
18     temperature_model_parameters=temperature_parameters,  
19     modules_per_string=1,  
20     strings_per_inverter=1)  
21  
22 modelChain = ModelChain(system,location)
```

Com o modelo especificado, resta agora definirmos em que condições de irradiância esse arranjo estará exposto, isto é, em qual período de tempo, estação do ano e horários deve-se realizar a simulação.

TERCEIRO PASSO: SUA PRIMEIRA SIMULAÇÃO

Para realizarmos a simulação primeiramente devemos determinar o período de tempo analisado. Para isso, utilizaremos a biblioteca *Pandas* que fornece uma série de ferramentas para análise e manipulação de dados. Após importarmos a biblioteca e definimos uma variável *times* que será um *data_range*, cujos argumentos definem o período em que se deseja realizar a simulação, bem como a granularidade de dados desejada. Nesse caso, deseja-se uma simulação horária.

```
import pandas as pd

times = pd.date_range(start="2019-01-01", end="2019-01-02",
                      freq="1h",
                      tz=location.tz)
```

```
module_parameters = module,
inverter_parameters = inverter,
temperature_model_parameters=temperature_parameters,
modules_per_string=1,
strings_per_inverter=1)
```

O fuso horário (*tz*) é definido como sendo o mesmo do objeto *location* criado anteriormente.

TERCEIRO PASSO: SUA PRIMEIRA SIMULAÇÃO

Realizaremos uma simulação em dia de céu limpo.

Para isso, criaremos uma variável *clear_sky* que receberá as informações de irradiação, para um dia de céu limpo, na *location* especificada, dentro do período *times* definido.

```
    ,  
    altitude=190,  
    name='FAET UFMT')
```



```
times = pd.date_range(start="2019-01-01", end="2019-01-02",  
                      freq="1h",  
                      tz=location.tz)  
  
clear_sky = location.get_clearsky(times)  
  
modelChain.run_model(clear_sky)
```

A função *run_model* executa o modelChain criado dentro do cenário *clear_sky* estipulado.

Com isso, a simulação foi executada e os resultados podem ser plotados

```
modules_per_string=1,  
strings_per_inverter=1)
```

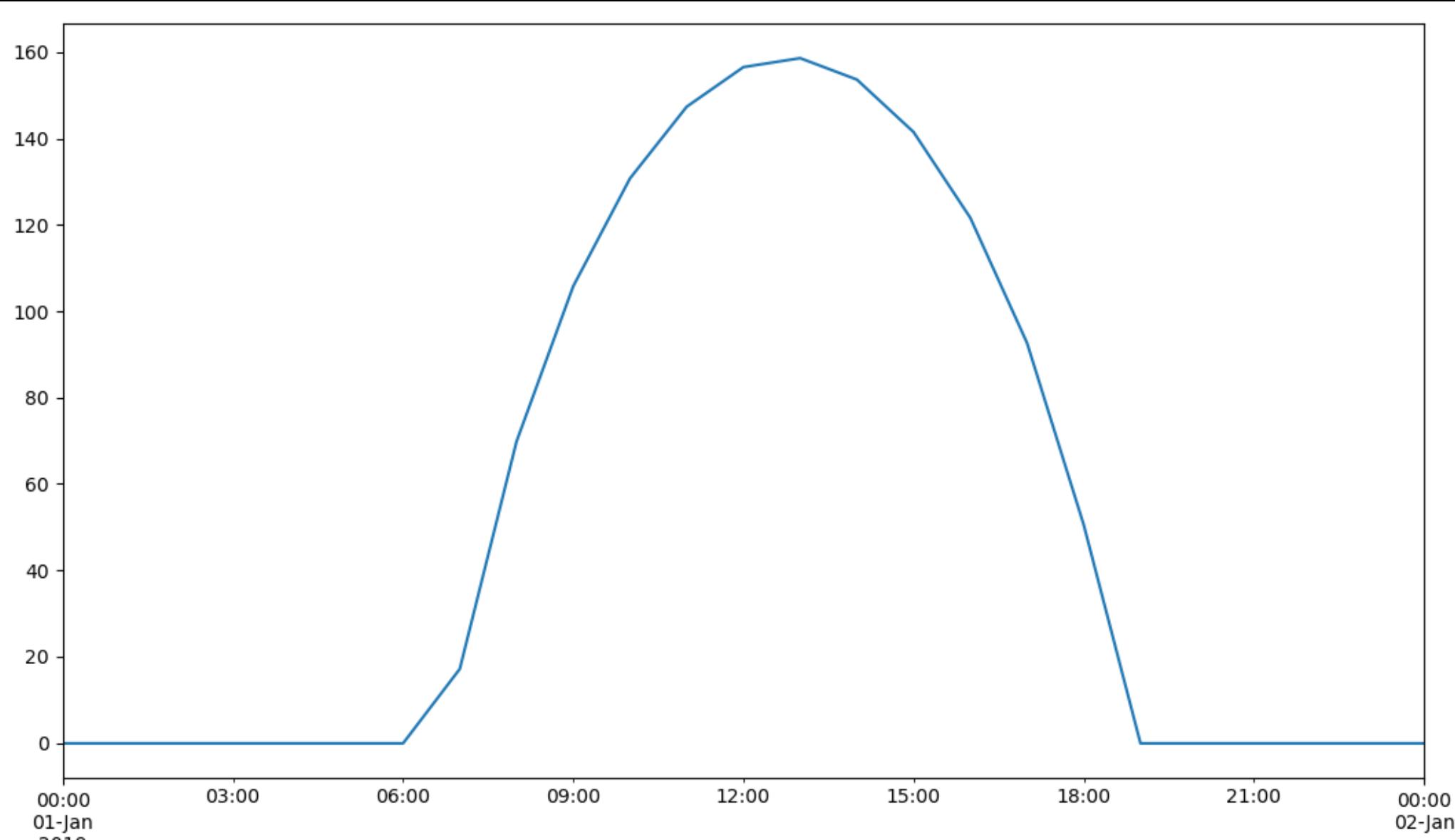
TERCEIRO PASSO: SUA PRIMEIRA SIMULAÇÃO

Para a plotagem dos resultados utiliza-se a biblioteca *matplotlib*. O código abaixo é usado para a plotagem do gráfico de potência CA gerada pelo arranjo.



```
import matplotlib.pyplot as plt  
  
modelChain.results.ac.plot(figsize=(16,9))  
plt.show()
```

A curva gerada é apresentada abaixo:



TERCEIRO PASSO: SUA PRIMEIRA SIMULAÇÃO

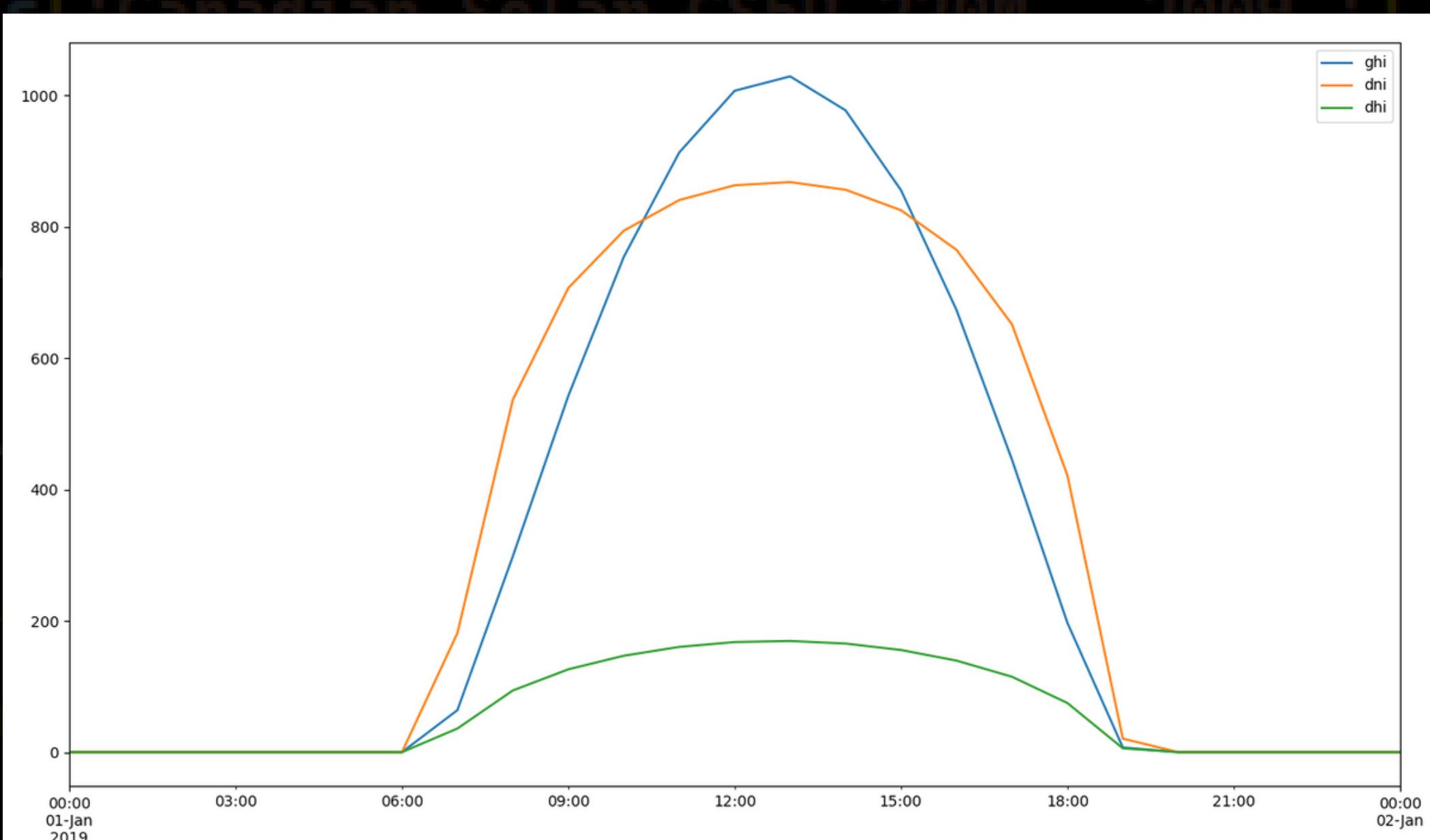
Pode-se ainda plotar os gráficos de irradiância gerados para o cenário elaborado, de maneira simples, usando o código abaixo.



```
clear_sky.plot(figsize=(16,9))  
plt.show()
```

```
= pvlib  
= pvlib.pvsystem.retrieve_sam('CECInverter')
```

Obtém-se o seguinte resultado



TERCEIRO PASSO: SUA PRIMEIRA SIMULAÇÃO

As irradâncias apresentadas são:

- **GHI**: *Global Horizontal Irradiance*, sendo essa a irradânciā total vinda do sol, em uma superfície horizontal na terra, sendo determinada por:

$$GHI = DHI + DNI \cdot \cos(Z)$$

Onde Z é o ângulo Zenital do arranjo e:

- **DNI**: *Direct Normal Irradiance*, é medida na superfície da terra para uma dada localização com a superfície do elemento perpendicular ao sol.
- **DHI**: *Diffuse Horizontal Irradiance*, é a radiação na superfície da terra pela luz espalhada pela atmosfera.

TERCEIRO PASSO: SUA PRIMEIRA SIMULAÇÃO

Observa-se que os resultados estão condizentes com o esperado. A geração se inicia por volta das 6 horas da manhã, tendo seu pico em torno do meio dia, e se encerra em torno das 19h. A potência máxima de pico gerada mostrada segue a capacidade dos inversores, bem como dos módulos FV e a quantidade desses dispostos no modelo

Na seção a seguir são apresentados alguns estudos de caso onde avalia-se os resultados gerados pela biblioteca para distintos cenários, bem como a plotagem de outras variáveis de interesse.

Apesar de condizentes, os resultados [de fato] são gerados considerando condições ideais de geração, em dia de céu limpo. Após a seção de exemplos, descubra como vincular dados meteorológicos ao pvlib para a obtenção de curvas de geração realísticas.

TERCEIRO PASSO: SUA PRIMEIRA SIMULAÇÃO

Confira o código completo utilizado para o primeiro exemplo.

