**UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO**
**CENTRO TECNOLÓGICO**
**COORDENADORIA DE ENGENHARIA ELÉTRICA**

**HIGOR DAVID OLIVEIRA**

# MÉTODOS NUMÉRICOS APLICADOS À ELETROMAGNETISMO: MÉTODO DAS DIFERENÇAS FINITAS

**Vitória, ES**
**Setembro de 2019**

**HIGOR DAVID OLIVEIRA**

# MÉTODOS NUMÉRICOS APLICADOS À ELETROMAGNETISMO: MÉTODO DAS DIFERENÇAS FINITAS

Trabalho apresentado para a obtenção de nota parcial na disciplina de Eletromagnetismo 2, oferecida pelo Universidade Federal do Espírito Santo para a graduação em Engenharia Elétrica.
Professor: Carlos Eduardo Schmidt Castellani.
Turma: 01 - 2019/2

## Sumário

## 1. Introdução teórica

Um método numérico refere-se ao algoritmo que, por meio de um número finito de operações numéricas, é capaz de alcançar a solução de sistemas modelados computacionalmente. Esses sistemas computacionais podem ser diversos, como exemplo, podem ser soluções de problemas reais, de otimização ou de simulações. Assim também, existem vários métodos diferentes e, dentre os mais conhecidos, há o método das diferenças finitas, foco deste trabalho.

O método das diferenças finitas é uma técnica numérica usada para resolver sistemas descritos analiticamente, no qual, por meio da realimentação e da convergência de um sistema descrito por um conjunto de equações, por várias vezes, calcula-se a solução do conjunto e realimenta-o com a solução encontrada. Devido às condições de contorno do sistema e da convergência do mesmo, a cada iteração do algoritmo a solução encontrada para o sistema se torna mais próxima da solução análitica. Uma vez que a solução encontrada se encontra dentro de uma margem de erro, considera-se que a solução encontrada é suficientemente próxima da real.

O algoritmo funciona com base em três características: o equacionamento feito a partir de uma equação diferencial; a determinação de um domínio do sistema; e o estabelecimento das equações de contorno e as condições iniciais.

O método é útil e muito usado na solução de equações diferenciais parciais. Entre os usos mais comuns, é possível destacar o uso para solução das equações de Poisson e as equações de Laplace.

$$\nabla^2 V = -\frac{\rho s}{\varepsilon}; \tag{1.0}$$

Onde, **V** é o vetor potencial elétrico,

$\rho s$ é a densidade volumétrica de cargas,

$\varepsilon$ é a constante de permissividade do meio.

É possível chegar nas equações acima tendo como base a fórmula de campo elétrico (1.1), a Primeira Lei de Maxwell (1.2) e a Equação de Laplace (1.3),

$$E = -\nabla V; \quad \& \quad \nabla E = -\frac{\rho s}{\varepsilon}; \tag{1.1 e 1.2}$$

$$\nabla^2 \psi = \frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2} + \frac{\partial^2 \psi}{\partial z^2}; \tag{1.3}$$

A equação (1.0) trata-se de uma equação diferencial de ordem 2. Computacionalmente, por meio das equações de Laplace (1.3), tais equações diferenciais podem ser discretizadas junto com o seu domínio, para então, serem simplificadas e descritas por como funções de variáveis discretas, tornando o cálculo da derivada parcial como simples média entre pontos próximos. Quanto maior o número de pontos que se descreve a variável discreta, mais próxima se torna a sua simplificação da derivada real.

$$\frac{\partial\,\psi_{(xo,\,yo,\,zo)}}{\partial x} = \frac{\psi_{(xo+\Delta x/2,\,yo,\,zo)} - (xo-\Delta x/2,\,yo,\,zo)}{\Delta x}\;; \qquad (1.4)$$

$$= \psi' = \frac{\psi_{[i+1,\,j,\,k]} - \psi_{[i-1,\,j,\,k]}}{\Delta i};$$

$$\frac{\partial^2\,\psi_{(xo,\,yo,\,zo)}}{\partial x^2} = \frac{\partial\,\psi'_{(xo,\,yo,\,zo)}}{\partial x} = \frac{\psi'_{(xo+\Delta x/2,\,yo,\,zo)} - (xo-\Delta x/2,\,yo,\,zo)}{\Delta x}; \qquad (1.5)$$

$$= \frac{\psi_{(xo+\Delta x,\,yo,\,zo)} - 2\cdot\psi_{(xo,\,yo,\,zo)} + \psi_{(xo-\Delta x,\,yo,\,zo)}}{(\Delta x)\cdot(\Delta x)};$$

$$= \frac{\psi_{(i+1,\,j,\,k)} - 2\cdot\psi_{(i,\,j,\,k)} + \psi_{(i-1,\,j,\,k)}}{(\Delta i)^2}; \qquad (1.6)$$

O mesmo vale para **y** e **z**.

