Curso Introdutório de MATLAB Aplicado a Processamento de Sinais

Juliana Camilo Inácio

Instituto Federal de Santa Catarina
Campus São José
juliana_camilo@yahoo.com.br

21 de Junho de 2011

Conteúdo Programático

Objetivos

- A proposta deste minicurso é familiarizar os alunos com o ambiente MATLAB, principalmente com os *toolbox Communications* e *Signal Processing*, para a aplicação de conceitos de processamento de sinais.
- Serão apresentadas as principais funções destes *toolbox*, de modo que, a partir daqui os alunos possam explorar o *software* para a aplicação dos problemas mais variados.

Apresentação dos *Toolbox*

- **Toolbox Communications:**
  - é uma extensão do ambiente MATLAB com funções, gráficos e interface gráfica para explorar, projetar, analisar e simular algoritmos de sistemas de comunicação.
- **Toolbox Signal Processing:**
  - é um conjunto das ferramentas baseadas no ambiente computacional numérico MATLAB. O *toolbox* suporta um número grande de operações de processamento de sinais, da geração de sinais à projeto e implementação de filtros, modelagem paramétrica e análise espectral.

Apresentação do *Toolbox*

- A função **help** do MATLAB é uma das funções mais importantes do *software*. Com uma documentação de alta qualidade, muitas das informações apresentadas aqui a seguir, foram retiradas desta.

Geração de Sinais

- No MATLAB podemos trabalhar com diferentes tipos de sinais.
- Aqui iremos gerar as funções mais conhecidas
  - senos
  - cossenos
  - exponenciais
  - sinais binários

  para a aplicar os conceitos de modulação, filtragem e FFT.
- Os mesmos conceitos aplicados à estas funções, podem ser aplicados à sinais externos (sinais de áudio e imagens, por exemplo) capturados e tratados pelo *software*.

Senos

- O comando **sin(x)** gera o seno dos elementos de x (em radianos)
- onde **x** pode ser, por exemplo, $2\pi ft$
- sendo f a frequência do sinal e t o intervalo de tempo no qual ele será amostrado

Senos

- Existem algumas formas de gerar o argumento da função seno.
- Podemos optar por definir um vetor **t** já considerando a frequência de amostragem do sinal
  - Ex: $t = 0 : 0,002 : 2 - 0,002$
  - que é o mesmo que $t = 0 : 1/fs : 2 - 1/fs$ sendo $fs \approx 460Hz$
- Ou podemos optar por colocá-la no próprio argumento
  - Ex: $\sin(\frac{2\pi ft}{f_s})$ onde o vetor **t** é apenas $t = 0 : 400$ por exemplo
- Ou ainda, podemos definir **x** como um intervalo
  - Ex: $x = -\pi : 1 : \pi$

Cossenos

- O comando **cos(x)** gera o cosseno dos elementos de x (em radianos)
- Assim como no seno, **x** pode ser $2\pi ft$, e todas as considerações aplicadas ao seno, podem ser aplicadas ao cosseno quanto ao seu argumento.
- Além do método tradicional, pode-se gerar um cosseno a partir de um seno, deslocando-do de 90 graus
  - Ex: $\sin(2\pi ft + \frac{\pi}{2})$

Exponenciais

- O comando **exp(x)** retorna uma exponecial dos elementos de x. Seu domínio inclui os números complexos.
- Assim como nas funções seno e cosseno, definimos o argumento **x**, com um vetor **t**
  - Ex: $exp(at)$ sendo **a** uma constante, positiva ou negativa, ou um número complexo.

Combinação de Sinais

- Podemos também fazer uma combinação desses sinais gerados. Ou seja, somar duas senóides com fases e frequências diferentes, ou a multiplicação de uma senóide com uma exponencial, gerando um seno amortecido.

Combinação de Sinais

- EXERCÍCIO
  - Gere o sinal sinusoidal sugerido abaixo, onde $f_1 = 10Hz$, $f_2 = 300Hz$ e $f_s = 500Hz$, plote-o e observe seu comportamento.
  - $g = 2\sin(\frac{2\pi f_1 t}{f_s}) + 3\cos(\frac{2\pi f_2 t}{f_s})$

Plot de Sinais discretos

- Os sinais mostrados até agora foram "plotados" no tempo contínuo, com a função **plot**.
- Podemos plotar os mesmos sinais no tempo discreto com a função **stem**.