```
● ● ●  
1 import pvlib  
2 from pvlib.modelchain import ModelChain  
3 from pvlib.location import Location  
4 from pvlib.pvsystem import PVSystem  
5 from pvlib.temperature import TEMPERATURE_MODEL_PARAMETERS  
6 import pandas as pd  
7 import matplotlib.pyplot as plt  
8  
9 #Location of FAET UFMT  
10 location = Location(latitude=-15.608259564142905,  
11                      longitude=-56.06485209329807,  
12                      tz='America/Fortaleza',  
13                      altitude=190,  
14                      name='FAET UFMT')  
15  
16 sandia_models = pvlib.pvsystem.retrieve_sam('SandiaMod')  
17 cec_inverters = pvlib.pvsystem.retrieve_sam('CECInverter')  
18  
19 module = sandia_models['Canadian_Solar_CS5P_220M__2009_']  
20 inverter = cec_inverters['ABB_MICRO_0_25_I_OUTD_US_208_208V_']  
21  
22 temperature_parameters = TEMPERATURE_MODEL_PARAMETERS['sapm']['open_rack_glass_glass']  
23  
24 system = PVSystem(surface_tilt=45,surface_azimuth=180,  
25                     module_parameters = module,  
26                     inverter_parameters = inverter,  
27                     temperature_model_parameters=temperature_parameters,  
28                     modules_per_string=1,  
29                     strings_per_inverter=1)  
30  
31 modelChain = ModelChain(system,location)  
32  
33  
34  
35 times = pd.date_range(start="2019-01-01",end="2019-01-02",  
36                      freq="1h",  
37                      tz=location.tz)  
38  
39 clear_sky = location.get_clearsky(times)  
40  
41 modelChain.run_model(clear_sky)  
42 modelChain.results.ac.plot(figsize=(16,9))  
43  
44 plt.show()
```

strings_per_inverter=1)

```
elchain import ModelChain
ation import Location
ystem import PVSystem
operatore import TEMPERATURE_MODEL_PARAMETERS

as pd
lib.pyplot as plt

AET UFMT
ation(latitude=-15.608259564142905,
longitude=-56.06485209329807,
tz='America/Fortaleza',
altitude=190,
name='FAET UFMT')

= pvli MÃOS NA MASSA:
= pvli EXEMPLOS
a_mode
_inver
parameters = TEMPERATURE_MODEL_PARAMETERS['sapm']['op
cem(surface_tilt=45,surface_azimuth=180,
module_parameters = module,
inverter_parameters = inverter,
temperature_model_parameters=temperature_params,
modules_per_string=1,
strings_per_inverter=1)

odelChain(system,location)
e_range(start="2019-01-01",end="2019-01-02",
freq="1h",
```

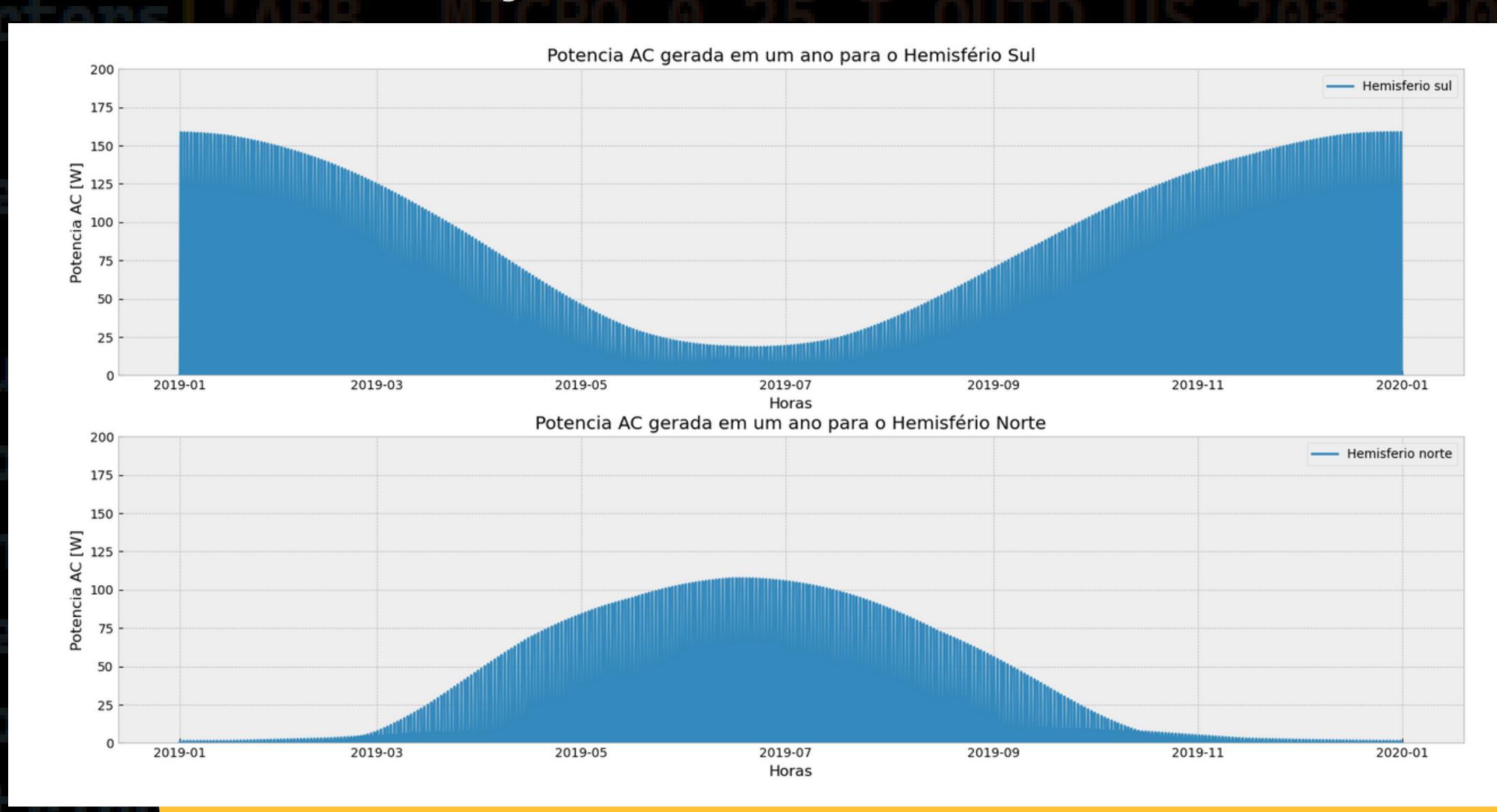
EXEMPLO 1: GERAÇÃO ANUAL NOS HEMISFÉRIOS NORTE E SUL

```
● ● ●
1 #Importando as bibliotecas e módulos necessários para a execução do código.
2 import pvlib
3 from pvlib.modelchain import ModelChain
4 from pvlib.location import Location
5 from pvlib.pvsystem import PVSystem
6 from pvlib.temperature import TEMPERATURE_MODEL_PARAMETERS
7
8 import pandas as pd
9 import matplotlib.pyplot as plt
10
11 #Criando um objeto do tipo location, para cada hemisfério, com as informações de coordenada geográfica, fuso horário, altitude e nome, que serão utilizadas pelo ModelChain.
12
13 #A localização no hemisfério sul se refere ao endereço da FAET/UFMT.
14 location_sul = Location(latitude=-15.608259564142905,
15                         longitude=-56.06485209329807,
16                         tz='America/Fortaleza',
17                         altitude=190,
18                         name='FAET UFMT')
19
20 #A localização se refere ao endereço da Universidade de New Castle, no Reino Unido.
21 location_norte = Location(latitude= 54.978,
22                            longitude= -1.615,
23                            tz='US/Eastern',
24                            altitude=190,
25                            name='Hemisferio norte')
26
27 #Considera-se os mesmos módulos e inversores para ambos os cenários de avaliação
28 #Escolhendo as bases de dados de módulos e inversores a serem utilizados.
29 sandia_models = pvlib.pvsystem.retrieve_sam('SandiaMod')
30 cec_inverters = pvlib.pvsystem.retrieve_sam('CECInverter')
31 #Selecionando o modelo do módulo e inverter a ser utilizado.
32 module = sandia_models['Canadian_Solar_CS5P_220M__2009_']
33 inverter = cec_inverters['ABB_MICRO_0_25_I_OUTD_US_208__208V_']
34
35 #Definindo o modelo de desempenho de acordo com a temperatura de costa de célula.
36 temperature_parameters = TEMPERATURE_MODEL_PARAMETERS['sapm']['open_rack_glass_glass']
37
38 #Definindo o sistema PV, com as informações de ângulo, ajuste de temperatura, modelo de módulo e de inversor, bem como definindo a quantidade de módulos utilizados por string e a quantidade de strings adotadas.
39 system_sul = PVSystem(surface_tilt=45,surface_azimuth=180,
40                       module_parameters = module,
41                       inverter_parameters = inverter,
42                       temperature_model_parameters=temperature_parameters,
43                       modules_per_string=1,
44                       strings_per_inverter=1)
45
46 system_norte = PVSystem(surface_tilt=45,surface_azimuth=30,
47                           module_parameters = module,
48                           inverter_parameters = inverter,
49                           temperature_model_parameters=temperature_parameters,
50                           modules_per_string=1,
51                           strings_per_inverter=1)
52
53 #Criação dos ModelChain, com as informações de localização e sistema PV.
54 modelChain_sul = ModelChain(system_sul,location_sul)
55 modelChain_norte = ModelChain(system_norte,location_norte)
56
57 #Criação de um Pandas Dataframe com os horários de início e fim da simulação, para uma dada granularidade de dados e fuso horário.
58 #Seleciona-se o período relativo ao ano de 2019.
59 times = pd.date_range(start="2019-01-01",end="2020-01-01",
60                      freq="1h",
61                      tz=location_norte.tz)
62 #Coletando as informações de temperatura, umidade relativa e vento, para os horários definidos, considerando condições atmosféricas de céu limpo.
63 clear_sky_sul = location_sul.get_clearsky(times)
64 clear_sky_norte = location_norte.get_clearsky(times)
65
66 #Execução do modelo de simulação, considerando condições atmosféricas de céu limpo e plotagem da potência CA gerada.
67 modelChain_sul.run_model(clear_sky_sul)
68 modelChain_norte.run_model(clear_sky_norte)
69
70 #Mudando o estilo de cores dos gráficos
71 plt.style.use('bmh')
72 #Criando um subplot para o gráfico relativo a geração anual no hemisfério sul
73 plt.subplot(2,1,1)
74 plt.plot(modelChain_sul.results.ac,label='Hemisferio sul')
75 plt.title('Potencia AC gerada em um ano para o Hemisferio Sul')
76 plt.xlabel('Horas')
77 plt.ylabel('Potencia AC [W]')
78 plt.ylim(0,200)
79 plt.legend()
80 #Criando um subplot para o gráfico relativo a geração anual no hemisfério norte
81 plt.subplot(2,1,2)
82 plt.plot(modelChain_norte.results.ac,label='Hemisferio norte')
83 plt.title('Potencia AC gerada em um ano para o Hemisferio Norte')
84 plt.xlabel('Horas')
85 plt.ylabel('Potencia AC [W]')
86 plt.ylim(0,200)
87 plt.legend()
88
89 plt.show()
```

EXEMPLO 1: GERAÇÃO ANUAL NOS HEMISFÉRIOS NORTE E SUL

Para esse exemplo, define-se uma simulação para todo o ano de 2019, com granularidade de dados de 1 hora. Selecionam-se duas localizações: uma posicionada na FAET-UFMT (hemisfério sul) e outra localizada próxima ao MIT (*Massachusetts Institute of Technology*) (hemisfério norte)

Os gráficos gerados podem ser observados na Figura abaixo, onde observa-se uma complementariedade entre ambas as curvas. Para o hemisfério norte (curva inferior), o pico de geração ocorre em torno da estação verão, entre os meses de junho e setembro. Já para o hemisfério sul, a mesma estação ocorre entre dezembro e março.

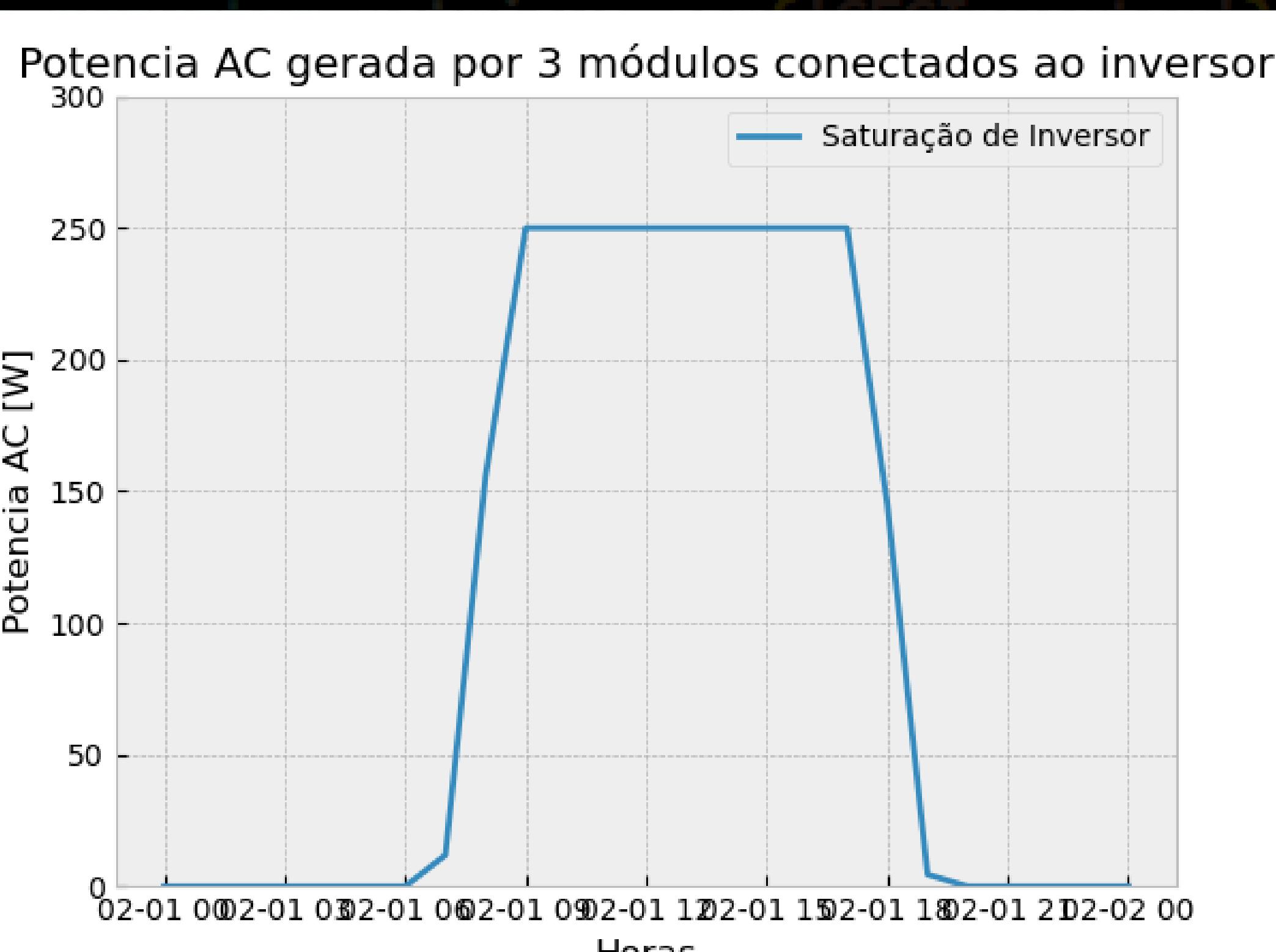


EXEMPLO 2: SATURAÇÃO DO INVERSOR