Tendo como foco deste trabalho o cálculo do potencial sobre um campo potencial em **x** e **y**, descrito com as condições de contorno da equação (1.1), substituindo as equações (1.6) e suas equivalentes em (1.0), podemos chegar nas equações que serão usadas,

$$\frac{V_{[i+1,\,j\,]} + V_{[i,\,j+1]} + V_{[i-1,\,j]} + V_{\,[i\,,\,j-1]} + 4\cdot V_{[i,\,j]]}}{\Delta i^2 + \Delta j^2} = -\frac{\rho s}{\varepsilon}; \qquad (1.7)$$

Considerando $\rho$s = 0, temos por fim,

$$V_{[i,\,j]} = V_{[i+1,\,j]}\cdot\frac{1}{4} + V_{[i,\,j+1]}\cdot\frac{1}{4} + V_{[i-1,\,j]}\cdot\frac{1}{4} + V_{[i,\,j-1]}\cdot\frac{1}{4}; \qquad (1.8)$$

## 2. Objetivos

O objetivo consiste no estudo do método numérico das diferenças finitas feito por meio da aplicação do algoritmo duas vezes no cálculo de um campo potencial gerado por duas placas de potencial 15V, no primeiro caso com as placas na borda do campo, e, no segundo caso, distanciadas em d na parte central do campo.

## 3. Implementação

O algoritmo foi implementado em linguagem do *matlab*, compilado e executado nele. O código se encontra como **apêndices A**, para as placas de potencial elétrico nas bordas do campo, e **B,** para as placas na parte interna ao campo. O princípio de funcionamento do algoritmo é feito usando a equação 1.8, explicada anteriormente. O código pode ser dividido em três partes: a parte de declaração das variáveis e criação do campo potencial; a parte que contém os cálculos do algoritmo e as ***nIteracoes*** iterações do código; e a parte que contém a gravação de um arquivo de vídeo.

A declaração das variáveis é a parte mais importante para a implementação das diferenças finitas. Estabeleceu-se que o erro seria absoluto, ***erro***, sendo a soma dos módulos das variações dos valores de cada ponto da matriz, ***campo***, com relação ao valor anterior, ***campoAnterior***. O algoritmo termina quando o valor do erro se torna menor que o valor do erro mínimo, ***erroMin***.

***nPontos*** é a variável que dirá qual o tamanho da matriz nPontos x nPontos do campo potencial. Com relação ao campo potencial, ***campo***, esta é uma matriz onde cada ponto contém dois valores: o valor do potencial elétrico naquele ponto, ***valores***; e uma flag (verdadeiro ou falso ou, tambem, *true* ou *false*) que indica se aquele ponto é parte de uma placa de potencial ou não, ***ehPlaca***. Se o ponto em questão não for placa, o seu valor mudará a cada iteração do código, de acordo com o cálculo do algoritmo, se não, o valor daquele ponto fica inalterado.

As placas de potencial elétrico são estruturadas como *structs*, ou seja, estruturas com mais elementos, entre eles: tamanho, descrito em porcentagem com relação ao tamanho do campo, ***.tam***; a posição, descrito em valores discretos, indicando em qual coluna a placa ficará, .***pos***, sempre começando do meio e indo para ***tam/2*nPontos***

para cima e para baixo; e os valores de potencial, em volts, ***valor***. A posição das placas e a distância entre elas é controlada pela variável ***distanciaEntrePlacas***, que indicará em porcentagem a distância entre as placas no campo. Antes de passar para a parte de cálculo do algoritmo, o programa percorre todo o campo, atribuindo os potenciais e as flags das placas de potencial. Após isso, acontece o cálculo do algoritmo.

A parte que contém os cálculos do algoritmo é composto basicamente de 2 ***for*** e um ***while***. Os dois ***for*** percorrem todo o campo, percorrendo toda coluna ***j*** e linha ***i*** de 1 à ***nPontos*** - 1, ou seja, passando por todos os pontos internos da matriz. A cada ponto que se passa com ***i,j*** é calculado o valor de potencial elétrico dela de acordo com a Fórmula 1.8, com exceção dos pontos onde são placas, que possuem a flag ( index i,j,2) como true.