Sequências Aleatórias

- No MATLAB, temos a possibilidade de gerar sequências aleatórias. Existem algumas funções que realizam esta tarefa:
  - rand
  - randint
  - randn
- Aqui veremos sobre as duas primeiras.

Sequências Aleatórias

- O comando **randint** gera uma sequência de números inteiros aleatórios distribuídos uniformemente.

- **randint(m,n,rg)**
  - gera uma matriz **m x n** de números inteiros, que variam numa faixa especificada pelo parâmetro **rg**. Por default, se **rg** não for especificado, a matriz será binária.

- **randint(m,n,rg,'state')**
  - neste caso a sequência tem um "estado inicial" especificado pelo parâmetro *state*; a aleatoriedade está apenas na primeira execução, pois se o comando for repetido para o mesmo estado inicial, a sequência será a mesma em todas as execuções.

Sequências Aleatórias

- O comando **rand** gera numeros pseudoaleatórios distribuídos uniformemente.
- O comando **rand** tem a mesma síntaxe do comando **randint**, a única diferença é que no comando **rand** os números gerados não são inteiros, e sim números fracionados.

Sequências Aleatórias

- Para facilitar a visualização do sinal gerado, utilizaremos a função **rectpulse**, que aplica a modelagem do pulso retangular a sequência de entrada.

- **rectpulse(x,nsamp)**
  - **x** - é o sinal de entrada
  - **nsamp** - número de amostras para cada elemento de x

Sequências Aleatórias

- EXERCÍCIO
  - Gere uma sequência binária, com comprimento 50 e com um estado inicial que você desejar. Formate o sinal digital com a função **rectpulse**, plote-o e observe o seu comportamento.

Modulação Analógica

- No *Toolbox Communications* existem algumas funções que realizam mudulação analógica.
  - ammod
  - fmmod
  - pmmod
  - ssbmod
- Vamos tomar como exemplo a modulação em amplitude (ammod), pois as demais seguem o mesmo padrão de síntaxe.

Modulação Analógica

- **ammod(x,fc,fs)**
  - **x** - sinal a ser modulado
  - **fc** - frequência da portadora
  - **fs** - frequência de amostragem
- **amdemod(y,fc,fs)**
  - **y** - sinal a ser demodulado
  - **fc** - frequência da portadora
  - **fs** - frequência de amostragem

Comparação de Sequências

- **[number,ratio] = biterr(x,y)** - computa o número de erro de bit e a taxa de erro de bit
  - **x** - sinal original
  - **y** - sinal recuperado
- **[number,ratio] = symerr(x,y)** - computa o número de erro de símbolo e a taxa de erro de símbolo
  - **x** - sinal original
  - **y** - sinal recuperado

Modulação Digital

- Assim como no caso de modulação analógica, no *Toolbox Communications* existem algumas funções que realizam mudulação digital.
  - pskmod
  - qammod
  - pammod
  - fskmod
- Vamos tomar como exemplo a modulação *Phase Shift keying* (pskmod), pois as demais seguem o mesmo padrão de síntaxe.

Modulação Digital

- **pskmod(x,M)** - modula o sinal x de acordo com o valor de M. Onde **M** é inteiro e potência de 2, e **x** deve ser formado por elementos inteiros entre 0 e M-1.
  - Ex: se M = 2, os elementos de **x** serão modulados entre 1 e -1 (BPSK)
- **pskdemod(y,M)** - demodula o sinal de acordo com o valor de M.

Modulação Digital

- **scatterplot** - plot "espalhado" do sinal
- Este tipo de plot ajuda no plot das constelações resultante da modulação digital.
- Este plot pode ser utilizado para diferentes aplicações, e para cada uma ele interpreta o dado de uma certa maneira
  - Se o sinal é complexo, ele interpreta a parte real como as componentes *in-phase* e a parte imaginária como as componentes em quadratura.