```
1 #INTRODUÇÃO AO PVLIB - Exemplo 3 - Simulação de Saturação do Inversor.
2 #
3 #Autor: Cristhian Gabriel da Rosa de Oliveira | Departamento de Engenharia Elétrica, UFMT
4 #Jun. 2023
5 #
6
7 #Importando as bibliotecas e módulos necessários para a execução do código.
8 import pvlib
9 from pvlib.modelchain import ModelChain
10 from pvlib.location import Location
11 from pvlib.pvsystem import PVSystem
12 from pvlib.temperature import TEMPERATURE_MODEL_PARAMETERS
13
14 import pandas as pd
15 import matplotlib.pyplot as plt
16
17 #Criando um objeto do tipo location, com as informações de coordenada geográfica, fuso horário, altitude e nome, que serão utilizadas pelo ModelChain. As coordenadas se referem ao endereço da FAET UFMT.
18 location = Location(latitude=-15.608259564142905,
19                      longitude=-56.06485209329807,
20                      tz='America/Fortaleza',
21                      altitude=190,
22                      name='FAET UFMT')
23
24 #Escolhendo as bases de dados de módulos e inversores a serem utilizados.
25 sandia_models = pvlib.pvsystem.retrieve_sam('SandiaMod')
26 cec_inverters = pvlib.pvsystem.retrieve_sam('CECInverter')
27 #Selecionando o modelo do módulo e inversor a ser utilizado.
28 module = sandia_models['Canadian_Solar_CS5P_220M__2009_']
29 inverter = cec_inverters['ABB_MICRO_0_25_I_OUTD_US_208__208V_']
30
31 #Definindo o modelo de desempenho de acordo com a temperatura de costa de célula.
32 temperature_parameters = TEMPERATURE_MODEL_PARAMETERS['sapm'][['open_rack_glass_glass']]
33
34 #Definindo o sistema PV, com as informações de ângulo, ajuste de temperatura, modelo de módulo e de inversor. Define-se 3 módulos conectados em série em uma string, de forma a gerar a saturação do inversor.
35 system = PVSystem(surface_tilt=45,surface_azimuth=180,
36                     module_parameters = module,
37                     inverter_parameters = inverter,
38                     temperature_model_parameters=temperature_parameters,
39                     modules_per_string=3,
40                     strings_per_inverter=1)
41
42 #Criação de um ModelChain com base nos dados do sistema PV e localização definidos.
43 modelChain = ModelChain(system,location)
44
45 #Criação de um Pandas Dataframe com os horários de início e fim da simulação, para uma dada granularidade de dados e fuso horário.
46 times = pd.date_range(start="2019-02-01",end="2019-02-02",
47                        freq="1h",
48                        tz=location.tz)
49 #Coletando as informações de temperatura, umidade relativa e vento, para os horários definidos, considerando condições atmosféricas de céu limpo.
50 clear_sky = location.get_clearsky(times)
51
52 #Execução do modelo de simulação, considerando condições atmosféricas de céu limpo e plotagem da potência CA gerada.
53 modelChain.run_model(clear_sky)
54
55 #Plotagem da potência CA gerada.
56 plt.style.use('bmh')
57 plt.plot(modelChain.results.ac,label='Saturação de Inversor')
58 plt.title('Potencia AC gerada por 3 módulos conectados ao inversor')
59 plt.xlabel('Horas')
60 plt.ylabel('Potencia AC [W]')
61 plt.ylim(0,300)
62 plt.legend()
63
64 #modelChain.results.ac.plot(figsize=(16,9))
65 plt.show()
```

EXEMPLO 2: SATURAÇÃO DO INVERSOR

Para gerar a saturação do inversor configuram-se três módulos em série (3 módulos por string, em uma string), conectados ao inversor. Os resultados são observados no gráfico abaixo, onde observa-se a saturação do inversor ocorrendo para 250 [W], cujo resultado está de acordo com o especificado no *datasheet* fornecido pelo fabricante.



EXEMPLO 3: SALVANDO OS DADOS EM ARQUIVO .CSV

```
● ● ●  
1 #INTRODUÇÃO AO PVLIB - Exemplo 4: Salva dados em arquivo .csv  
2 #  
3 #Autor: Cristhian Gabriel da Rosa de Oliveira | Departamento de Engenharia Elétrica, UFMT  
4 #Jun. 2023  
5 #  
6  
7 #Importando as bibliotecas e módulos necessários para a execução do código.  
8 import pvlib  
9 from pvlib.modelchain import ModelChain  
10 from pvlib.location import Location  
11 from pvlib.pvsystem import PVSystem  
12 from pvlib.temperature import TEMPERATURE_MODEL_PARAMETERS  
13  
14 import pandas as pd  
15 import matplotlib.pyplot as plt  
16  
17 #Criando um objeto do tipo location, com as informações de coordenada geográfica, fuso horário, altitude e nome, que serão utilizadas pelo ModelChain. As coordenadas se referem ao endereço da FAET UFMT.  
18 location = Location(latitude=-15.608259564142905,  
19                     longitude=-56.06485209329807,  
20                     tz='America/Fortaleza',  
21                     altitude=190,  
22                     name='FAET UFMT')  
23  
24 #Escolhendo as bases de dados de módulos e inversores a serem utilizados.  
25 sandia_models = pvlib.pvsystem.retrieve_sam('SandiaMod')  
26 cec_inverters = pvlib.pvsystem.retrieve_sam('CECInverter')  
27 #Selecionando o modelo do módulo e inversor a ser utilizado.  
28 module = sandia_models['Canadian_Solar_CS5P_220M__2009_']  
29 inverter = cec_inverters['ABB_MICRO_0_25_I_OUTD_US_208__208V_']  
30  
31 #Definindo o modelo de desempenho de acordo com a temperatura de costa de célula.  
32 temperature_parameters = TEMPERATURE_MODEL_PARAMETERS['sapm']['open_rack_glass_glass']  
33  
34 #Definindo o sistema PV, com as informações de ângulo, ajuste de temperatura, modelo de módulo e de inversor, bem como definindo a quantidade de módulos utilizados por string e a quantidade de strings adotadas.  
35 system = PVSystem(surface_tilt=45,surface_azimuth=180,  
36                     module_parameters = module,  
37                     inverter_parameters = inverter,  
38                     temperature_model_parameters=temperature_parameters,  
39                     modules_per_string=1,  
40                     strings_per_inverter=1)  
41  
42 #Criação de um ModelChain com base nos dados do sistema PV e localização definidos.  
43 modelChain = ModelChain(system,location)  
44  
45 #Criação de um Pandas Dataframe com os horários de início e fim da simulação, para uma dada granularidade de dados e fuso horário.  
46 times = pd.date_range(start="2019-01-01",end="2019-01-02",  
47                      freq="1h",  
48                      tz=location.tz)  
49 #Coletando as informações de temperatura, umidade relativa e vento, para os horários definidos, considerando condições atmosféricas de céu limpo.  
50 clear_sky = location.get_clearsky(times)  
51  
52 #Execução do modelo de simulação, considerando condições atmosféricas de céu limpo e plotagem da potência CA gerada.  
53 modelChain.run_model(clear_sky)  
54  
55 #Salva os valores de results.ac gerados em um arquivo csv com título "curva_AC"  
56 modelChain.results.ac.to_csv('curva_AC.csv')  
57 plt.show()
```

modelChain(system,location)

date_range(start="2019-01-01",end="2019-01-02",
freq="1h",
tz=location.tz)

EXEMPLO 3: SALVANDO OS DADOS EM ARQUIVO .CSV

Após a execução do ModelChain, através do comando `to_csv` é possível salvar os resultados gerados em um arquivo do tipo CSV. A biblioteca irá salvar os dados em duas colunas, sendo o da esquerda relativo ao dia e horário simulados e, na direita, a **potência gerada**, em Watts. Em horários sem geração um valor negativo estará presente, sendo esse relativo ao **consumo de potência** do arranjo, necessário para que esse se mantenha ligado.

11	2019-01-01 09:00:00-03:00	105.79702705612479
12	2019-01-01 10:00:00-03:00	130.75987256151885
13	2019-01-01 11:00:00-03:00	147.3684068034921
14	2019-01-01 12:00:00-03:00	156.5444109634217
15	2019-01-01 13:00:00-03:00	158.6181408277286
16	2019-01-01 14:00:00-03:00	153.65876748554768
17	2019-01-01 15:00:00-03:00	141.49901197294616
18	2019-01-01 16:00:00-03:00	121.63964453191855
19	2019-01-01 17:00:00-03:00	92.66223287809017

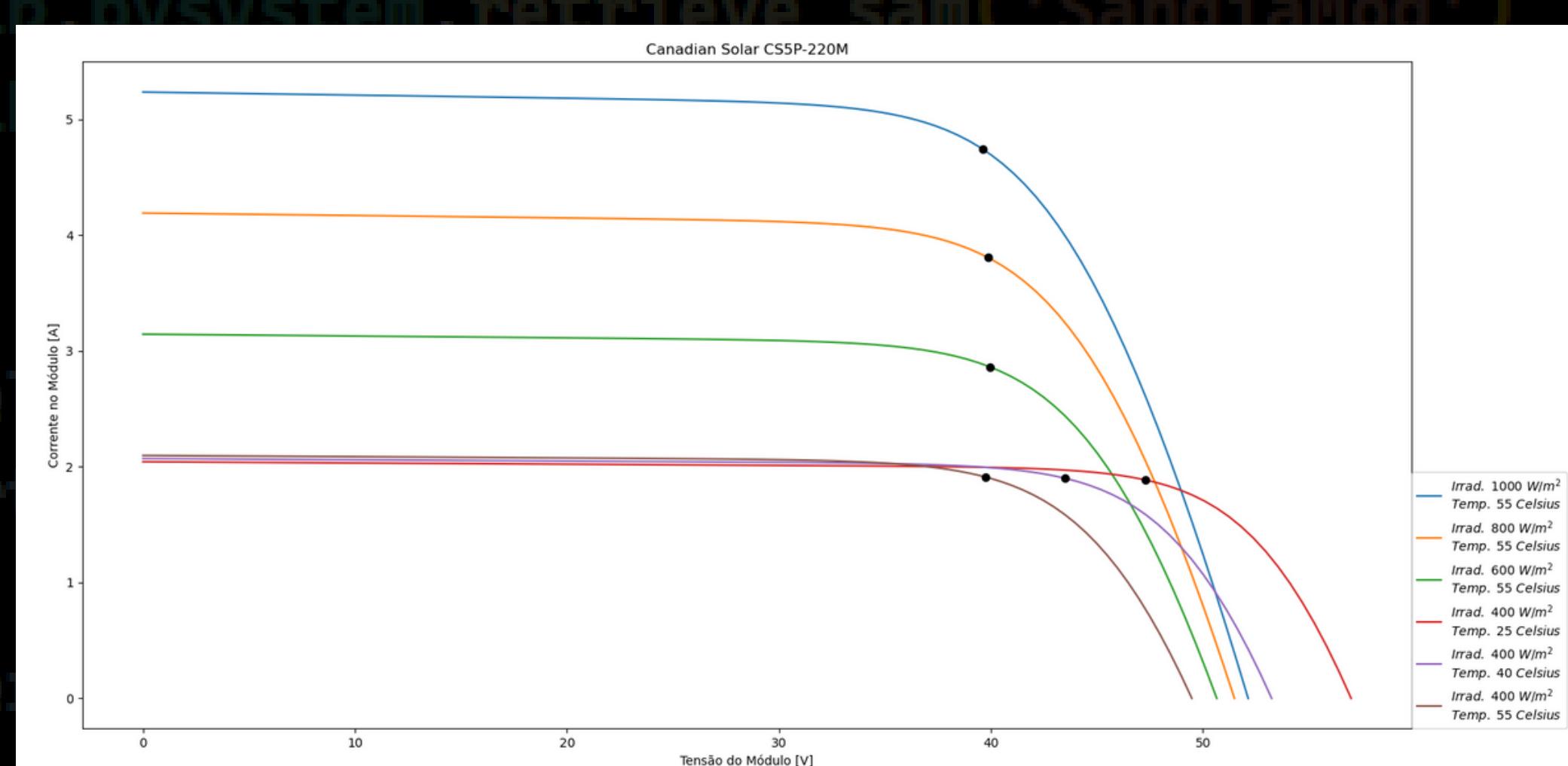
EXEMPLO 4: CURVA IV