A cada iteração do ***while*** é uma iteração do algoritmo das diferenças finitas, e, assim, toda iteração calcula-se o novo ***erro***. A condição de parada do ***while*** é quando o erro se encontra menor que o erro mínimo admitido, ***erroMin***.

A animação é toda feita na parte do while. O ***frame*** é a imagem do gráfico mostrada na tela. A cada vez que acontece um iteração, atualiza-se o campo potencial mostrado e então move-se a angulação da câmera, ***view1*** e ***view2***, um pouco. Ao mesmo tempo, é feito um cálculo do tempo de cada frame mostrado para a animação final, ***tempoFrame***; Cada frame mostrado é armazenado em ***frames***.

Já a gravação do vídeo, é feita quando a animação termina. A animação é ajustada para que se repita o mesmo frame da imagem diretamente proporcional ao ***tempoFrame*** daquele frame. Assim, os frames iniciais são repetidos mais vezes no vídeo, permanecendo mais tempo na tela, e os frames finais passam mais rápido.

## 4. Resultados e Discussões

Como resultado, temos que inicialmente o campo potencial é composto basicamente pelos potenciais da placa enquanto todo o resto do campo se encontra com potencial zero.

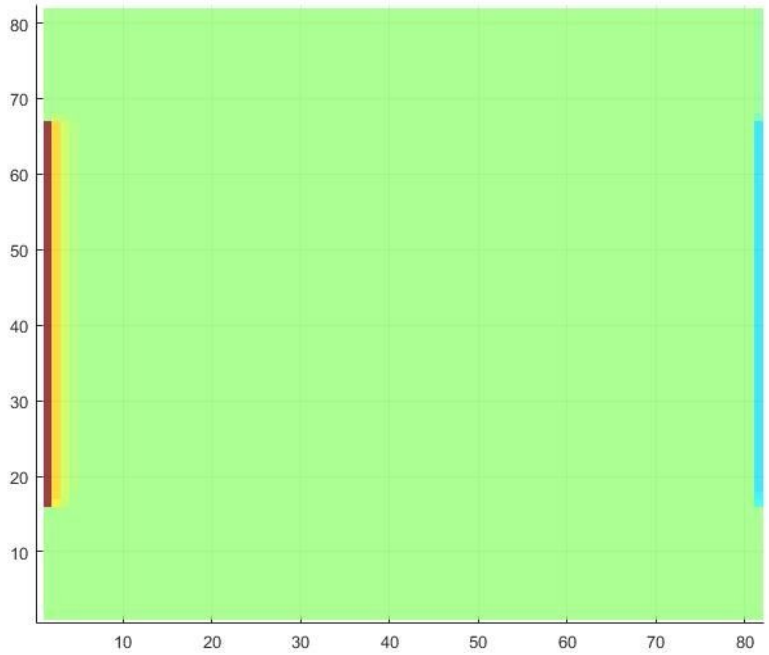


**Figura 1** - Campo do potencial elétrico após a primeira iteração do código, placa nas bordas. **Fonte:** autoria do grupo, feita com auxílio do *Matlab*.

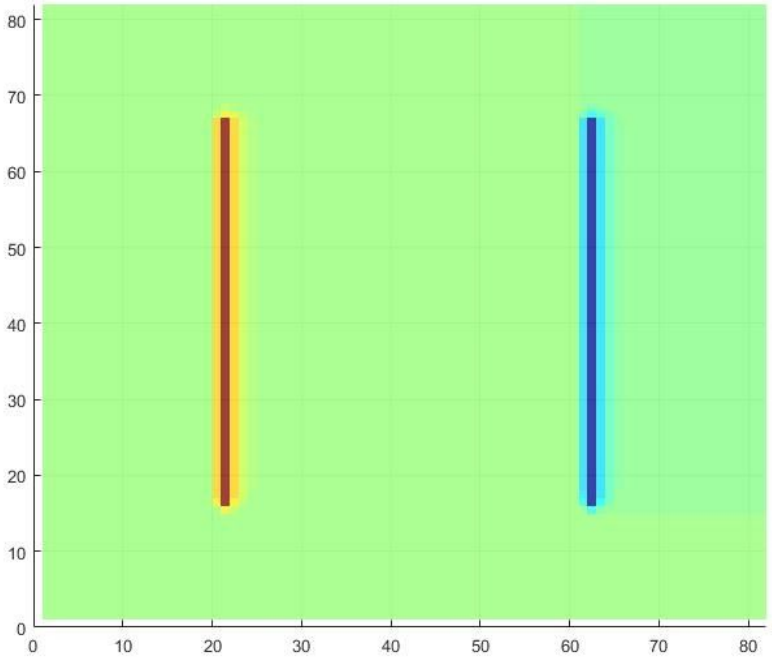


**Figura 2** - Campo do potencial elétrico após a primeira iteração do código, placas na parte interna. **Fonte:** autoria do grupo, feita com auxílio do *Matlab*.