Canais de comunicação

- Para canais de comunicação, temos duas opções:
  - canais Rayleigh
  - canais Rician

- Aqui vamos optar pelos canais de Rayleigh, já que estes possuem a opção de múltiplos percursos, característica de sistemas de comunicação sem fio. Enquanto que os canais de Rician só possuem a opção de único percurso.

Canais de comunicação

- O comando **rayleighchan** constrói um objeto do canal de Rayleigh

```
>> rayleighchan

ans =

              ChannelType: 'Rayleigh'
        InputSamplePeriod: 1
          DopplerSpectrum: [1x1 doppler.jakes]
          MaxDopplerShift: 0
                PathDelays: 0
             AvgPathGaindB: 0
       NormalizePathGains: 1
             StoreHistory: 0
           StorePathGains: 0
                 PathGains: -0.3059 - 1.1777i
       ChannelFilterDelay: 0
     ResetBeforeFiltering: 1
      NumSamplesProcessed: 0
```

Canais de Comunicação

- **rayleighchan(ts,fd)** - canal com percurso único
  - **ts** - tempo de amostragem do sinal em segundos
  - **fd** - máximo desvio Doppler em Hertz
- **rayleighchan(ts,fd,tau,pdb)** - canal com múltiplos percursos
  - **tau** - vetor com os atrasos de cada percurso em segundos
  - **pdb** - vetor com o ganho de cada percurso em dB

Canais de Comunicação

- Com o objeto canal criado, podemos observar o seu comportamento através de **plot(chan)**.

- EXERCÍCIO:
  - Gere o canal de Rayleigh com múltiplos percursos sugerido abaixo e observe o seu comportamento através do plot do canal.

  - ts = 1/5000; fd = 4;
  - chan = rayleighchan(ts,fd,[0 1.5e-5 3.2e-5], [0, -3, -3]);

Inserção de Ruído

- a função **awgn** insere ruído branco Gaussiano ao sinal.
- **awgn(x,snr)**
  - **x** - sinal de entrada.
  - **snr** - relação sinal ruído para cada amostra do sinal,em dB

Filtros

- Os filtros podem sem gerados a partir de funções, ou pelas ferramentas FDATOOL e FVTOOL do *toolbox Signal Processing*.
  - Para abrir o FDATOOL, basta digitar **fdatool** na janela de comandos do MATLAB, que logo abrirá uma nova janela para o projeto de filtros. Existe bastante documantação sobre esta ferramenta no próprio **help** do *software*.
  - E **fvtool(b,a)** abre uma ferramenta de visualização de filtros, sendo **b** e **a** o numerador e o denominador da equação diferença do filtro.
- Aqui veremos apenas algumas funções para o projeto e implementação de filtros, dentre as várias que existem.

Filtros

- O comando **filter** pode ser usado tanto como um objeto, que implementa a filtragem de uma outra função, quanto como o próprio filtro.

- **filter(b,a,x)**
  - **b** - é o numerador da equação diferença do filtro
  - **a** - é o denominador da equação diferença do filtro
  - **x** - é o sinal a ser filtrado

- **filter(h,x)**
  - **h** - é o filtro a ser implementado
  - **x** - sinal a ser filtrado

- Neste caso, o filtro é implementado, e não há o acréscimo de amostras no sinal de saída.

Filtros

- O comando **conv** realiza a convolução e multiplicação polinomial entre duas funções.
- Este comando pode ser utilizado, assim como o **filter**, para a implementação de filtros já criados.
- A diferença é que este, devido ao processo de convolução, acrescenta no sinal de saída o número de amostras do filtro.

