Universidade Federal Fluminense – UFF
Escola de Engenharia – TCE
Curso de Engenharia de Telecomunicações – TGT
Programa de Educação Tutorial – PET

Projetos PET-Tele

Desenvolvimento de código Octave para busca de máximos em sondagem de canal rádio móvel

(Versão: A2013M06D20)

Autores: Carina Ribeiro Barbio Corrêa

Tutor: Alexandre Santos de la Vega

Niterói – RJ
Junho / 2013

1 Introdução

O presente trabalho foi desenvolvido com o intuito de colaborar com o trabalho de tese de doutoramento do professor Carlos Eduardo Salles (UFF/TCE/TET), realizado na Pontifícia Universidade Católica do Rio de Janeiro (PUC-RJ).

Nas sondas utilizadas para a caracterização dos ambientes nos quais se propagam os sinais eletromagnéticos podem ser detectados diversos sinais que chegam ao receptor por diferentes trajetos, cada qual com seu atraso, amplitude e fase.

O primeiro problema é: i) separar os raios reais detectados pela sonda dos sinais espúrios e ruídos que podem mascarar o resultado, ii) calcular o seu valor complexo e iii) calcular o instante de chegada de cada um deles.

A Figura 1 apresenta uma imagem do resultado de uma medição, mostrando três raios recebidos que devem ser separados dos sinais espúrios através do estabelecimento de um limiar.

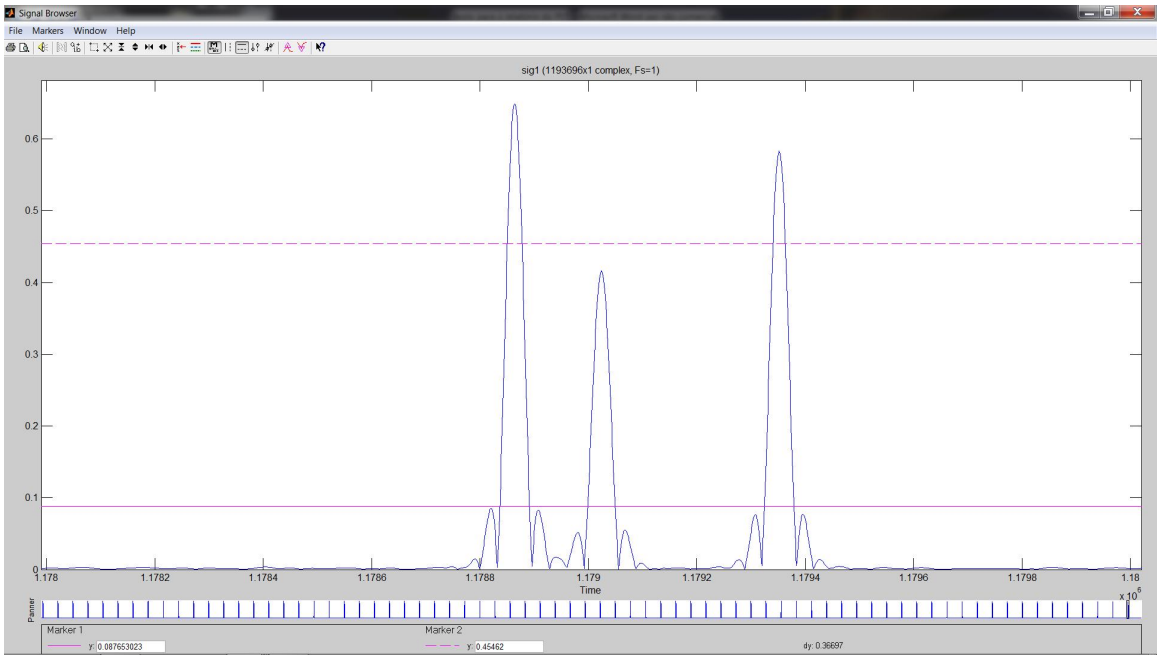

Figura 1: Resultado de uma medição mostrando três raios recebidos.