A medida que o algoritmo se avança, as linhas de equipotenciais vão “se espalhando” pelo campo, a partir das placas.

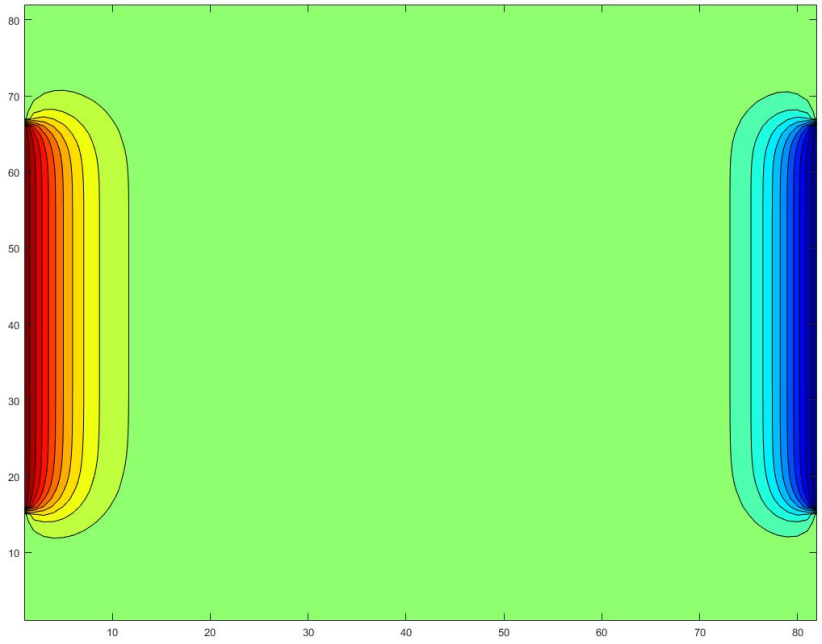


**Figura 3** - Campo do potencial elétrico após aproximadamente 40 iterações do algoritmo, placa nas bordas. **Fonte:** autoria do grupo, feita com auxílio do *Matlab*.

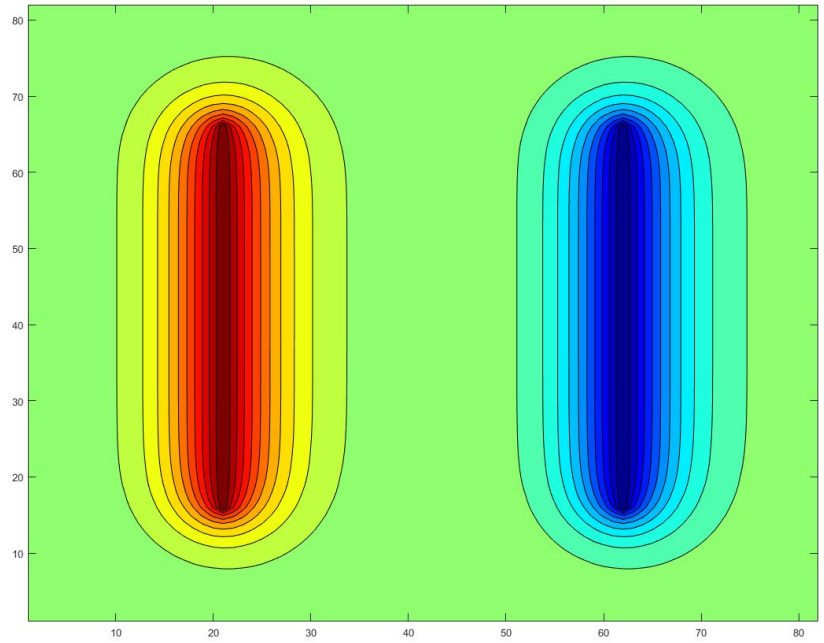


**Figura 3** - Campo do potencial elétrico após aproximadamente 40 iterações do algoritmo, placa na parte interna ao campo. **Fonte:** autoria do grupo, feita com auxílio do *Matlab*.