Filtros FIR

- O comando **fir1** projeta um filtro FIR baseado em janela
- **fir1(n,wn)** - configuração default (filtro Passa Baixa com janela de Hamming)
  - **n** - ordem do filtro.
  - **wn** - frequência de corte, variando de 0 a 1, sendo 1 a $fs/2$
- **fir1(n,wn,'ftype','window')**
  - **ftype** - tipo de filtro (Passa Baixa, Passa Alta, etc)
  - **window** - tipo de janela (Butterworth, Hamming, Chebyshev, etc)

Filtros FIR

- O comando **firrcos** projeta um filtro FIR cosseno levantado
- **firrcos(n,wn,df,fs)**
  - **n** - ordem do filtro
  - **wn** - frequência de corte, variando de 0 a 1, sendo 1 $fs/2$
  - **df** - largura da banda de transição
  - **fs** - frequência de amostragem

Filtros IIR

- A função **butter** projeta filtros IIR Butterworth
- **butter(n,wn,'ftype')**
  - **n** - é a ordem
  - **wn** - é a frequência de corte variando entre 0 e 1, sendo 1 $fs/2$
  - **ftype** - é o tipo de filtro (Passa Baixa, Passa Alta, etc)

Filtros IIR

- A função **cheby1** projeta filtros IIR Chebyshev
- **cheby1(n,R,wn,'ftype')**
  - **n** - é a ordem
  - **R** - é o pico do sobrepassamento na banda de passagem da função Chebyshev
  - **wn** - é a frequência de corte variando entre 0 e 1, sendo 1 $fs/2$
  - **ftype** - é o tipo de filtro (Passa Baixa, Passa Alta, etc)

FFT e IFFT

- O comando **fft** realiza a transformada discreta de Fourier.
  - **fft(x)**
- O comando **ifft** realiza a transformada discreta inversa de Fourier.
  - **ifft(x)**
- E o comando **fftshift** auxilia na hora da vizualização do espectro de frequências da resultante da FFT, pois ele reorganiza as saídas da função **fft**, para que a componente de frequência zero fique no meio do vetor.

Introdução de Imagens

- Entre as muitas linhas de pesquisa que envolvem o uso do *software* MATLAB, uma delas é a que trabalha com processamento de imagens.
- Veremos aqui uma breve abordagem, apenas para dar início à área que pode ser explorada.

Introdução de Imagens

- Podemos abrir imagens com o *software* MATLAB e realizar diversos testes.
- A imagem pode ser de diversos formatos (bmp, jpg, png, gif, pbm, etc...).
- Se a imagem for tratada por um algoritmo salvo em **.m**, ela deve estar salva no mesmo diretório em que este esteja.

Introdução de Imagens

- O comando **imread('nome da imagem.formato')** lê a imagem salva em algum diretório e armazena no *workspace* do *software*.
- O comando **imshow('nome da imagem.formato')** ou **imshow(nome da varialvel que armazena imagem)** mostra a imagem, como uma espécie de plot.
- As imagens tem dimensões diferentes de outros dados, por isso as vezes é conveniente reorganizar suas dimensões e o tipo de dado, para só depois processar os dados.
- Se a imagem for tratada por um algoritmo salvo em **.m**, ela deve estar salva no mesmo diretório em que este esteja.

Introdução de Imagens

- Vamos pegar o exemplo de uma imagem com dimensão $45 \times 45 \times 3$ com o dado do tipo **uint8**

- Essa matriz tem 3 dimensões, e para algumas funções é mais conveniente tê-la em forma de vetor, ou de uma matriz comum, com o dado do tipo *double*.

- As função a seguir auxiliam neste processo de reorganização da imagem
  - **dec2bin** - converte de decimal para binário;
  - **double** - converte o dado para precisão tipo double
  - **reshape** - reorganiza as dimensões da matriz
  - **bin2dec** - converte de binário para decimal
  - **char** - converte para dado do tipo character
  - **uint8** - converte o dado para o tipo uint8

Conclusão

- Com as noções passadas aqui já é possível realizar algumas simulações simples na área de processamento de sinais, e explorar mais o *software* para a solução de problemas.

Bibliografia

- MATHWORKS
  Signal Processing Toolbox 6 - User's Guide
  Natick, MA - version 6
- MATHWORKS
  Communications System Toolbox 5 - User's Guide
  Natick, MA - version 5
- LATHI, B.P
  Sinais e Sistemas Lineares
  Bookman, 2007.
- *Help do MATLAB*

Agradecimentos

Agradeço a todos pela presença!