```
●●●

1 #Importando as bibliotecas e módulos necessários para a execução do código.
2 from pvlib import pvsystem
3 import pandas as pd
4 import matplotlib.pyplot as plt
5
6 #Definindo os parâmetros do módulo usado para traçar a curva IV do arranjo.
7 # Dados do módulo Canadian Solar CS5P-220M.
8 parametros = {
9     'Name': 'Canadian Solar CS5P-220M',
10    'BIPV': 'N',
11    'Date': '10/5/2009',
12    'T_NOCT': 42.4,
13    'A_c': 1.7,
14    'N_s': 96,
15    'I_sc_ref': 5.1,
16    'V_oc_ref': 59.4,
17    'I_mp_ref': 4.69,
18    'V_mp_ref': 46.9,
19    'alpha_sc': 0.004539,
20    'beta_oc': -0.22216,
21    'a_ref': 2.6373,
22    'I_L_ref': 5.114,
23    'I_o_ref': 8.196e-10,
24    'R_s': 1.065,
25    'R_sh_ref': 381.68,
26    'Adjust': 8.7,
27    'gamma_r': -0.476,
28    'Version': 'MM106',
29    'PTC': 200.1,
30    'Technology': 'Mono-c-Si',
31 }
31 #Definindo os casos nos quais a curva IV será traçada, sendo o par de valores referente, respectivamente, a nível de irradiação ( $\text{W/m}^2$ ) e temperatura ( $^{\circ}\text{C}$ ), e armazenando-os em um Pandas Dataframe.
32 casos = [
33     (1000, 55),
34     (800, 55),
35     (600, 55),
36     (400, 25),
37     (400, 40),
38     (400, 55)
39 ]
40 condicoes = pd.DataFrame(casos, columns=['Irrad.', 'Temp.'])
41
42 # Ajusta os parâmetros de referência de acordo com os casos definidos usando o modelo descrito por De Soto no artigo:
43 # W. De Soto et al., "Improvement and validation of a model for photovoltaic array performance", Solar Energy, vol 80, pp. 78-88, 2006.
44 IL, I0, Rs, Rsh, nNsVth = pvsystem.calcparms_desoto(
45     condicoes['Irrad.'],
46     condicoes['Temp.'],
47     alpha_sc=parametros['alpha_sc'],
48     a_ref=parametros['a_ref'],
49     I_L_ref=parametros['I_L_ref'],
50     I_o_ref=parametros['I_o_ref'],
51     R_sh_ref=parametros['R_sh_ref'],
52     R_s=parametros['R_s'],
53     EgRef=1.121,
54     dEgdT=-0.0002677
55 )
56 #Anexando os parâmetros no objeto PVSystem para cálculo das curvas IV.
57 curve_info = pvsystem.singlediode(
58     photocurrent=IL,
59     saturation_current=I0,
60     resistance_series=Rs,
61     resistance_shunt=Rsh,
62     nNsVth=nNsVth,
63     ivcurve_pnts=100,
64     method='Lambertw'
65 )
66
67 # Plotagem das curvas IV
68 plt.figure()
69 for i, caso in condicoes.iterrows():
70     label = (
71         f"$Irrad. \$ " + f"${caso['Irrad.']} \text{ W/m}^2\$ \n"
72         f"$Temp. \$ " + f"${caso['Temp.']} \text{ }^{\circ}\text{C}\$"
73     )
74     plt.plot(curve_info['v'][i], curve_info['i'][i], label=label)
75     v_mp = curve_info['v_mp'][i]
76     i_mp = curve_info['i_mp'][i]
77     # mark the MPP
78     plt.plot([v_mp], [i_mp], ls='', marker='o', c='k')
79
80 plt.legend(loc=(1.0, 0))
81 plt.xlabel('Tensão do Módulo [V]')
82 plt.ylabel('Corrente no Módulo [A]')
83 plt.title(parametros['Name'])
84 plt.show()
85 plt.gcf().set_tight_layout(True)
86
87 #Printando os resultados obtidos de tensão, corrente e potência.
88 print(pd.DataFrame({
89     'i_sc': curve_info['i_sc'],
90     'v_oc': curve_info['v_oc'],
91     'i_mp': curve_info['i_mp'],
92     'v_mp': curve_info['v_mp'],
93     'p_mp': curve_info['p_mp'],
94 }))
```

EXEMPLO 4: CURVA IV

Para traçar a *curva IV* definem-se os parâmetros do módulo o qual deseja-se traçar a curva. Após isso, utiliza-se do modelo proposto por *De Soto* para calcular os parâmetros elétricos da curva IV, para uma dada temperatura e irradiação, baseado nas características do módulo. Então, calcula-se a curva IV por meio da solução da equação *single diode*, usando o método *Lambert W*.



Cabe ressaltar que outros modelos e equacionamentos são possíveis. A lista completa pode ser encontrada na documentação da biblioteca.

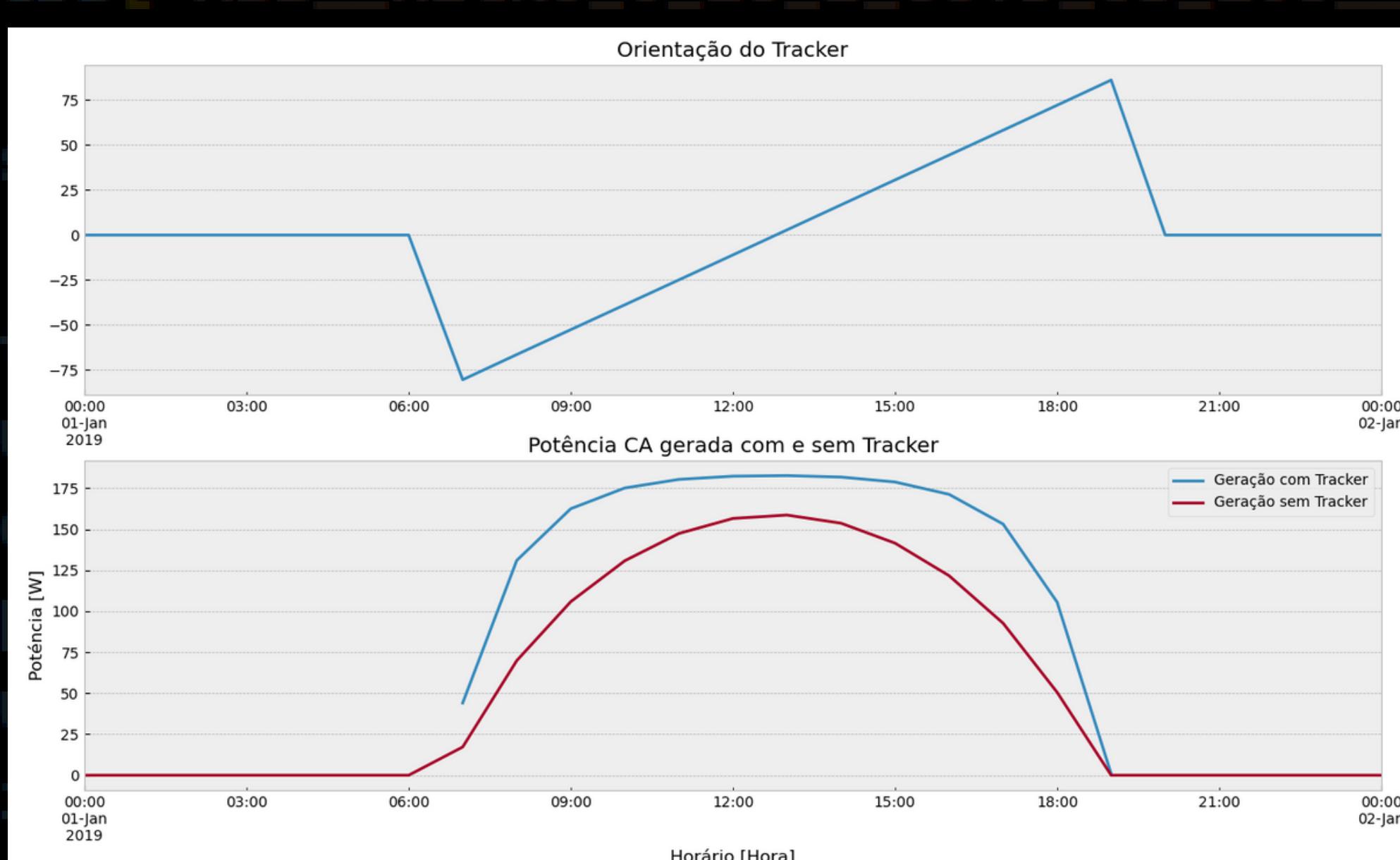
EXEMPLO 5: TRACKER DE EIXO ÚNICO

