Fundação Escola Técnica Liberato Salzano Vieira da Cunha

MANUAL DE MANUTENÇÃO

Simulador de células fotovoltaicas
e suas curvas características

Anthony Silva Guerreiro
Artur Ritzel

Turma 4411

Novo Hamburgo,
7 de Dezembro de 2023

SUMÁRIO

FUNCIONAMENTO

Utilizando o Octave, são implementadas funções para auxílio nas simulações e cálculos das curvas características das células fotovoltaicas, utilizando o método do Modelo de Diodo Único. O Octave possui pacotes de extensão para uso da comunicação MQTT, o meio de comunicação que será utilizado no programa. Os dados serão enviados e recebidos em uma plataforma gráfica que tenha integração com o celular: Tago.IO, uma plataforma de integração de IoT que possui todos os recursos necessários para o funcionamento do programa. Todos os códigos estão no meu github,

A parte principal do programa resume-se no cálculo e simulação das relações de corrente/potência/tensão, no Octave. Depois de 11 funções de cálculos não funcionais, imprecisas ou problemáticas em algum aspecto, a 12ª e 13ª funções se mostraram extremamente úteis e eficazes. Enquanto a primeira delas calcula os pontos de corrente/potência por tensão com a temperatura fixa, variando a irradiância solar, a segunda calcula esses pontos com a irradiância solar fixa, enquanto a temperatura fixa é variada.

A entrada dessas funções é a mesma, e ambas precisam das mesmas variáveis de entrada: a corrente de curto circuito, a tensão de circuito aberto, a tensão máxima de energia e o coeficiente de temperatura. Todos esses valores são disponibilizados pelo fabricante.

O código das primeira das duas funções citadas pode ser encontrada abaixo (**ou no meu github, com a formatação correta**, acessando o repositório em https://github.com/arturritzel/curva-caracteristica-celula-fotovoltaica):

```
function sim_cel12(I_SC, Vdc, Vmp, TC)

 global V Iplot Pplot
```

Essas variáveis são globais: ao invés de retorno, essas variáveis calculadas serão declaradas no escopo global, para evitar repetições de cálculo.

```
 % variaveis
 q = 1.60217662 * (10^(-19)); % elementary charge
 k = 1.38064852 * (10^(-23));  % Boltzmanns constant
 n = 1.4;      % ideality factor
```

q e k são valores fixos/constantes, enquanto o fator de idealidade é um fator arbitrário que deve ser ajustado: esse valor equilibra perdas em cálculos que se baseiam em cenários ideais, como por exemplo, cálculos que não consideram fatores de potência e, no método de cálculos do modelo de diodo único, perdas no diodo.

```
 T = 298.15;      % temperatura da celula (KELVIN)
 I_r = 200:200:1000    % input de irradiancia

 % V = linspace(0,Vmp/Vdc);    % usando tensão como variável de entrada
```

```
 V = linspace(0, Vmp/Vdc, 35);

 T_0 = 298.15;       % temp de referencia = 25C
 I_r0 = 1000       % irradiancia de referencia
 V_OC = 0.721;     % v ref

 V_g = 1.79*(10^(-19));  % band gap em joules
 [V_m,I_rm] = meshgrid(V,I_r);    % criando a meshgrid
  I_s0 = 1.2799*(10^-8);        % corrente de saturação (equacao encontrada no artigo referenciado)
  I_ph = ((I_SC/I_r0).*I_rm).*(1+ TC*(T-T_0));  % equacao de fotocorrente (encontrada no artigo referenciado)
    I_s  =  I_s0.*(T./T_0).^(3/n).*exp((-(q*V_g)/n*k).*((1./T)-(1/T_0)));    % corrente de saturacao (artigo referenciado)
 I = I_ph - I_s.*exp(((q*V_m)/(n*k*T))-1);   % equacao corrente
 P = I.*V;


 Iplot=I;
 Iplot(Iplot<0)=nan; % evitando valores incondizentes
 Pplot = P;
 Pplot(Pplot<0)=nan;

 figure(1);
 plot(V,Iplot);
 xlabel('Tensão (V)');
 ylabel('Corrente (I)');
 grid on;

 figure(2);
 plot(V,Pplot);
 xlabel('Tensão (V)');
 ylabel('Potência (P)');
 grid on;

endfunction
```