A cada iteração, observa-se que o potencial de cada ponto do campo varia cada vez menos, até que a variação se torna imperceptível, próxima de zero. Assim, quando o algoritmo chega ao fim, temos os potenciais da placa distribuídos sobre o campo de

forma uniforme, com maiores potenciais próximos à placa de maior potencial e menor potencial, à placa de menor potencial, como pode ser visto nas figuras abaixo;

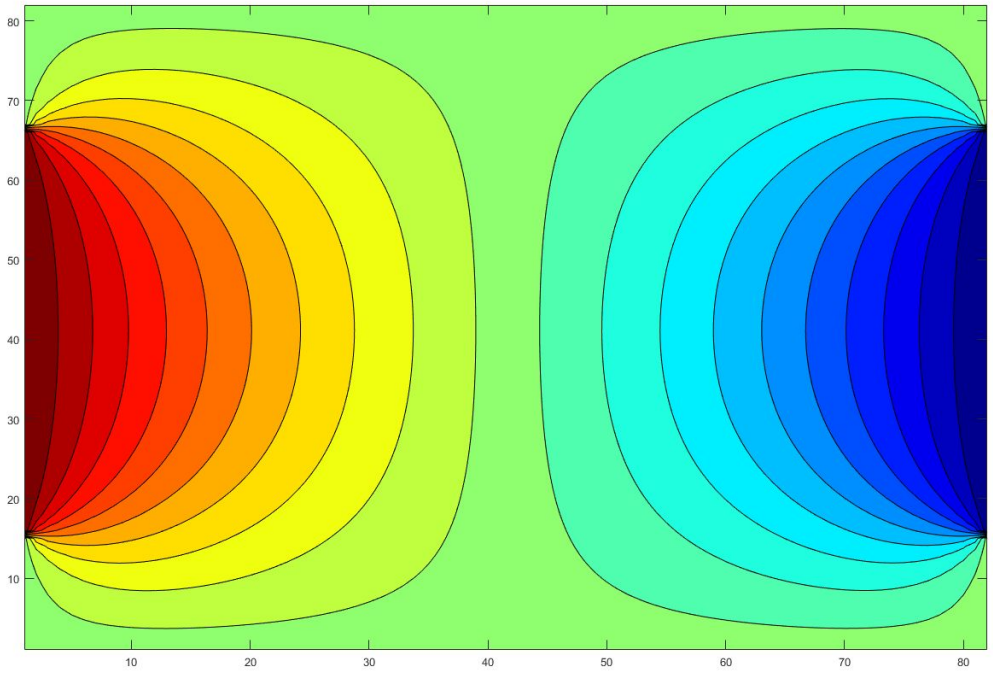


**Figura 5** - Campo do potencial elétrico após aproximadamente 900 iterações do algoritmo, quando o ***erro*** é menor que 0.5, placa nas bordas. **Fonte:** autoria do grupo, feita com auxílio do *Matlab*.