Em decorrência do método de sondagem, das distorções e dos ruídos, os resultados podem se apresentar de uma forma em que seja difícil identificar o máximo de cada trajeto detectado pela sonda, como indica a sequência de imagens apresentada na Figura 2. Nestas imagens, existem três grupos (*clusters*) de máximos, dos quais só interessa um único valor, relativo a um único instante em cada *cluster*.

Três funções foram criadas para a identificação tanto de *clusters* quanto do valor máximo dentro de cada *cluster*. Na função *filtraRaios*, os *raios* são comparados, de forma a localizar os que têm os maiores picos e, depois, retornar uma matriz com os respectivos tempos e amplitudes desses picos. Na função *analisaSinais*, o objetivo é comparar o sinal original com o sinal de cada método utilizado na análise (STDCC, filtro casado e OFDM) e, depois, imprimir o resultado que mais se aproxima do sinal original. A função *filtraRegioes* tem como objetivo agrupar os máximos em *clusters* e, dentro de cada um deles, escolher um único par "instante-valor" que represente o máximo absoluto do *cluster*.

Cada uma das funções criadas é descrita a seguir. No Apêndice A, as funções são listadas, sem comentários.

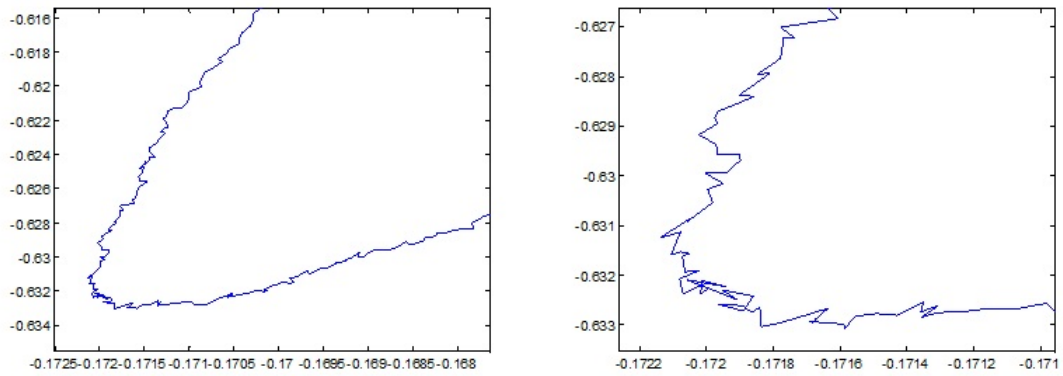

Figura 2: Sinais que possuem três *clusters*.

2 Função *filtraRaios*

A função *filtraRaios* receberá uma matriz representando um sinal. Esta matriz terá um número arbitrário de linhas e apenas duas colunas. A primeira coluna referir-se-á ao tempo e a segunda à amplitude do sinal, registrados em determinados pontos de observação. Cada linha da matriz representa um par "instante-valor". Com isso, a quantidade de linhas da matriz será igual à quantidade de pontos de observação. Esta função analisará o sinal, procurando por valores máximos de amplitude, acima de um patamar de ruído. O patamar de ruído será definido na chamada da função. O *patamarRuido* é o valor utilizado para filtrar os ruídos e qualquer valor abaixo deste será desconsiderado. Como resultado, ela retornará uma matriz com os valores máximos de amplitude e seus respectivos tempos.

A chamada da função define que ela possui dois parâmetros de entrada, que são *amplitudes* e *patamarRuido*, e um parâmetro de saída, *raios*.

```
function [raios] = filtraRaios(amplitudes, patamarRuido)
```

Primeiramente, será iniciado o vetor do parâmetro *raios*. Depois, será obtido o número de linhas da matriz.

```
raios = [ ];
nRaios = 0;
nPontos = size(amplitudes, 1);
if nPontos <= 1
    return;
end
```

No primeiro *if* será verificado se a primeira amplitude é um ponto de máximo. No *if* mais interno será verificado se há um ponto de máximo entre amplitudes que são iguais.

```
if amplitudes(1,2) > amplitudes(2,2)
    if amplitudes(1, :) >= patamarRuido
      raios = [ raios; amplitudes(1, :) ];
        nRaios = nRaios + 1;
    end
end
```