O código da segunda função é muito semelhante, e tem pouquíssimas diferenças em relação à primeira: apenas o necessário para variar a variável de temperatura ao invés da irradiância solar.

```
function sim_cel13(I_SC, Vdc, Vmp, TC)

 global V Pplot2 Iplot2;

 q = 1.60217662 * (10^(-19));
 k = 1.38064852 * (10^(-23));
```

```
n = 1.4;

I_r = 800; % irradiância fixa nessa função

% V = linspace(0,Vmp/Vdc);    % usando tensão como variável de entrada
V = linspace(0, Vmp/Vdc, 35);

T_0 = 298.15;
V_OC = 0.721;
V_g = 1.79*(10^(-19));

% varia a temperatura(T) de 298.15 K para 338.15 K em degraus de 10 K
temperatures = 298.15:10:338.15;

Iplot2 = zeros(length(temperatures), length(V));
Pplot2 = zeros(length(temperatures), length(V));

for i = 1:length(temperatures)
  T = temperatures(i);
  I_s0 = 1.2799*(10^-8);
  I_ph = ((I_SC/I_r).*I_r).*(1 + TC*(T - T_0));
  I_s = I_s0.*(T./T_0).^(3/n).*exp((-(q*V_g)/n*k).*((1./T)-(1/T_0)));
  I = I_ph - I_s.*exp(((q*V)/(n*k*T)) - 1);
  P = I.*V;

  Iplot3 = I;
  Iplot3(Iplot3 < 0) = nan;
  Pplot3 = P;
  Pplot3(Pplot3 < 0) = nan;

  Iplot2(i, :) = Iplot3;
  Pplot2(i, :) = Pplot3;
end

% plotando todas as curvas juntas
figure(3);
plot(V, Iplot2);
xlabel('Tensão (V)');
ylabel('Corrente (I)');
title('Variação da Corrente com a Temperatura');
legend(arrayfun(@(T) ['T = ' num2str(T) ' K'], temperatures, ' ', false));
grid on;

figure(4);
plot(V, Pplot2);
xlabel('Tensão (V)');
ylabel('Potência (P)');
title('Variação da Potência com a Temperatura');
```

```
 legend(arrayfun(@(T) ['T = ' num2str(T) ' K'], temperatures, ' ', false));
 grid on;

end
```

De nada adianta calcular os pontos se não enviamos ele à nossa interface, no tago.io. É para isso que servem as próximas funções. No início do programa, é necessário importar os pacotes e também iniciar as variáveis globais utilizadas nas rotinas de código:

```
pkg load mqtt

% conectando ao broker mqtt
global client;
client = mqttclient('mqtt.tago.io', 'Port', 1883, 'Username', 'Token', 'Password',
'insira-seu-token');

% inscrever-se caso queira receber as proprias mensagens enviadas
%subs = subscribe(client, "answer", "Callback", @recebemessage);

% para receber os pedidos; define função de callback para recebemessage()
subs = subscribe(client, "request", "Callback", @recebemessage);

global entrada1;
entrada1 = -99; % valor usado para verificação posterior
global entrada2;
entrada2 = -99;
global entrada3;
entrada3 = -99;
global entrada4;
entrada4 = -99;

global Iplot;
global Pplot;
global V;
global Iplot2;
global Pplot2;
```