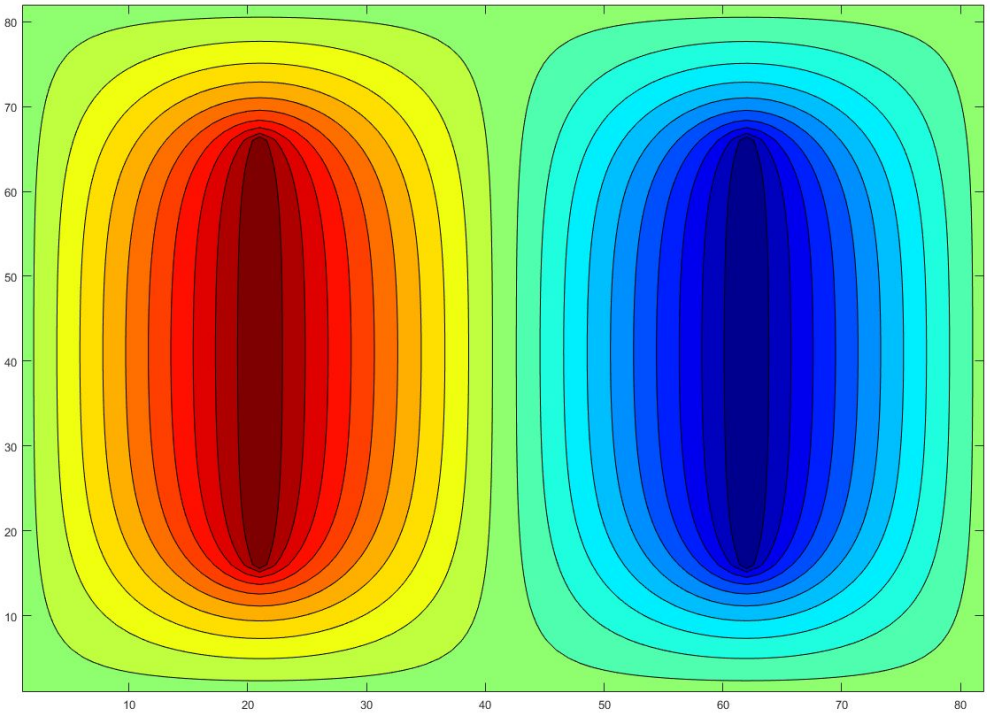


**Figura 6** - Campo do potencial elétrico após aproximadamente 900 iterações do algoritmo, quando o ***erro*** é menor que 0.1, placa na parte interna ao campo. **Fonte:** autoria do grupo, feita com auxílio do *Matlab*.

Como a variação dos valores de potencial elétrico nas iterações finais é muito pequeno, consideramos que chegamos na situação de equilíbrio do sistema. E, como

foi possível obter o resultado de forma numérica, podemos concluir que, graças ao método numérico das diferenças finitas, é possível prever o resultado de um experimento sem necessariamente realizá-lo.

## 5. Conclusão

Com base nos resultados obtidos, mostrados nas Figuras **5** e **6**, podemos concluir que o algoritmo numérico das diferenças finitas foi implementado com sucesso. Sendo possível, com a implementação, prever o campo potencial elétrico resultante sobre uma área dado as condições iniciais e as condições de contorno.

Tendo em mente que o objetivo deste trabalho era a implementação do algoritmo numérico das diferenças finitas para cálculo dos potenciais elétricos sobre um campo dado duas placas de potencial elétrico nas bordas, no primeiro caso, e duas placas na parte interna ao campo no segundo caso, podemos concluir que o objetivo do trabalho foi alcançado.

## Referências Bibliográficas

SADIKU, Matthew, N. **Fundamentos de Eletromagnetismo**. 5ed. 2012 Grupo BookMan. Disponível em acervo UFES, biblioteca central.

LOBÃO, Prof. Diomar Cesar. **Introdução a métodos numéricos.** 2017. UFF - Volta redonda. Disponível em: <http://www.professores.uff.br/diomarcesarlobao/wp-content/uploads/sites/85/2017/09/note6.pdf>. Acesso em: 01 de set 2019;

ARAUJO, Eduardo. **Métodos numéricos para simulação de engenharia.** Blog ESSS, 2017. Disponível em: <https://www.esss.co/blog/metodos-numericos-para-simulacao-na-engenharia/>. Acesso em: 01 de set 2019;

CODECOGS. **Editor de equações LaTeX online.** Versão 3.2. Disponível em: <https://www.codecogs.com/latex/eqneditor.php?lang=pt-br>. Acesso em: 03 de set 2019;

ALMEIDA, Pedro S. **Eletrônica Analógica:** *revisão de cálculo vetorial.* 2015, UFJF. Disponível em: <http://www.ufjf.br/pedro_almeida/files/2015/04/CEL065-%E2%80%93-000-Revis%C3%A3o-C%C3%A1lculo-Vetorial.pdf>. Acesso em: 01 de set 2019;

MEDEIROS, Mariane Affonso; de PAULA, Samuel; DIAS, Vinicius; MOREIRA, William. **Potencial Elétrico.** 2014. Disponível em: <http://fisica3potencialeletricoutfpr.blogspot.com/>. Acesso em: 01 de set 2019.