No *for* é verificado se os pontos intermediários são raios, sem considerar o último. Para um ponto ser um raio, sua amplitude precisa ser maior (ou igual) que a do ponto anterior e maior que a do ponto posterior. No final, verifica-se se o último ponto é um raio, bem como armazenam-se o tempo e a amplitude de cada raio em uma nova matriz.

```
    for i = 2 : nPontos - 1
        if(amplitudes(i, 2) >= amplitudes(i-1, 2)
                            && amplitudes(i,2)
                            >  amplitudes(i+1, 2) )
            if amplitudes(i, 2) >= patamarRuido
                raios = [ raios; amplitudes(i, :) ];
                nRaios = nRaios + 1;
            end
        end
    end
    if( amplitudes(nPontos, 2) > amplitudes(nPontos-1, 2) )
        if amplitudes(nPontos, 2) >= patamarRuido
            raios = [ raios; amplitudes(nPontos, :) ];
            nRaios = nRaios + 1;
        end
    end
end
```

Para executar a função, devemos proceder da seguinte forma: inicialmente, é digitada a matriz desejada, com 2 colunas e um número de linhas qualquer. Após digitada a matriz, a função será chamada. No exemplo abaixo, usaremos o patamar de ruído como sendo 0.15.

```
amps = [  0 0.1;   1 0.1;   2 0.1;   3 0.14;  4 0.13;  5 0.12;
          6 0.18;  7 0.2;   8 0.19;  9 0.17; 10 0.2;  11 0.20;
         13 0.27; 14 0.27; 15 0.35; 16 0.38; 17 0.34 ];

raios = filtraRaios(amps, 0.15);
```

Como resultado da função, teremos uma matriz de duas colunas com os valores dos máximos de amplitude acima de um patamar de ruído.

3 Função *analisaSinais*

Esta função tem como objetivo comparar o sinal original com o sinal de cada método utilizado na análise: STDCC, filtro casado e OFDM. Ela irá receber quatro matrizes de duas colunas: uma do sinal original e as outras dos três métodos. Primeiramente, serão capturados os raios de cada método, descartando o ruído de acordo com o patamar utilizado na função *filtraRaios*. Depois desse procedimento, serão encontrados os máximos de cada sinal. Por último, será calculado o delta dos máximos de cada método referente ao sinal original, e será mostrado como resultado o método que possuir o menor delta.

Na chamada da função, é informado que a mesma possui cinco parâmetros de entrada: *original*, *metodo1*, *metodo2*, *metodo3* e *patamarRuido*, bem como sete parâmetros de saída: *raiosOriginal*, *raiosMetodo1*, *raiosMetodo2*, *raiosMetodo3*, *delta1*, *delta2*, *delta3*.

```
function [raiosOriginal,
          raiosMetodo1, raiosMetodo2, raiosMetodo3,
          delta1, delta2, delta3] =
          analisaSinais (original,
                         metodo1, metodo2, metodo3,
                         patamarRuido)
```

Primeiro, os raios de cada método serão capturados, descartando os ruídos de acordo com o patamar definido. Depois, serão achados os picos de cada método e o delta de cada método será calculado.

```
raiosOriginal = filtraRaios(original, patamarRuido);
raiosMetodo1  = filtraRaios(metodo1,  patamarRuido);
raiosMetodo2  = filtraRaios(metodo2,  patamarRuido);
raiosMetodo3  = filtraRaios(metodo3,  patamarRuido);

picoOriginal = max( raiosOriginal(:, 2) );
picoMetodo1  = max( raiosMetodo1 (:, 2) );
picoMetodo2  = max( raiosMetodo2 (:, 2) );
picoMetodo3  = max( raiosMetodo3 (:, 2) );

delta1 = picoOriginal - picoMetodo1;
delta2 = picoOriginal - picoMetodo2;
delta3 = picoOriginal - picoMetodo3;
```