Como definido acima, a função de callback após recebimento da mensagem via MQTT é a função recebemessage(). Ela é responsável por receber a mensagem da interface e interpretá-la, traduzindo a mensagem em uma variável de entrada.

```
function recebemessage(t,v) % recebe mensagem via mqtt
 printf("Topic: %s / Message: %s\n", t, v);
```

```
 global entrada1 entrada2 entrada3 entrada4;

 parts = strsplit(v, ':');
  if numel(parts) == 2
     variav = strtrim(parts{1});
     value = str2double(strtrim(parts{2}));

     % atualiza a variavel de entrada
     switch variav
        case 'entrada1'
           entrada1 = value;
        case 'entrada2'
           entrada2 = value;
        case 'entrada3'
           entrada3 = value;
        case 'entrada4'
           entrada4 = value;
        otherwise
           fprintf('recebido: %s\n', variav);
     end
  end

  checagem();

endfunction
```

Ao receber cada mensagem, o programa checa se já possui todas as variáveis necessárias, utilizando a função checagem() – e caso isso seja verdadeiro, ela já pode enviar os pontos calculados dos gráficos:

```
function checagem % verifica se já possui todas as variáveis necessárias para calcular os pontos

 global entrada1 entrada2 entrada3 entrada4;

  if(entrada1 != -99 && entrada2 != -99 && entrada3 != -99 && entrada4 != -99)

    envia()

    entrada1 = -99;
    entrada2 = -99;
    entrada3 = -99;
    entrada4 = -99;

  end
```

```
endfunction
```

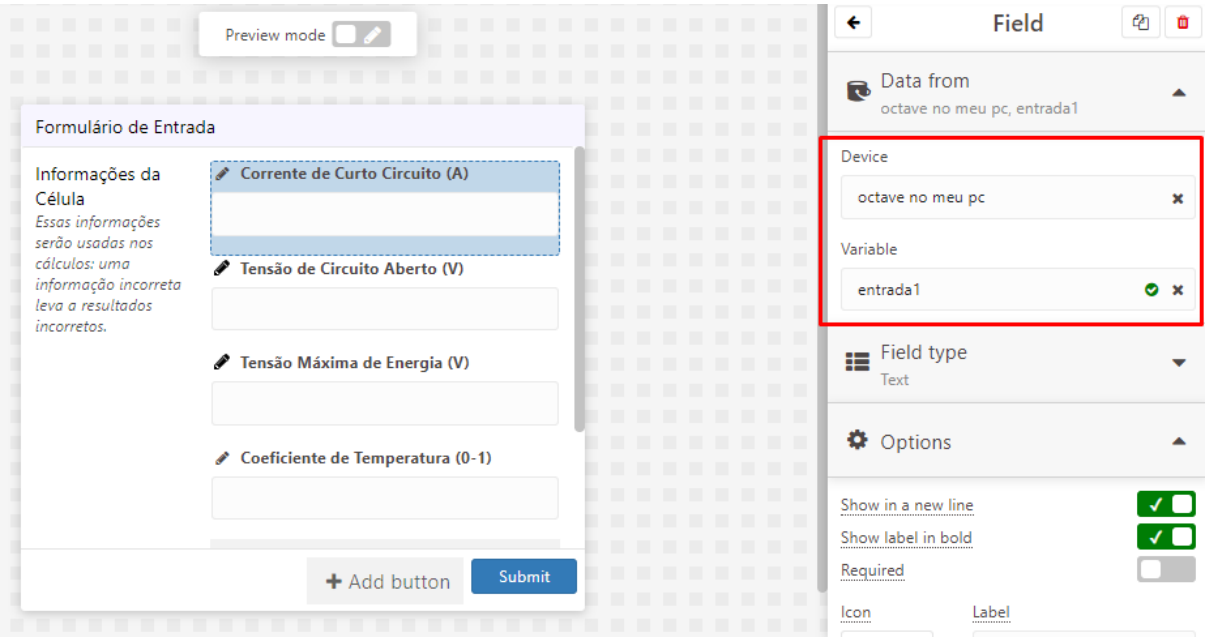
A função envia() utiliza as entradas para calcular todos os pontos de ambas as funções de simulação, e envia todos os pontos de cada curva, de cada gráfico, via MQTT:

```
function envia()
  global client;

  global entrada1 entrada2 entrada3 entrada4;
  global V Pplot Iplot Pplot2 Iplot2

  sim_cel12(entrada1, entrada2, entrada3, entrada4) % calcula os pontos
  sim_cel13(entrada1, entrada2, entrada3, entrada4)

  CV = V;
  CPplot = Pplot;
  CIplot = Iplot;
  CPplot2 = Pplot2;
  CIplot2 = Iplot2;
  #CPplot2 = Pplot;
  #CIplot2 = Iplot;

  disp(Pplot(1,5));
  disp(length(CV));

  for indice = 1:length(CV)
    disp(indice);

      % envia todos os pontos em uma única mensagem que será interpretada no
tago.io
                    message = sprintf('[ { "variable": "payload", "value":
"%i,%i,%i,%i,%i,%i,%i,%i,%i,%i,%i,%i,%i,%i,%i,%i,%i,%i,%i,%i" } ]', CV(indice),
CPplot(1,indice), CPplot(2,indice), CPplot(3,indice), CPplot(4,indice), CPplot(5,indice),
CIplot(1,indice), CIplot(2,indice), CIplot(3,indice), CIplot(4,indice), CIplot(5,indice),
CPplot2(1,indice), CPplot2(2,indice), CPplot2(3,indice), CPplot2(4,indice),
CPplot2(5,indice), CIplot2(1,indice), CIplot2(2,indice), CIplot2(3,indice),
CIplot2(4,indice), CIplot2(5,indice));

    disp(message);

    write(client, "answer", message);

    pause(0.3);
  end

endfunction
```