```
1 #INTRODUÇÃO AO PVLIB - Exemplo 6: Simulando o efeito de um Tracker de Eixo único.
2 #
3 #Autor: Cristhian Gabriel da Rosa de Oliveira | Departamento de Engenharia Elétrica, UFMT
4 #Jun. 2023
5 #
6
7 #Importando as bibliotecas e módulos necessários para a execução do código.
8 #Em especial, no módulo pvsystem, importamos o método SingleAxisTrackerMount, que permite definir um tracker de eixo Único.
9 import pvlib
10 from pvlib.modelchain import ModelChain
11 from pvlib.location import Location
12 from pvlib.pvsystem import PVSystem, Array, SingleAxisTrackerMount
13 from pvlib.temperature import TEMPERATURE_MODEL_PARAMETERS
14
15 import pandas as pd
16 import matplotlib.pyplot as plt
17
18 #Criando um objeto do tipo location, com as informações de coordenada geográfica, fuso horário, altitude e nome, que serão utilizadas pelo ModelChain. As coordenadas se referem ao endereço da FAET UFMT.
19 location = Location(latitude=-15.608259564142905,
20                      longitude=-56.06485209329807,
21                      tz='America/Fortaleza',
22                      altitude=190,
23                      name='FAET UFMT')
24
25 #Escolhendo as bases de dados de módulos e inversores a serem utilizados.
26 sandia_models = pvlib.pvsystem.retrieve_sam('SandiaMod')
27 cec_inverters = pvlib.pvsystem.retrieve_sam('CECInverter')
28 #Selecionando o modelo do módulo e inverter a ser utilizado.
29 module = sandia_models['Canadian_Solar_CSSP_220M__2009_']
30 inverter = cec_inverters['ABB_MICRO_0_25_I_OUTD_US_208__208V_']
31
32 #Definindo o modelo de desempenho de acordo com a temperatura de costa de célula.
33 temperature_parameters = TEMPERATURE_MODEL_PARAMETERS['sapm']['open_rack_glass_glass']
34
35 #Definindo o sistema PV, com as informações de ângulo, ajuste de temperatura, modelo de módulo e de inverter, bem como definindo a quantidade de módulos utilizados por string e a quantidade de strings adotadas.
36 system = PVSystem(surface_tilt=45,surface_azimuth=180,
37                     module_parameters = module,
38                     inverter_parameters = inverter,
39                     temperature_model_parameters=temperature_parameters,
40                     modules_per_string=1,
41                     strings_per_inverter=1)
42
43 #Criação de um ModelChain com base nos dados do sistema PV e localização definidos.
44 modelChain = ModelChain(system,location)
45
46 #Criação de um Pandas Dataframe com os horários de início e fim da simulação, para uma dada granularidade de dados e fuso horário.
47 times = pd.date_range(start="2019-01-01",end="2019-01-02",
48                       freq="1h",
49                       tz=location.tz)
50 #Coletando as informações de temperatura, umidade relativa e vento, para os horários definidos, considerando condições atmosféricas de céu limpo.
51 clear_sky = location.get_clearsky(times)
52
53 #Definindo os parâmetros do Tracker de Eixo Único.
54 tracker = SingleAxisTrackerMount(axis_tilt = 0,
55                                   axis_azimuth=180,
56                                   max_angle = 90,
57                                   backtrack = False)
58
59 #Coletando a posição do sol que será usada para orientar o sistema de Tracker.
60 posicao_sol = location.get_solarposition(times)
61 #Definindo os valores de ângulo do Tracker de Eixo Único, conforme a posição do sol.
62 orientacao = tracker.get_orientation(solar zenith=posicao_sol['apparent zenith'],
63                                       solar azimuth=posicao_sol['azimuth'])
64
65 plt.style.use('bmh')
66
67 #Plotando o ângulo de orientação do Tracker ao longo de um dia.
68 plt.subplot(2,1,1)
69 orientacao['tracker_theta'].fillna(0).plot(title="Orientação do Tracker")
70
71 #Definindo o array de módulos conectados ao tracker
72 array = Array(mount=tracker,module_parameters=module,
73                temperature_model_parameters=temperature_parameters,
74                modules_per_string=1,strings=1)
75
76 #Criando a Modelchain do modelo que simula a atuação do Tracker de Eixo Único.
77 system_tracker = PVSystem(arrays=[array],inverter_parameters=inverter)
78 modelChain_tracker = ModelChain(system_tracker,location)
79
80 #Realizando a simulação de ambos os modelos para comparação.
81 modelChain_tracker.run_model(clear_sky)
82 modelChain.run_model(clear_sky)
83
84 #Plotando os resultados de potência CA gerada com e sem Tracker.
85 plt.subplot(2,1,2)
86 modelChain_tracker.results.ac.plot(figsize=(16,9))
87 modelChain.results.ac.plot(figsize=(16,9))
88 plt.legend(['Geração com Tracker', 'Geração sem Tracker'])
89
90 plt.show()
```

EXEMPLO 5: TRACKER DE EIXO ÚNICO

Trackers de eixo único são dispositivos que permitem que os painéis solares acompanhem o movimento do sol ao longo do dia, ajustando sua posição para maximizar a captura da luz solar, aumentando a produção de energia em uma planta de geração fotovoltaica.

Os gráficos abaixo apresentam a variação de ângulo de orientação do tracker ao longo de um dia (gráfico superior), onde observa-se o retorno à posição inicial em torno do meio dia, conforme esperado. O gráfico inferior exibe a diferença de potência gerada com e sem o uso de tracker, onde observa-se uma maior energia aproveitada no primeiro cenário.



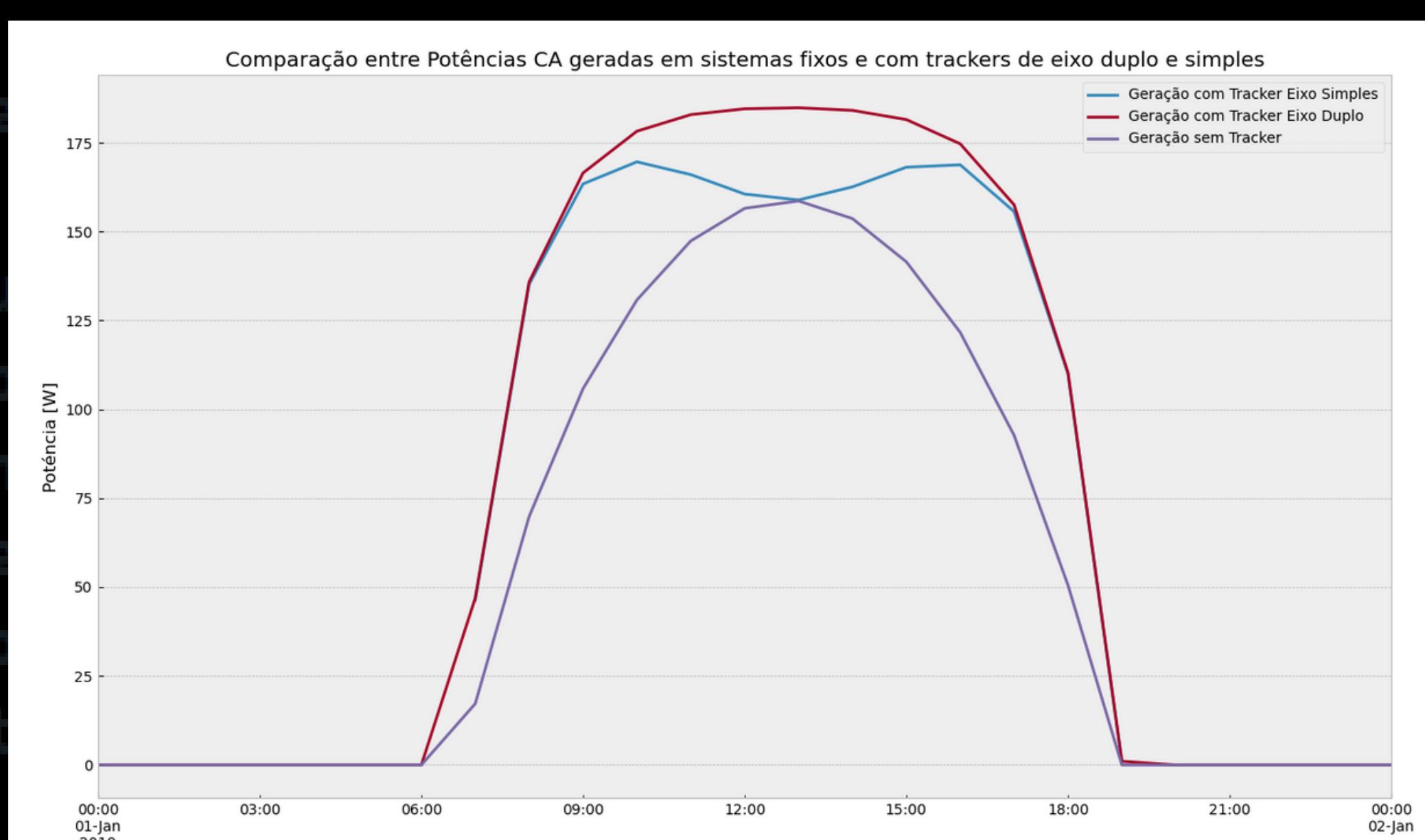
EXEMPLO 6: TRACKER DE EIXO DUPLO

```
●●●
1 #Importando as bibliotecas e módulos necessários para a execução do código.
2 #Em especial, no módulo pvsystem, importamos o método SingleAxisTrackerMount, que permite de
finir um tracker de eixo Único.
3 import pvlib
4 from pvlib.modelchain import ModelChain
5 from pvlib.location import Location
6 from pvlib.pvsystem import PVSystem, Array, SingleAxisTrackerMount, AbstractMount
7 from pvlib.temperature import TEMPERATURE_MODEL_PARAMETERS
8
9 import pandas as pd
10 import matplotlib.pyplot as plt
11
12 #Criando um objeto do tipo location, com as informações de coordenada geográfica, fuso horá
rio, altitude e nome, que serão utilizadas pelo ModelChain. As coordenadas se referem ao endre
ço da FAET UFMT.
13 location = Location(latitude=-15.608259564142905,
14                      longitude=-56.06485209329807,
15                      tz='America/Fortaleza',
16                      altitude=190,
17                      name='FAET UFMT')
18
19 #Escolhendo as bases de dados de módulos e inversores a serem utilizados.
20 sandia_models = pvlib.pvsystem.retrieve_sam('SandiaMod')
21 cec_inverters = pvlib.pvsystem.retrieve_sam('CECInverter')
22 #Selecionando o modelo do módulo e inverSOR a ser utilizado.
23 module = sandia_models['Canadian_Solar_CSSP_220M__2009_']
24 inverter = cec_inverters['ABB_MICRO_0_25_I_OUTD_US_208_208V_']
25
26 #Definindo o modelo de desempenho de acordo com a temperatura de costa de célula.
27 temperature_parameters = TEMPERATURE_MODEL_PARAMETERS['sapm']['open_rack_glass_glass']
28
29 #Definindo o sistema PV, com as informações de ângulo, ajuste de temperatura, modelo de mód
ulo e de inverSOR, bem como definindo a quantidade de módulos utilizados por string e a quant
idade de strings adotadas.
30 system = PVSystem(surface_tilt=45,surface_azimuth=180,
31                     module_parameters = module,
32                     inverter_parameters = inverter,
33                     temperature_model_parameters=temperature_parameters,
34                     modules_per_string=1,
35                     strings_per_inverter=1)
36
37 #Criação de um ModelChain com base nos dados do sistema PV e localização definidos.
38 modelChain = ModelChain(system,location)
39
40 #Criação de um Pandas Dataframe com os horários de início e fim da simulação, para uma dada
granularidade de dados e fuso horário.
41 times = pd.date_range(start="2019-01-01",end="2019-01-02",
42                       freq="1h",
43                       tz=location.tz)
44 #Coletando as informações de temperatura, umidade relativa e vento, para os horários definid
os, considerando condições atmosféricas de céu limpo.
45 clear_sky = location.get_clearsky(times)
46
47 #Definindo os parâmetros do Tracker de Eixo Único.
48 tracker = SingleAxisTrackerMount(axis_tilt = 45,
49                                   axis_azimuth=180,
50                                   max_angle = 90,
51                                   backtrack = False)
52
53 #Coletando a posição do sol que será usada para orientar o sistema de Tracker.
54 posicao_sol = location.get_solarposition(times)
55 #Definindo os valores de ângulo do Tracker de Eixo Único, conforme a posição do sol.
56 orientacao = tracker.get_orientation(solar zenith=posicao_sol['apparent zenith'],
57                                       solar azimuth=posicao_sol['azimuth'])
58
59 #Definindo o array de módulos conectados ao tracker
60 array = Array(mount=tracker,module_parameters=module,
61                 temperature_model_parameters=temperature_parameters,
62                 modules_per_string=1,strings=1)
63
64 #Criando a Modelchain do modelo que simula a atuação do Tracker de Eixo Único.
65 system_tracker = PVSystem(arrays=[array],inverter_parameters=inverter)
66 modelChain_tracker = ModelChain(system_tracker,location)
67
68 #Definindo a classe que implementa o Tracker de Eixo Duplo.
69 class DualAxisTrackerMount(AbstractMount):
70     def get_orientation(self, solar zenith, solar azimuth):
71         return {'surface_tilt': solar zenith,'surface_azimuth':solar azimuth}
72
73 tracker_eixo_duplo = DualAxisTrackerMount()
74
75 #Definindo o array de módulos conectados ao tracker de eixo duplo
76 array_tracker_duplo = Array(mount=tracker_eixo_duplo,module_parameters=module,
77                               temperature_model_parameters=temperature_parameters,
78                               modules_per_string=1,strings=1)
79
80 #Criando o PVSystem do modelo de tracker em eixo duplo
81 system_tracker_duplo = PVSystem(arrays=[array_tracker_duplo],inverter_parameters=inverter)
82 modelChain_tracker_duplo = ModelChain(system_tracker_duplo,location)
83
84 #Realizando a simulação dos três modelos para comparação.
85 modelChain_tracker_duplo.run_model(clear_sky)
86 modelChain_tracker.run_model(clear_sky)
87 modelChain.run_model(clear_sky)
88
89 #Plotando os resultados de potência CA gerada com e sem Tracker.
90 plt.style.use('bmh')
91 modelChain_tracker.results.ac.plot(figsize=(16,9))
92 modelChain_tracker_duplo.results.ac.plot(figsize=(16,9))
93 modelChain.results.ac.plot(figsize=(16,9))
94 plt.legend(['Geração com Tracker Eixo Simples','Geração com Tracker Eixo Duplo','
Geração sem Tracker'])
95 plt.title('
Comparação entre Potências CA geradas em sistemas fixos e com trackers de eixo duplo e simpl
es
')
96 plt.ylabel('Potência [W]')
97 plt.xlabel('Horário [Hora]')
98
99 plt.show()
```

EXEMPLO 6: TRACKER DE EIXO DUPLO

Trackers de eixo duplo permitem que os painéis solares acompanhem tanto o movimento do sol ao longo do dia quanto a variação da altura do sol ao longo do ano. Isso permite que os painéis solares ajustem sua posição verticalmente, otimizando ainda mais a captura da luz solar. Diferentemente dos trackers de eixo único, os trackers de eixo duplo proporcionam uma maior produção de energia ao longo do ano.

No gráfico abaixo é possível observar a **potência adicional gerada**, proporcionada pelo tracker de eixo duplo, em comparação com o de eixo simples. Em especial, em horários próximos ao pico de geração.