## Apêndice A - Código para placas de potencial elétrico nas bordas

```
%% Trabalho de mag II - Metodo interativo
% Autor: Higor David Oliveira
% Disciplina: Eletromagnetismo II - 2019/2
% Prof. Carlos Eduardo Castellani

%% Codificação: win1532
clear;
close all;

%% Perguntas
% - Deseja ver a animação? (S/N)
mostrarAnimacao = 'S';
% - Usar contour para plot? (S/N) Se não, será usado surface
usarContour = 'S';
% - Deseja gravar o vídeo? (S/N)
gravarVideo = 'S'; % para gravar o vídeo é necessário rodar a animação
%

%% definição das variáveis do programa
erroMin = 0.7; % erro mínimo antes da condição de parada
erro = erroMin + 1; % valor inicial aleatorio do erro
nPontos = 82; % numero de pontos por linhas/coluna
nIteracoes = 0; % registrador de iterações do código
distanciaEntrePlacas = 100; % distância em porcentagem entre as placas

% definição das placas
placa(1).tam = 60; % tamanho das placas em porcentagem
placa(1).pos = ceil(nPontos/2 - ceil(distanciaEntrePlacas)/2*nPontos/100 ) + 1; % posicao em qual pixel, no caso a
primeira coluna
placa(1).valores = 15; % valor de potencial da placa
placa(2).tam = 60; % tamanho da placa em porcentagem
placa(2).pos = ceil(nPontos/2 + ceil(distanciaEntrePlacas)/2*nPontos/100 ); % ultima coluna
placa(2).valores = -15;

curvas = placa(1).valores:placa(2).valores;
% caso queira adicionar novas placas, basta seguir o padrão aimca

% definição do campo e seus auxiliares
campoAnterior = zeros(nPontos, nPontos); %matriz com valores anteriores
campo = zeros(nPontos, nPontos, 2); %matriz dos campos
% as duas primeiras dimensões são para os valores e a terceira é uma flag,
% que indica se aquele ponto é uma placa ou não.

%% mesclagem do campo com as placas
for nPlaca = placa
  campo((nPontos/2 - ceil( nPlaca.tam/2/100*nPontos) ):(nPontos/2 + ceil( nPlaca.tam/2/100*nPontos) ),nPlaca.pos,
1) = nPlaca.valores; %coloca valor
  campo((nPontos/2 - ceil( nPlaca.tam/2/100*nPontos) ):(nPontos/2 + ceil( nPlaca.tam/2/100*nPontos) ),nPlaca.pos,
2) = true; %indica que é placa
end

%% implementação do metodo interativo
while erro > erroMin
```

```
   %% algoritmo de diferenças finitas
   campoAnterior = campo(:,:,1);
   for i = 2:nPontos - 1
     for j = 2:nPontos - 1
        if campo(j,i,2) == 0 %verifica se o ponto não é placa
           campo(j,i,1) = campo(j-1,i,1)/4 + campo(j,i-1,1)/4 + campo(j+1,i,1)/4 + campo(j,i+1,1)/4;
        end
     end
   end
   erro = sum ( sum ( abs (campo(:,:,1) - campoAnterior) ) ); % calcula o delta dos valores da matriz
   nIteracoes = nIteracoes + 1;
   %

   %% animação
   if mostrarAnimacao == 'S'
      if usarContour == 'S'
         contourf(campo(:,:,1), 20);
         if nIteracoes == 1
           h = gca;
           h.NextPlot = 'replacechildren';
           colormap Jet;
         end
         pause(0.01);
      else
         surf(campo(:,:,1), 'FaceAlpha', 0.75,'EdgeColor', 'none');
         colormap Jet;
         h = gca;
         % configuracoes de primeira vez
         if nIteracoes == 1
            h.XLim = [0 nPontos];
            h.YLim = [0 nPontos];
            h.ZLim = [-15 15];
            h.NextPlot = 'replacechildren'; %para nao resetar as configurações
         end
         view1 = -40 + nIteracoes/1000*80;
         view2 = 20 + nIteracoes/1000*25;
         setview = [view1 view2];
         h.View = setview;
      end

      % pega o tempo de cada frame
      tempoFrame(nIteracoes) = 1*3/(10 + nIteracoes*2);
      frames(nIteracoes) = getframe(gcf);
   end
end
%

%% plot final
if mostrarAnimacao == 'N'
   contourf(campo(:,:,1), 20);
   colormap Jet;
end

%% gravação do vídeo
```

```
if mostrarAnimacao == 'S' && gravarVideo == 'S'
    video = VideoWriter('magContourPlacasExternas');
    frameRate = ceil(1/tempoFrame(nIteracoes)/4);
    video.FrameRate = frameRate;
    open(video);
    for i = 1:length(frames)
       k = tempoFrame(i);
       while k >= 1/frameRate
          writeVideo(video,frames(i));
          k = k - 1/frameRate;
       end
       writeVideo(video,frames(i));
    end
    close(video);
end
%
```