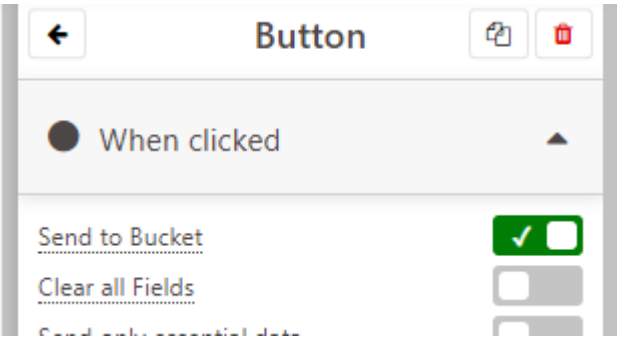
Com as rotinas de funcionamento prontas no octave, precisamos moldar a interface do usuário para que seja possível:

- a) enviar os dados para simulação;
- b) receber e tratar o resultado da simulação;
- c) mostrar ao usuário os valores simulados.

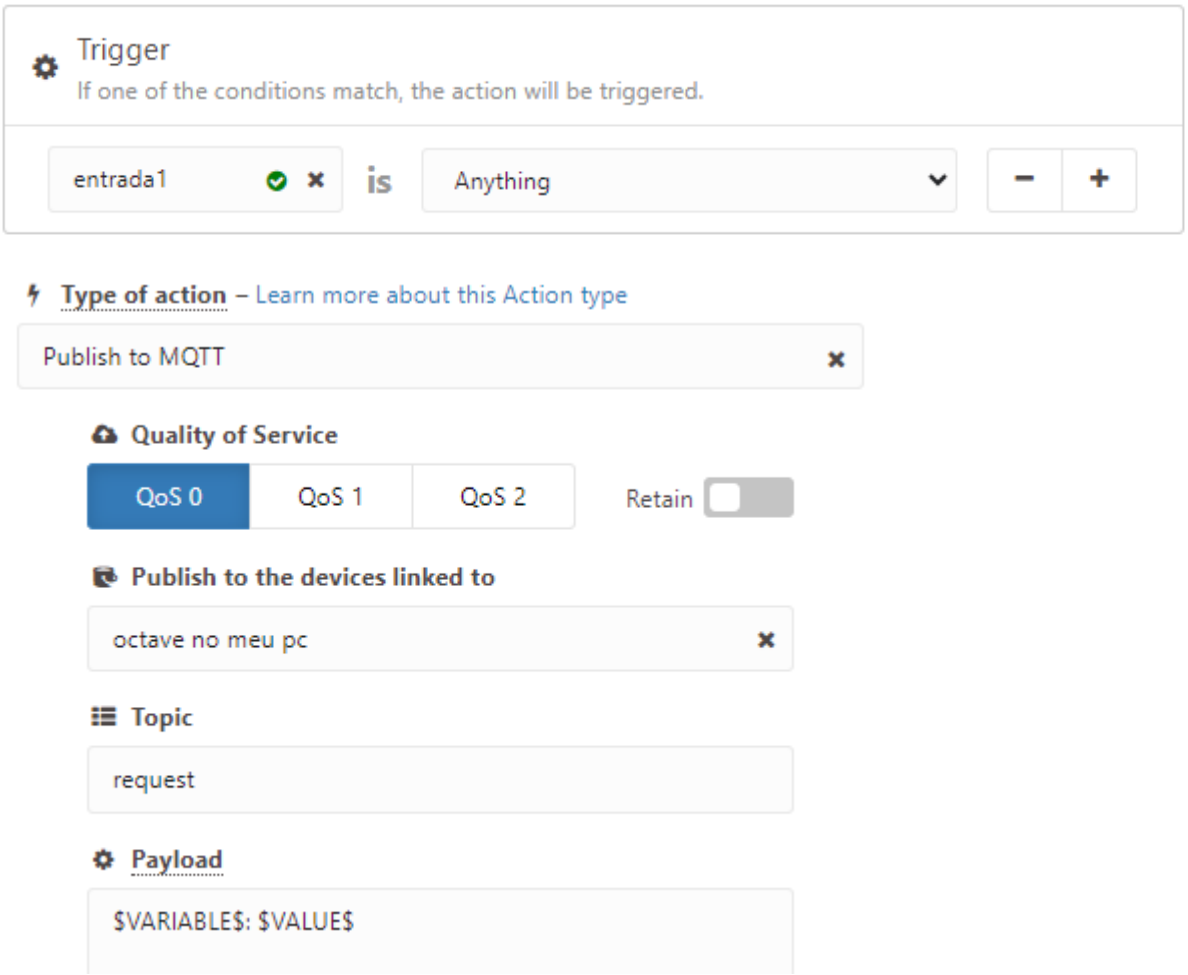
O Tago.io disponibiliza diversos objetos de interface; para a entrada de dados do usuário, utilizamos o Input Form, um formulário de entrada de valores. Cada campo de entrada no formulário corresponde a uma variável de entrada, e é associada a uma variável no sistema do Tago.io, como mostrado na imagem abaixo:



O botão “submit” é configurado para que esses dados sejam gravados na memória do sistema (“Send to Bucket”). Dessa forma, é fácil configurar um evento de envio quando as variáveis forem alteradas:



Com isso, criamos uma ação configurada para que o dado seja enviado toda vez que ele seja alterado: isso é, toda vez que o usuário confirmar o formulário:



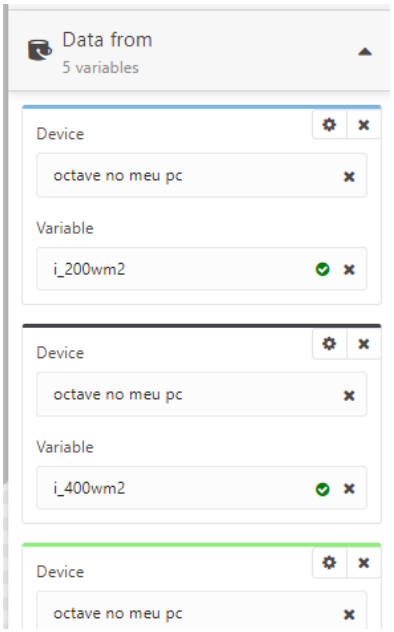
Quando essas variáveis são enviadas ao Octave, as simulações acontecem e, então, as mensagens são recebidas e tratadas em um código simples, mas mecânico:

```
      const mqtt_payload = payload.find((data) => data.variable === "payload" || (data.metadata && data.metadata.mqtt_topic));
      if (mqtt_payload) {
        // Split the content by the separator ,
        const splitted_value = mqtt_payload.value.split(',');
        // splitted_value content will be ['temp', '12', 'hum', '50']
        // index starts from 0

       // Normalize the data to TagoIO format.
       // We use Number function to cast number values, so we can use it on chart widgets, etc.
        const data = [
```