```
elchain import ModelChain
ation import Location
ystem import PVSystem
operatore import TEMPERATURE_MODEL_PARAMETERS

as pd
lib.pyplot as plt

AET UFMT
ation(latitude=-15.608259564142905,
longitude=-56.06485209329807,
tz='America/Fortaleza',
altitude=190,
name='FAET UFMT')

= pvli ADICIONANDO DADOS
= pvli
  _mode
  _inver
parameters = TEMPERATURE_MODEL_PARAMETERS['sapm']['op
cem(surface_tilt=45,surface_azimuth=180,
module_parameters = module,
inverter_parameters = inverter,
temperature_model_parameters=temperature_params,
modules_per_string=1,
strings_per_inverter=1)

odelChain(system,location)
e_range(start="2019-01-01",end="2019-01-02",
freq="1h",
```

O QUE É O PVGIS?

PVGIS (*Photovoltaic Geographical Information System*) é uma ferramenta vinculada ao EU Science Hub, da comissão europeia, que fornece informação sobre radiação solar e performance de sistemas fotovoltaicos para qualquer localização na Europa e África, além de grande parte da Ásia e América.

Essa ferramenta utiliza de dados de radiação solar de alta qualidade, obtidos de imagens de satélite, além de dados de temperatura e velocidade de vento a partir de modelos climáticos.



Os modelos são validados por meio de medições realizadas em módulos comerciais localizadas no JRC'S European Solar Test Installation, sendo essa instalação certificada pela ISO 17025.

TYPICAL METEOROLOGICAL YEAR

Uma das ferramentas disponibilizadas pelo *PVGIS* é a chamada *TMY (Typical Meteorological Year) Generator*. Em tradução livre, um ano meteorológico típico refere-se a um conjunto de dados de um ano de referência para uma dada localidade, obtidos através de dados de medições reais. Dentre os dados incluem-se, mas não apenas:

- Data e hora em formato UTC;
- Temperatura do ar;
- Umidade Relativa;
- Irradiâncias global horizontal, direta e difusa;
- Radiação infravermelha;
- Velocidade e direção do vento;
- Pressão do ar na superfície;

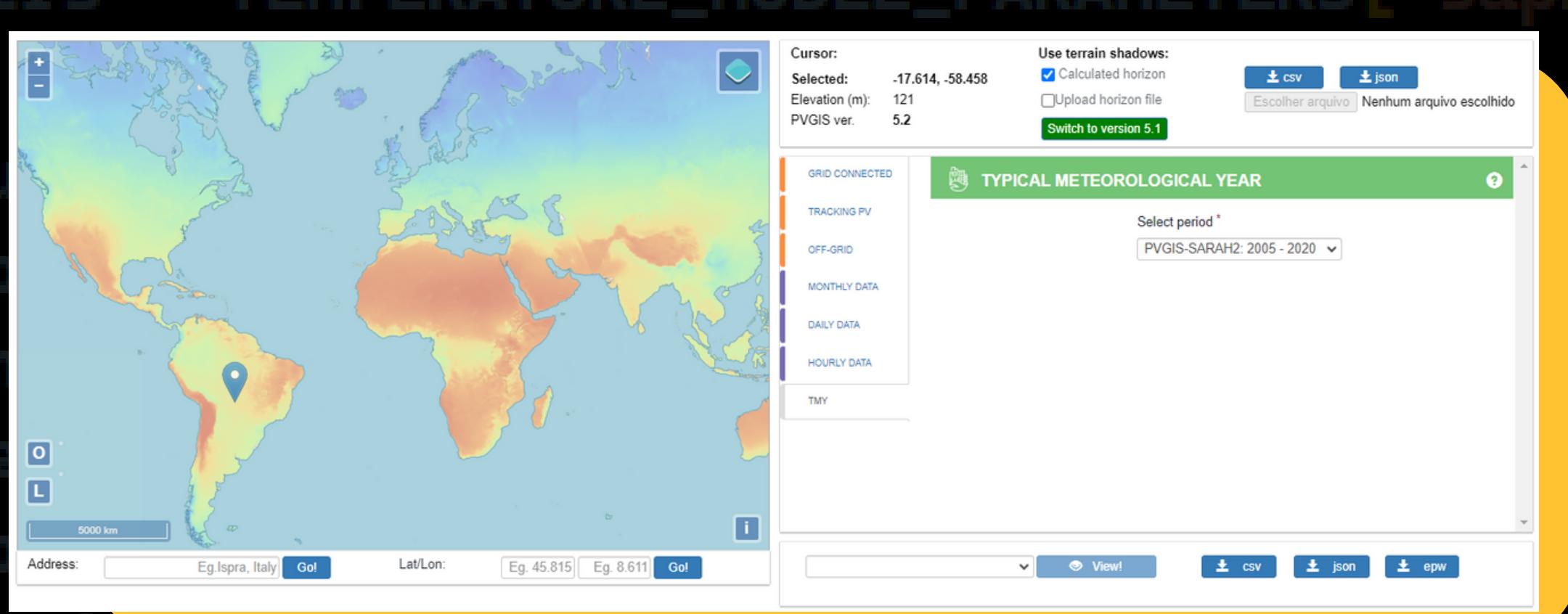
A partir desse conjunto de dados, aplicados como entrada para a biblioteca *PVLIB* é possível gerar uma estimação de geração solar muito mais realística. A seguir, veja como realizar essa junção.

PRIMEIRO PASSO: OBTENDO OS DADOS TMY

Primeiramente, é necessário que se obtenham os dados referentes ao ano meteorológico típico para a localização de interesse. Para isso, acesse o *TMY Generator*, disponível no link abaixo:

https://re.jrc.ec.europa.eu/pvg_tools/en/

Em seguida, selecione as coordenadas da localização que deseja-se realizar a simulação, ou manualmente com o cursos do mouse, ou inserindo as coordenadas na caixa de texto disponível na parte inferior do mapa. Após a seleção, baixe os dados no formato de arquivo de preferência. Nesse exemplo, usaremos dados em formato .csv



SEGUNDO PASSO: PROCESSANDO OS DADOS

Após mover o arquivo .csv para a pasta de seu projeto, será necessário um pré-processamento dos dados fornecidos para que esses possam ser lidos pelo PVLIB.

Longitude=-56.06485209329807,

O intuito desse pré-processamento é gerar um novo arquivo .csv que contenha apenas as informações, no formato correto, que o PVLIB irá usar. O código abaixo realiza a filtragem dos dados de tempo, temperatura do ar, irradiâncias e velocidade do vento, ignorando as 16 primeiras linhas que contém informações não relevantes para a simulação.

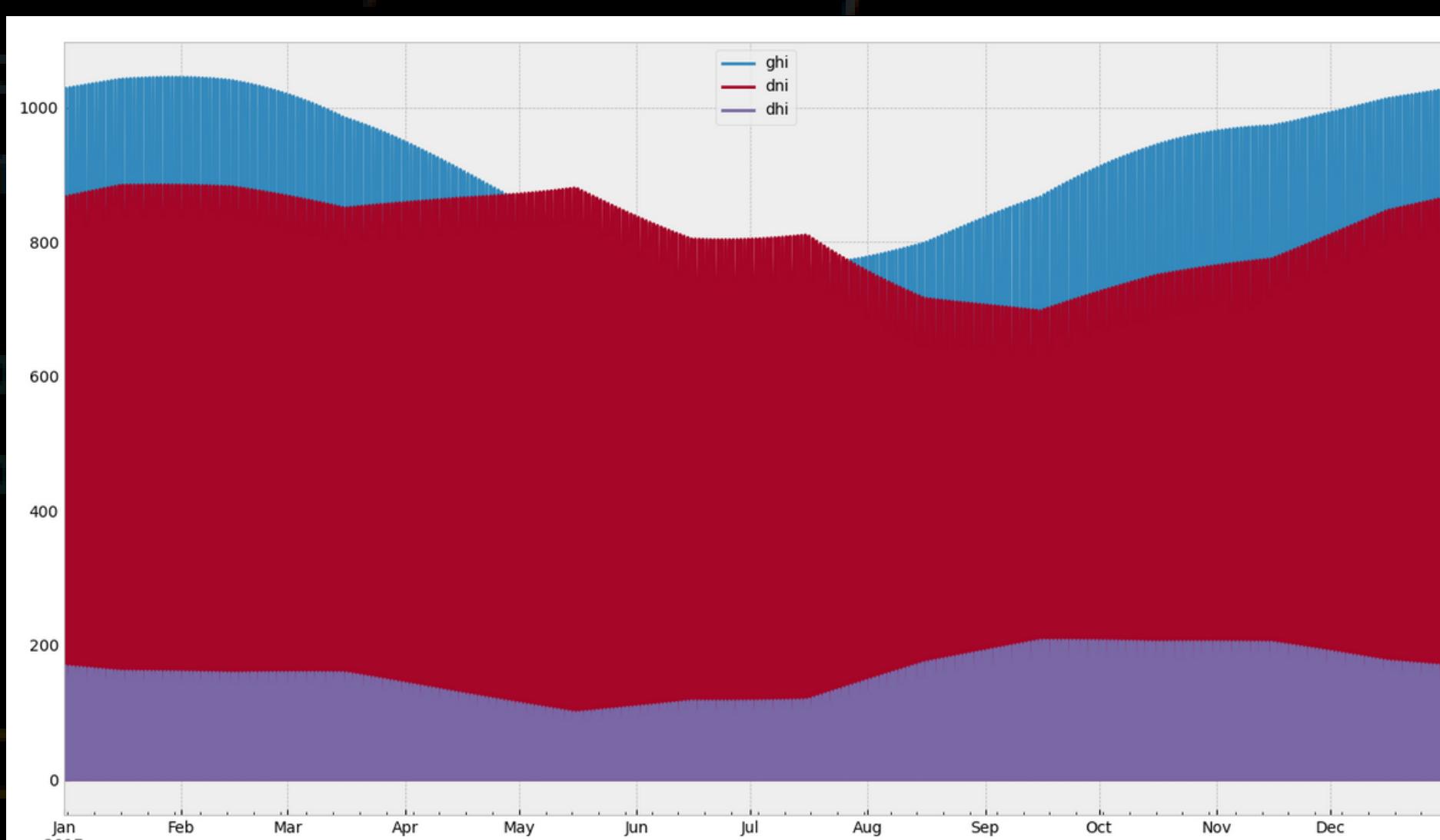
```
● ● ●
1 import pandas as pd
2 import matplotlib.pyplot as plt
3
4 #Lendo as colunas referentes a tempo, temperatura do ar, irradiância e velocidade do vento do arquivo .csv
5 #Pulam-se as 16 primeiras linhas, por conterem informações que não são relevantes para o cálculo.
6 tmy = pd.read_csv("IntroducaoAoPVLIB\Exemplos\TMY_FAET_UFMT.csv",skiprows=16,nrows=8770,
7                   usecols=["time(UTC)","T2m","G(h)","Gb(n)","Gd(h)","WS10m"],
8                   index_col=0)
9
10 #Plotando os dados meteorológicos obtidos
11 plt.style.use('bmh')
12 tmy.plot(figsize=(16,9))
```

modules_per_string=1,
strings_per_inverter=1)

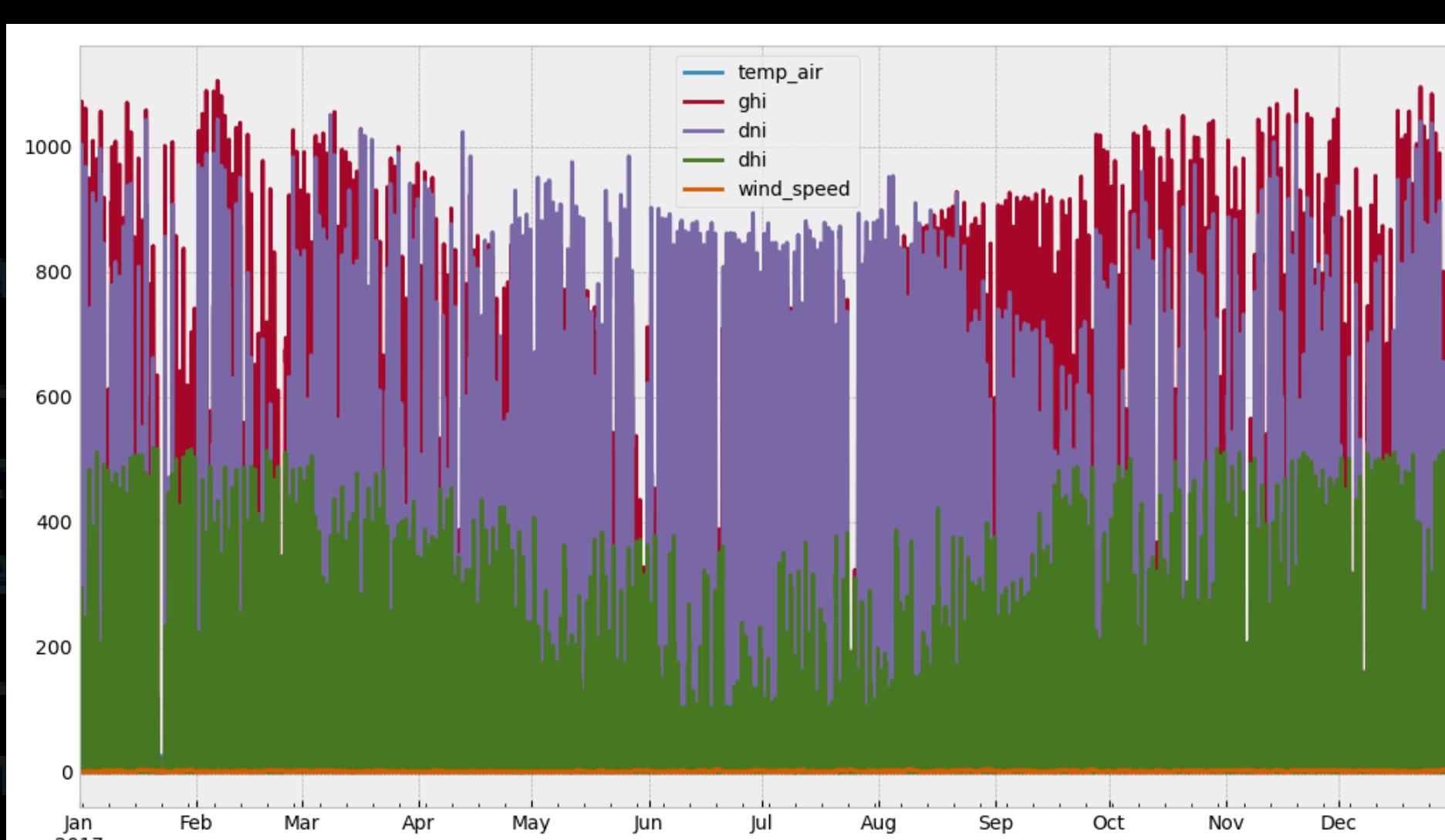
SEGUNDO PASSO: PROCESSANDO OS DADOS

Ao executar-se o código é possível observarmos as curvas de irradiância obtidas usando dados TMY e compará-las com os dados em condições de céu limpo.

Considerando condições de céu limpo:



Considerando dados obtidos do PVGIS:



SEGUNDO PASSO: PROCESSANDO OS DADOS

Embora hajam semelhanças entre os gráficos, é nítida a maior intermitência obtida quando se consideram condições atmosféricas realísticas.

Os dados obtidos podem ser salvos em um arquivo em formato **.csv** que será lido pelo **PVLIB** para a estimação de potência CA gerada no período determinado. Para salvar os dados utiliza-se o código abaixo:

```
# Selecionando os dados TMY referentes a um período desejado, nesse caso, o ano de 2017.  
tmy.index=pd.date_range(start="2017-01-01 00:00", end="2017-12-31 23:00", freq="h")  
  
# Alterando o nome das colunas para o formato que o pvlib irá interpretar.  
tmy.columns = ["temp_air", "ghi", "dni", "dhi", "wind_speed"]  
  
# Salvando os dados TMY para leitura do PVLIB  
tmy.to_csv("IntroducaoAoPVLIB\Exemplos\PVLIB_TMY.csv")
```

Dessa forma, o processamento dos dados TMY para leitura pelo PVLIB se resume em *filtrar um intervalo de tempo desejado*, entre todos os dados disponíveis e salvá-los em um arquivo **.csv** cujas colunas possuam um nome interpretável pelo PVLIB. **_per_inverter=1**)

SEGUNDO PASSO: PROCESSANDO OS DADOS

O código completo para processamento dos dados obtidos e plotagem das curvas de dados meteorológicos está disponível abaixo:

```
tz='America/Fortaleza',
altitude=190,
name='FAET UFMT')
```

```
● ● ●
1 #Importando as bibliotecas e módulos necessários para a execução do código.
2 import pandas as pd
3 import matplotlib.pyplot as plt
4
5 #Lendo as colunas referentes a tempo, temperatura do ar, irradiação e velocidade do vento do arquivo .csv
6 #Pulam-se as 16 primeiras linhas, por conterem informações que não são relevantes para o cálculo.
7 tmy = pd.read_csv("IntroducaoAoPVLIB\Exemplos\TMY_FAET_UFMT.csv",skiprows=16,nrows=8760,
8                   usecols=["time(UTC)","T2m","G(h)","Gb(n)","Gd(h)","WS10m"],
9                   index_col=0)
10
11 #Selecionando os dados TMY referentes a um período desejado, nesse caso, o ano de 2017.
12 tmy.index=pd.date_range(start="2017-01-01 00:00", end="2017-12-31 23:00",freq="h")
13
14 #Alterando o nome das colunas para o formato que o pvlib irá interpretar.
15 tmy.columns = ["temp_air","ghi","dni","dhi","wind_speed"]
16
17 #Salvando os dados TMY para leitura do PVLIB
18 tmy.to_csv("IntroducaoAoPVLIB\Exemplos\PVLIB_TMY.csv")
19
20 #Plotando os dados meteorológicos obtidos
21 plt.style.use('bmh')
22 tmy.plot(figsize=(16,9))
```

```
inverter_parameters = inverter,
temperature_model_parameters=temperature_parameters,
modules_per_string=1,
strings_per_inverter=1)
```

```
modelChain(system,location)
```

```
e_range(start="2019-01-01",end="2019-01-02",
freq="1h",
```

TERCEIRO PASSO: UTILIZAÇÃO DOS DADOS NO PVLIB

Agora, deve-se vincular o arquivo .csv gerado na etapa de processamento com a biblioteca, para que essa possa ler os dados obtidos e processar a simulação os utilizando.

Considerando código de simulação a céu limpo inicialmente desenvolvido, as alterações ocorrem após a definição da ModelChain, substituindo a aquisição de dados a céu limpo pela leitura de dados do arquivo gerado, conforme o código abaixo.

```
#Lendo o arquivo .CSV com dados TMY processados.
tmy = pd.read_csv("IntroducaoAoPVLIB\Exemplos\PVLIB_TMY.csv", index_col=0)
tmy.index = pd.to_datetime(tmy.index)

#Execução do modelo de simulação, considerando condições atmosféricas obtidas a partir de dados TMY e plotagem da potência CA gerada.
modelChain.run_model(tmy)
plt.subplot(2,1,1)
modelChain.results.ac.plot(figsize=(16,9))

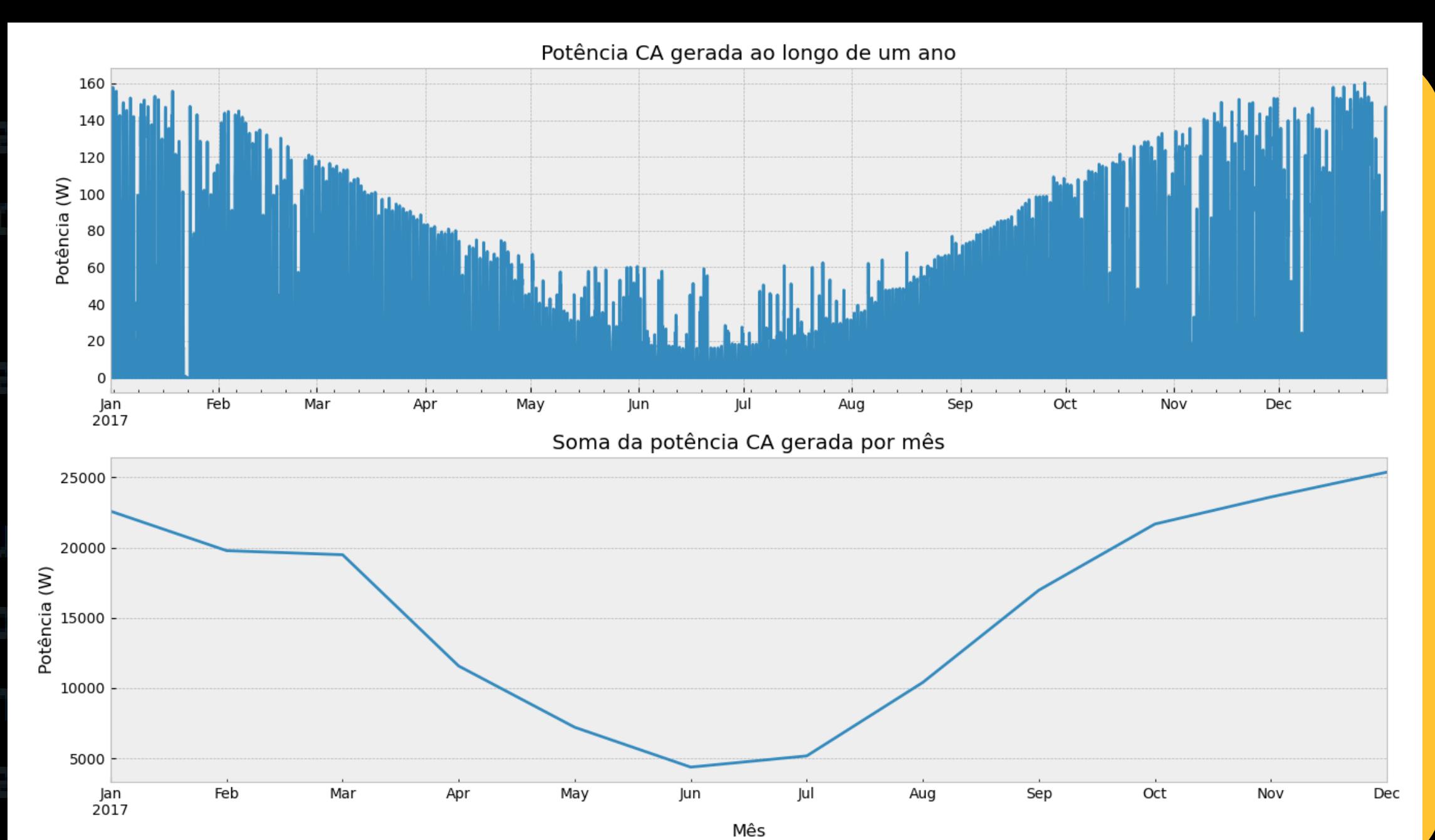
#Plotando a soma da energia CA gerada por mês.
plt.subplot(2,1,2)
modelChain.results.ac.resample("M").sum().plot(figsize=(16,9))