Após os raios terem sido encontrados, os picos de cada método terem sido capturados e os deltas calculados, as informações serão impressas.

```
fprintf('************************************\n');
fprintf('Sinal original\n');
fprintf('Numero de raios: %d\n', size(raiosOriginal, 1) );
fprintf('Raios: \n')

for i = 1:size(raiosOriginal, 1)
    fprintf('tempo: %f amplitude: %f\n',
             raiosOriginal(i, 1),  raiosOriginal(i, 2)  );
end

fprintf('\nMaior amplitude: %f\n', picoOriginal);
fprintf('************************************\n');

fprintf('Metodo 1\n');
fprintf('Numero de raios: %d\n', size(raiosMetodo1, 1) );
fprintf('Raios: \n')

for i = 1:size(raiosMetodo1, 1)
    fprintf('tempo: %f amplitude: %f\n',
           raiosMetodo1(i, 1),  raiosMetodo1(i, 2)  );
end
```

```
 fprintf('\nMaior amplitude: %f\n', picoMetodo1);
 fprintf('delta: %f\n', delta1);
 fprintf('**************************************\n');

 fprintf('Metodo 2\n');
 fprintf('Numero de raios: %d\n', size(raiosMetodo2, 1) );
 fprintf('Raios: \n')

 for i = 1:size(raiosMetodo2, 1)
     fprintf('tempo: %f amplitude: %f\n',
             raiosMetodo2(i, 1),  raiosMetodo2(i, 2)  );
 end

 fprintf('\nMaior amplitude: %f\n', picoMetodo2);
 fprintf('delta: %f\n', delta2);
 fprintf('**************************************\n');

fprintf('Metodo 3\n');
fprintf('Numero de raios: %d\n', size(raiosMetodo3, 1) );
fprintf('Raios: \n')

 for i = 1:size(raiosMetodo3, 1)
     fprintf('tempo: %f amplitude: %f\n',
              raiosMetodo3(i, 1),  raiosMetodo3(i, 2)  );
 end

 fprintf('\nMaior amplitude:\%f\n', picoMetodo3);
 fprintf('delta: %f\n', delta3);
 fprintf('**************************************\n');
```

Por último, será descoberto qual método é o mais eficaz e que melhor se aproxima do sinal original.

```
 menorDelta = min(  [abs(delta1) ; abs(delta2) ;
                     abs(delta3) ]  );

 fprintf('Metodos com menor delta: ' );

 if( abs(delta1) == menorDelta )
     fprintf(' metodo 1');
 end
 if( abs(delta2) == menorDelta )
     fprintf(' metodo 2');
 end
 if( abs(delta3) == menorDelta )
   fprintf(' metodo 3');
 end

 fprintf('\n')
end
```

O procedimento para executar a função será o seguinte: primeiramente, serão digitadas as quatro matrizes. Depois, a função será chamada. O patamar de ruído utilizado no exemplo abaixo também será de 0.15.

```
original = [ 0 0.10; 1 0.15; 2 0.25; 3 0.20 ; 4 0.05;
             5 0.20; 6 0.30; 7 0.45; 8 0.30 ];

metodo1  = [ 0 0.50; 1 0.10; 2 0.12; 3 0.10; 4 0.05; 5 0.20;
             6 0.30; 7 0.28 ];

metodo2  = [ 0 0.00; 1 0.54; 2 0.40; 3 0.25; 4 0.40; 5 0.35];

metodo3  = [ 0 0.00; 1 0.14; 2 0.28; 3 0.44; 4 0.36; 5 0.29;
             4 0.20; 5 0.12; 6 0.18; 7 0.27; 8 0.35; 9 0.40];

analisaSinais(original, metodo1, metodo2, metodo3, 0.15);
```

4 Função *filtraRegioes*

Esta função tem como objetivo procurar os pontos de máximo em uma região. Ela irá receber um patamar de ruído e uma distância mínima entre os picos. Após todas as posições terem sido verificadas, teremos um valor máximo dentro de cada região.