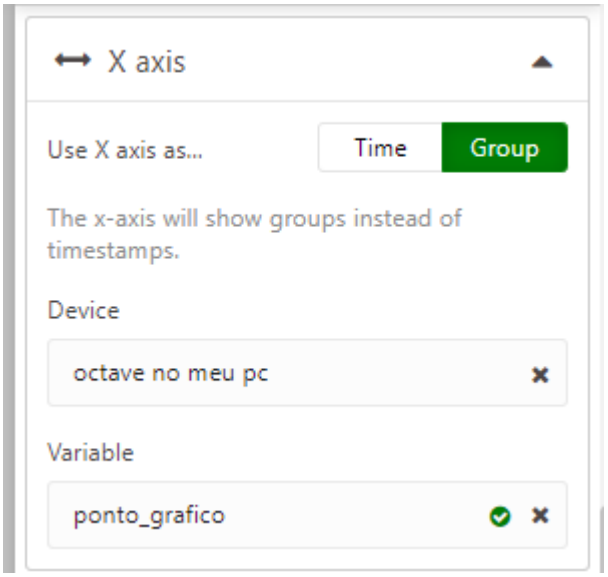
As linhas a seguir separam todos os 21 dados enviados para cada ponto entre: o ponto atual e os 5 pontos de cada um dos 4 gráficos.

```
                        {    variable:    'ponto_grafico',         value:
Number(splitted_value[0])},

          { variable: 'P_200Wm2',  value: Number(splitted_value[1])},
          { variable: 'P_400Wm2',  value: Number(splitted_value[2])},
          { variable: 'P_600Wm2',  value: Number(splitted_value[3])},
          { variable: 'P_800Wm2',  value: Number(splitted_value[4])},
          { variable: 'P_1000Wm2',  value: Number(splitted_value[5])},

          { variable: 'I_200Wm2',  value: Number(splitted_value[6])},
          { variable: 'I_400Wm2',  value: Number(splitted_value[7])},
          { variable: 'I_600Wm2',  value: Number(splitted_value[8])},
          { variable: 'I_800Wm2',  value: Number(splitted_value[9])},
          { variable: 'I_1000Wm2',  value: Number(splitted_value[10])},

          { variable: 'P_25C',  value: Number(splitted_value[11])},
          { variable: 'P_35C',  value: Number(splitted_value[12])},
          { variable: 'P_45C',  value: Number(splitted_value[13])},
          { variable: 'P_55C',  value: Number(splitted_value[14])},
          { variable: 'P_65C',  value: Number(splitted_value[15])},

          { variable: 'I_25C',  value: Number(splitted_value[16])},
          { variable: 'I_35C',  value: Number(splitted_value[17])},
          { variable: 'I_45C',  value: Number(splitted_value[18])},
          { variable: 'I_55C',  value: Number(splitted_value[19])},
          { variable: 'I_65C',  value: Number(splitted_value[20])},
        ];

        const group = String(new Date().getTime());
        payload = payload.concat(data).map(x => ({ ...x, group }));
      }
```

Com os dados recebidos, tratados e separados, basta mostrar eles ao usuário de uma forma amigável. Na interface do usuário, selecionamos o objeto de gráfico no formato de linha, e selecionamos quais variáveis ele deseja mostrar:



Configuramos o eixo X, para que ele mostre o ponto de tensão, ao invés do horário em que os dados foram recebidos:



Com o gráfico configurado, basta entrar os valores no formulário, e o programa automaticamente recebe, trata e nos mostra os valores simulados:

Gráfico P x V
placa em 25°C
variando Irradiância Solar