## Apêndice B - Código para placas de potencial elétrico na parte interna ao campo

```
%% Trabalho de mag II - Metodo interativo
% Autor: Higor David Oliveira
% Disciplina: Eletromagnetismo II - 2019/2
% Prof. Carlos Eduardo Castellani

%% Codificação: win1532
clear;
close all;

%% Perguntas
% - Deseja ver a animação? (S/N)
mostrarAnimacao = 'S';
% - Usar contour para plot? (S/N) Se não, será usado surface
usarContour = 'S';
% - Deseja gravar o vídeo? (S/N)
gravarVideo = 'S'; % para gravar o vídeo é necessário rodar a animação
%

%% definição das variáveis do programa
erroMin = 0.1; % erro mínimo antes da condição de parada
erro = erroMin + 1; % valor inicial aleatorio do erro
nPontos = 82; % numero de pontos por linhas/coluna
nIteracoes = 0; % registrador de iterações do código
distanciaEntrePlacas = 50;

% definição das placas
placa(1).tam = 60; % tamanho das placas em porcentagem
placa(1).pos = ceil(nPontos/2 - ceil(distanciaEntrePlacas)/2*nPontos/100 ) + 1; % posicao em qual pixel, no caso a
primeira coluna
placa(1).valores = 15; % valor de potencial da placa
placa(2).tam = 60; % tamanho da placa em porcentagem
placa(2).pos = ceil(nPontos/2 + ceil(distanciaEntrePlacas)/2*nPontos/100 ); % ultima coluna
placa(2).valores = -15;

curvas = placa(1).valores:placa(2).valores;
% caso queira adicionar novas placas, basta seguir o padrão aimca

% definição do campo e seus auxiliares
campoAnterior = zeros(nPontos, nPontos); %matriz com valores anteriores
campo = zeros(nPontos, nPontos, 2); %matriz dos campos
% as duas primeiras dimensões são para os valores e a terceira é uma flag,
% que indica se aquele ponto é uma placa ou não.

%% mesclagem do campo com as placas
for nPlaca = placa
  campo((nPontos/2 - ceil( nPlaca.tam/2/100*nPontos) ):(nPontos/2 + ceil( nPlaca.tam/2/100*nPontos) ),nPlaca.pos,
1) = nPlaca.valores; %coloca valor
  campo((nPontos/2 - ceil( nPlaca.tam/2/100*nPontos) ):(nPontos/2 + ceil( nPlaca.tam/2/100*nPontos) ),nPlaca.pos,
2) = true; %indica que é placa
end

%% implementação do metodo interativo
while erro > erroMin
```

```
   %% algoritmo de diferenças finitas
   campoAnterior = campo(:,:,1);
   for i = 2:nPontos - 1
     for j = 2:nPontos - 1
        if campo(j,i,2) == 0 %verifica se o ponto não é placa
           campo(j,i,1) = campo(j-1,i,1)/4 + campo(j,i-1,1)/4 + campo(j+1,i,1)/4 + campo(j,i+1,1)/4;
        end
     end
   end
   erro = sum ( sum ( abs (campo(:,:,1) - campoAnterior) ) ); % calcula o delta dos valores da matriz
   nIteracoes = nIteracoes + 1;
   %

   %% animação
   if mostrarAnimacao == 'S'
      if usarContour == 'S'
         contourf(campo(:,:,1), 20);
         if nIteracoes == 1
           h = gca;
           h.NextPlot = 'replacechildren';
           colormap Jet;
         end
         pause(0.01);
      else
         surf(campo(:,:,1), 'FaceAlpha', 0.75,'EdgeColor', 'none');
         colormap Jet;
         h = gca;
         % configuracoes de primeira vez
         if nIteracoes == 1
            h.XLim = [0 nPontos];
            h.YLim = [0 nPontos];
            h.ZLim = [-15 15];
            h.NextPlot = 'replacechildren'; %para nao resetar as configurações
         end
         view1 = -40 + nIteracoes/1000*80;
         view2 = 20 + nIteracoes/1000*25;
         setview = [view1 view2];
         h.View = setview;
      end

      % pega o tempo de cada frame
      tempoFrame(nIteracoes) = 1*3/(10 + nIteracoes*2);
      frames(nIteracoes) = getframe(gcf);
   end
end
%

%% plot final
if mostrarAnimacao == 'N'
   contourf(campo(:,:,1), 20);
   colormap Jet;
end

%% gravação do vídeo
```

```
if mostrarAnimacao == 'S' && gravarVideo == 'S'
    video = VideoWriter('magContourPlacasInternas');
    frameRate = ceil(1/tempoFrame(nIteracoes)/4);
    video.FrameRate = frameRate;
    open(video);
    for i = 1:length(frames)
      k = tempoFrame(i);
      while k >= 1/frameRate
         writeVideo(video,frames(i));
         k = k - 1/frameRate;
      end
      writeVideo(video,frames(i));
    end
    close(video);
end
%
```