A função possui três parâmetros de entrada: *amplitudes*, *partamarRuido* (que é o valor utilizado para filtrar os ruídos) e *distMinRaios* (distância temporal mínima entre os raios). Se dois raios estiverem a uma distância temporal menor que este último valor, o menor raio será descartado. Por sua vez, há apenas um parâmetro de saída: *raios*.

```
function [raios] = filtraRegioes(amplitudes,patamarRuido, distMinRaios)
```

Primeiro, a função *filtraRaios* será chamada para obter os raios.

```
raios = filtraRaios(amplitudes, patamarRuido);
```

Agora, a matriz já possui os raios. É necessário percorrer essa matriz verificando se existem raios cuja distância seja menor que o parâmetro *distMinRaios*. No *if*, os raios são comparados para ver qual possui maior amplitude.

```
i = 1;
while i < size(raios, 1)
  if raios(i+1,1) - raios(i,1)  <= distMinRaios
    if raios(i, 2) > raios(i+1, 2)
      raios = [ raios(1:i, :)   ; raios(i+2:end, :) ];
    elseif raios(i+1, 2) > raios(i, 2)
      raios = [ raios(1:i-1, :) ; raios(i+1:end, :) ];
```

Nesse caso, os dois raios têm a mesma amplitude. Esse caso pode ocasionar um funcionamento incorreto do programa. Então, os dois raios ficarão na matriz. Os raios serão percorridos, para ver se há outro raio na região que será maior que eles.

```
      else
        ii = i+1;
        j = i+2;
        while j < size(raios, 1)
          if raios(j, 1) - raios(i, 1) <= distMinRaios
```

Neste ponto percebe-se que há outro raio na região. O raio j será eliminado e não será mais necessário incrementar o índice j. No ponto $i = j$ é verificado que saímos da região.

```
            if raios(i, 2) > raios(j, 2)
              raios = [ raios(1:j-1, :) ; raios(j+1:end, :) ];
            elseif raios(i, 2) < raios(j, 2)
              raios = [ raios(1:i-1, :) ; raios(ii+1:end, :) ];
            break
          else
            ii = ii + 1;
         end
        else
          i = j;
          break
        end
       end
      end
     else
        i = i+1;
     end
    end
   end
```

Para executar essa função, o procedimento é o seguinte: primeiro, definem-se as amplitudes. Depois, ela é chamada. Os valores para o *patamarRuido* e para o *distMinRaios* são definidos pelo usuário.

```
amps = [  0 0.1;   1 0.1;   2 0.1;   3 0.14;  4 0.13;  5 0.12;
          6 0.18;  7 0.2;   8 0.19;  9 0.17; 10 0.2;  11 0.20;
         13 0.27; 14 0.27; 15 0.35; 16 0.38; 17 0.34 ];

raios = filtraRegioes(amplitudes, patamarRuido, distMinRaios);
```

Apêndice A
Listagens das funções

Função *filtraRaios*

```
function [raios] = filtraRaios2(amplitudes, patamarRuido)
    raios = [];
    nRaios = 0;
    nPontos = size(amplitudes, 1); %obtendo número de linhas
    if nPontos <= 1
        return;
    end
    if amplitudes(1,2) > amplitudes(2,2)
        if amplitudes(1, :) >= patamarRuido
            raios = [ raios; amplitudes(1, :) ];
            nRaios = nRaios + 1;
        end
    end
    for i = 2:nPontos-1
        if(amplitudes(i, 2) >= amplitudes(i-1, 2)
                            && amplitudes(i,2)
                            >  amplitudes(i+1, 2) )
            if amplitudes(i, 2) >= patamarRuido
                raios = [ raios; amplitudes(i, :) ];
                nRaios = nRaios + 1;
            end
        end
    end
    if( amplitudes(nPontos, 2) > amplitudes(nPontos-1, 2) )
         if amplitudes(nPontos, 2) >= patamarRuido
              raios = [ raios; amplitudes(nPontos, :) ];
              nRaios = nRaios + 1;
         end
    end
end
```