modules_per_string=1,
strings_per_inverter=1)
```

TERCEIRO PASSO: UTILIZAÇÃO DOS DADOS NO PVLIB

Dessa forma, a potência CA gerada ao longo de um ano (gráfico superior) para a localização da FAET, considerando dados meteorológicos realísticos, apresentará o comportamento abaixo.

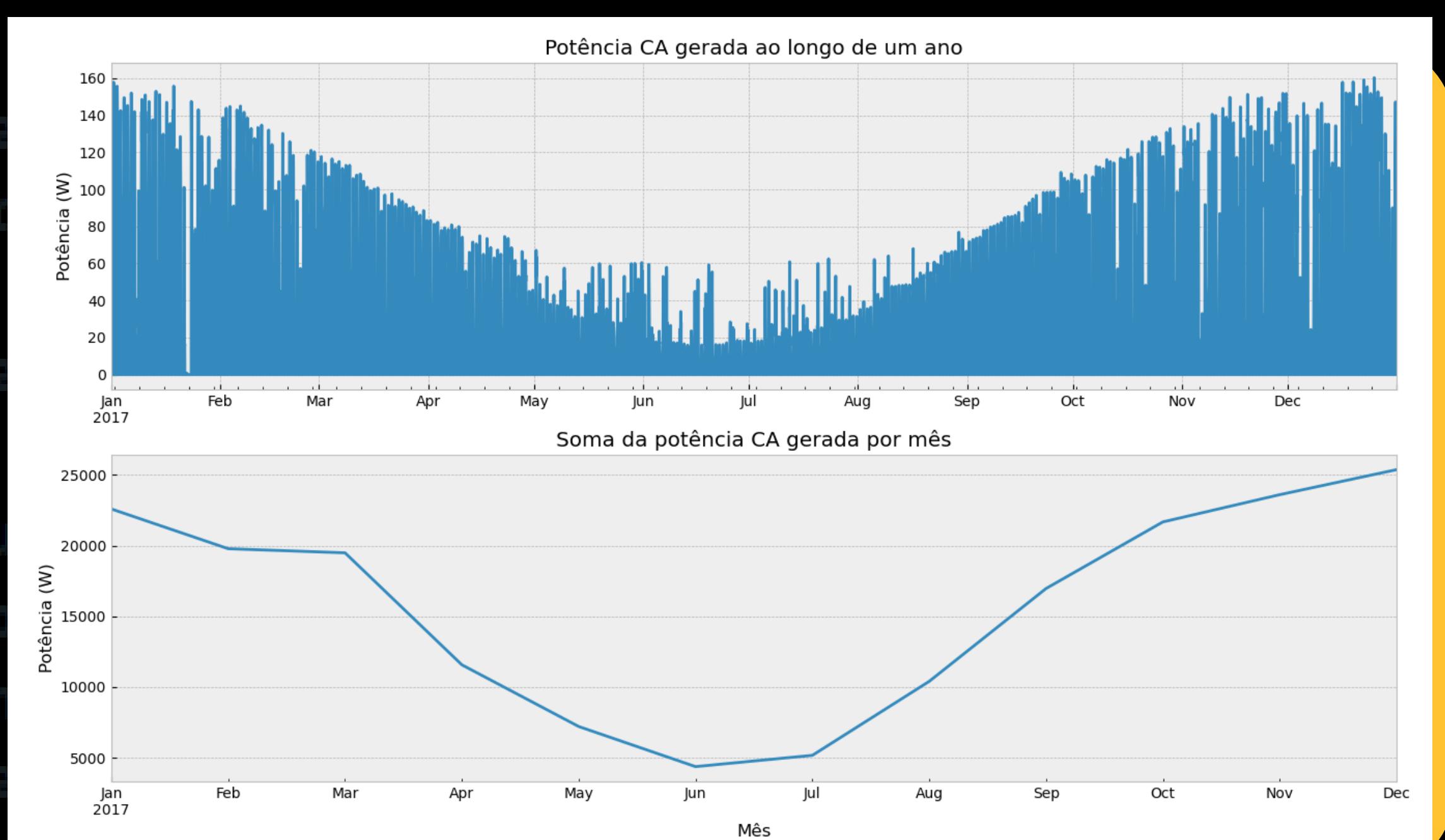
Pode-se comparar esses resultados com os obtidos para o Exemplo 1 desenvolvido anteriormente, onde evidencia-se a maior intermitênciados dados obtidos. O gráfico inferior apresenta a soma da energia gerada em cada mês para o mesmo período.



TERCEIRO PASSO: UTILIZAÇÃO DOS DADOS NO PVLIB

Dessa forma, a potência CA gerada ao longo de um ano (gráfico superior) para a localização da FAET, considerando dados meteorológicos realísticos, apresentará o comportamento abaixo.

Pode-se comparar esses resultados com os obtidos para o Exemplo 1 desenvolvido anteriormente, onde evidencia-se a maior intermitênciados dados obtidos. O gráfico inferior apresenta a soma da energia gerada em cada mês para o mesmo período.



elchain import ModelChain
ation import Location
ystem import PVSystem
perature import TEMPERATURE_MODEL_PARAMETERS

as pd
lib.pyplot as plt

FAET UFMT

ation(latitude=-15.608259564142905,
longitude=-56.06485209329807,
tz='America/Fortaleza',
altitude=190,
name='FAET UFMT')

= pvli
= pvli

CONCLUSÕES

a_mode
_inver

parameters = TEMPERATURE_MODEL_PARAMETERS['sapm']['op

cem(surface_tilt=45,surface_azimuth=180,
module_parameters = module,
inverter_parameters = inverter,
temperature_model_parameters=temperature_params,
modules_per_string=1,
strings_per_inverter=1)

odelChain(system,location)

e_range(start="2019-01-01",end="2019-01-02",
freq="1h",

CONCLUSÕES

O presente material teve por objetivo demonstrar a biblioteca PVLIB como ferramenta para a obtenção de dados de geração fotovoltaica. Além disso, demonstrando como é feito o vínculo com dados fornecidos pelo PVGIS para tornar as estimações ainda mais realísticas.

Por meio de exemplos expostos nesse material e disponibilizados no repositório público desenvolvido, os alunos e pesquisadores interessados no tópico puderam dar os primeiros passos na utilização dessas ferramentas.

Espero que este material tenha sido útil em seu aprendizado. Dúvidas, sugestões ou agradecimentos poderão ser feitas por meio do e-mail cristhiangro@gmail.com.

Abraços, e bons estudos.

Cristhian Groenewald

Engenheiro de Energia

Graduado em Engenharia Elétrica

Mestrado em Engenharia Elétrica

Doutorado em Engenharia Elétrica

Professor Adjunto da UFMT

E-mail: cristhiangro@gmail.com

SOBRE O AUTOR



Cristhian G. da R. de Oliveira

Graduando em Engenharia Elétrica pela Universidade Federal de Mato Grosso, é aluno bolsista FAPEMAT de iniciação científica atuando com projetos de pesquisa nas áreas de capacidade de hospedagem de geração distribuída em redes de distribuição, resiliência operacional e controle de redes de energia elétrica. Possui experiência com docência, tendo recebido o reconhecimento como "Educador do Futuro" pela Câmara Municipal de Goiânia por práticas inovadoras na área de educação.

REFERÊNCIAS

William F. Holmgren, Clifford W. Hansen, and Mark A. Mikofski. “*pvlib python: a python package for modeling solar energy systems.*” Journal of Open Source Software, 3(29), 884, (2018).

<https://doi.org/10.21105/joss.00884>

pvlib. API Reference. Disponível em: <<https://pvlib-python.readthedocs.io/en/stable/reference/index.html>>

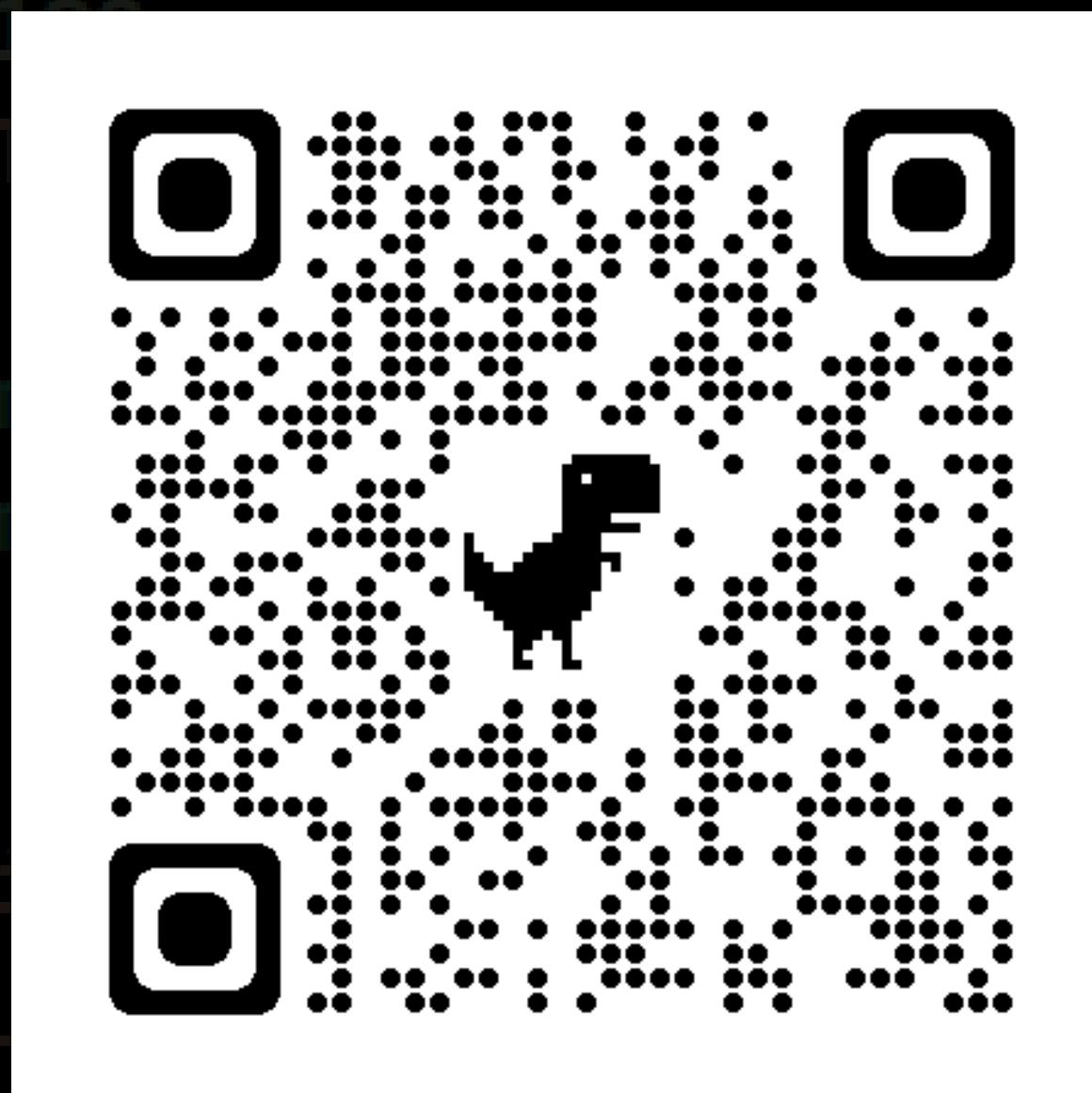
BIRK, Sascha. *Pvlib python Playlist.* Disponível em: <https://www.youtube.com/watch?v=zKzhMQaNjDI&list=PLK7k_QaEmaHsPk_mwzneTE2VTNCpYBiky>

PVGIS. PVGIS Online Tool. Disponível em:

<https://joint-research-centre.ec.europa.eu/pvgis-online-tool_en>

```
parameters = TEMPERATURE_MODEL_PARAMETERS['sapm']['op
PVGIS. PVGIS Online Tool. Disponível em:
<https://joint-research-centre.ec.europa.eu/pvgis-
online-tool\_en>
parameters = module,
inverter_parameters = inverter,
temperature_model_parameters=temperature_params,
modules_per_string=1,
strings_per_inverter=1)
```

Acesso o repositório desse material através do QR Code a baixo



Cristhian G. da R. de Oliveira
cristhiangro@gmail.com