Função *analisaSinais*

```
function [raiosOriginal, raiosMetodo1, raiosMetodo2, raiosMetodo3,
          delta1, delta2, delta3] = analisaSinais(original,
          metodo1, metodo2, metodo3, patamarRuido)

  raiosOriginal = filtraRaios2(original, patamarRuido);
  raiosMetodo1 = filtraRaios2(metodo1, patamarRuido);
  raiosMetodo2 = filtraRaios2(metodo2, patamarRuido);
  raiosMetodo3 = filtraRaios2(metodo3, patamarRuido);

  picoOriginal = max( raiosOriginal(:, 2) );
  picoMetodo1 = max( raiosMetodo1(:, 2) );
  picoMetodo2 = max( raiosMetodo2(:, 2) );
  picoMetodo3 = max( raiosMetodo3(:, 2) );

  delta1 = picoOriginal - picoMetodo1;
  delta2 = picoOriginal - picoMetodo2;
  delta3 = picoOriginal - picoMetodo3;

  fprintf('****************************************\n');
  fprintf('Sinal original\n');
  fprintf('Numero de raios: %d\n', size(raiosOriginal, 1) );
  fprintf('Raios: \n')

  for i = 1:size(raiosOriginal, 1)
      fprintf('tempo: %f amplitude: %f\n', raiosOriginal(i, 1),
               raiosOriginal(i, 2)  );
  end

  fprintf('\nMaior amplitude: %f\n', picoOriginal);
  fprintf('****************************************\n');

  fprintf('Metodo 1\n');
  fprintf('Numero de raios: %d\n', size(raiosMetodo1, 1) );
  fprintf('Raios: \n')

  for i = 1:size(raiosMetodo1, 1)
      fprintf('tempo: %f amplitude: %f\n', raiosMetodo1(i, 1),
               raiosMetodo1(i, 2)  );
  end

  fprintf('\nMaior amplitude: %f\n', picoMetodo1);
  fprintf('delta: %f\n', delta1);
  fprintf('****************************************\n');

  fprintf('Metodo 2\n');
  fprintf('Numero de raios: %d\n', size(raiosMetodo2, 1) );
  fprintf('Raios: \n')
```

```
  for i = 1:size(raiosMetodo2, 1)
      fprintf('tempo: %f amplitude: %f\n', raiosMetodo2(i, 1),
               raiosMetodo2(i, 2)  );
  end

  fprintf('\nMaior amplitude: %f\n', picoMetodo2);
  fprintf('delta: %f\n', delta2);
  fprintf('*****************************************\n');

  fprintf('Metodo 3\n');
  fprintf('Numero de raios: %d\n', size(raiosMetodo3, 1) );
  fprintf('Raios: \n')

  for i = 1:size(raiosMetodo3, 1)
      fprintf('tempo: %f amplitude: %f\n', raiosMetodo3(i, 1),
               raiosMetodo3(i, 2)  );
  end

  fprintf('\nMaior amplitude: %f\n', picoMetodo3);
  fprintf('delta: %f\n', delta3);
  fprintf('*****************************************\n');


  % Descobrindo o método mais próximo
  menorDelta = min(  [abs(delta1) ; abs(delta2) ; abs(delta3) ]  );
  fprintf('Metodos com menor delta: ' );
  if( abs(delta1) == menorDelta )
      fprintf(' metodo 1');
  end
  if( abs(delta2) == menorDelta )
      fprintf(' metodo 2');
  end
  if( abs(delta3) == menorDelta )
       fprintf(' metodo 3');
  end
  fprintf('\n')
end
```

Função *filtraRegioes*

```
function [raios] = filtraRegioes(amplitudes, patamarRuido, distMinRaios)

    raios = filtraRaios2(amplitudes, patamarRuido);

    i = 1;
    while i < size(raios, 1) %i não deve chegar ao último raio
        if raios(i+1,1) - raios(i,1)  <= distMinRaios
            if raios(i, 2) > raios(i+1, 2)
                raios = [raios(1:i,:)    ; raios(i+2:end,:)];
            elseif raios(i+1, 2) > raios(i, 2)
                raios = [raios(1:i-1, :) ; raios(i+1:end,:)];
            else
                ii = i+1;
                j = i+2;
                while j < size(raios, 1)
                    if raios(j, 1) - raios(i, 1) <= distMinRaios
                          if raios(i, 2) > raios(j, 2)
                            raios = [raios(1:j-1,:) ; raios(j+1:end,:)];
                        elseif raios(i, 2) < raios(j, 2)
                            raios = [raios(1:i-1,:) ; raios(ii+1:end,:)];
                            break
                        else
                            ii = ii + 1;
                        end
                    else
                        i = j;
                        break
                    end
                end
            end
        else
           i = i+1;
        end
    end
end
```