PTC5005 - Processamento Digital de Sinais I
Lista 01- Familiarização com o Matlab

MTMS, MG, 1999
CI, MTMS, 2007
MTMS, 2010

Não é nessário entregar e não vale nota, mas é recomendável que façam o exercício para obter familiarização com o Matlab.
Observações: Entregue os programas em Matlab com comentários junto com os gráficos pedidos e responda todas as perguntas.

Objetivos

Este exercício computacional tem por objetivo a familiarização com o Matlab através de sua utilização para tratamento de sinais e sistemas de tempo discreto.

O Matlab

O MatLab (de *MATrix LABoratory*) é um programa produzido pela The Mathworks, Inc., que possui inúmeros recursos matemáticos a fim de tratar matrizes e números complexos de forma otimizada. Trata-se de uma linguagem interpretada, com estrutura semelhante a C ou Pascal, porém voltada para processamento de vetores e matrizes. A interface do Matlab é composta basicamente de uma janela de comandos, com um *prompt* característico (>>). Além dessa janela podem coexistir diversas janelas gráficas, denominadas *figures*.

1.1 Introdução básica ao uso do Matlab

Após ter inicializado o Matlab, observe se o diretório que você deseja trabalhar é o que aparece em *Current Directory*. Se este não for o caso, procure o diretório desejado usando os botões

Os comandos `dir`, `demo` e `help` podem ser úteis. O primeiro dá uma lista dos arquivos do diretório, já os outros dois podem ser usados para se ter uma idéia geral do funcionamento do Matlab, por exemplo:

```
>> help inv
 INV    Matrix inverse.
    INV(X) is the inverse of the square matrix X.
    A warning message is printed if X is badly scaled or
    nearly singular.

    See also slash, pinv, cond, condest, lsqnonneg,
lscov.

    Overloaded methods:
       gf/inv
       lti/inv
       idmodel/inv
       uss/inv
       umat/inv
       ufrd/inv
       ndlft/inv
       atom/inv
       sym/inv

    Reference page in Help browser
       doc inv
```

Os comandos no Matlab devem sempre ser em letra minúscula. O comando `help`, mostrado anteriormente, fornece informações a respeito de uma função específica e além disso dá o nome de outras funções relacionadas à primeira.

Vetores linha são definidos entre colchetes, usando ou não vírgulas para separar os elementos. Por exemplo:

```
>>  b=[1, 2, 3];
```

Um vetor coluna é obtido da mesma forma, mas a separação entre os elementos deve ser feita com ponto e vírgula. Uma outra forma bastante útil é a que segue:

```
>>  n=-10:1:100;
```

Neste caso, o comando acima define um vetor n contendo números de –10 a 100 com passo de 1, isto é [-10, -9, -8,…,0,1,2,…,100]. O ponto e vírgula usado após o comando serve para que os elementos do vetor não sejam visualizados na janela de comandos. Além disso, o Matlab faz distinção entre letras minúsculas e maiúsculas na definição das variáveis. É muito útil também definir vetores de zeros ou uns. Para isso, deve-se usar as funções `zeros` ou `ones`. Dê os comandos

```
>> help zeros
>> help ones
```

para ver como elas funcionam.

Na definição de matrizes, deve-se separar as linhas usando ponto e vírgula. Por exemplo, vamos definir as matrizes A e B:

```
>>  A=[5 6 –2; 3 8 0; 4 2 4]
>>  B=[1 2 –4; 5 7 1; 5 0 2]
```

Com isso, agora se pode fazer algumas operações matriciais simples:

```
>> C=A*B
>> D=inv(C’)
```

sendo C a multiplicação das matrizes A e B, e D a inversa da matriz C transposta. Outras operações mais complexas também podem ser realizadas, ou seja,

```
>> [F, G]= eig(C)
>> H1=A(:,1)*B(2,:)
>> H2=A(:,1).*B(2,:)'
```

sendo F uma matriz formada pelos autovetores de C, G uma matriz diagonal formada pelos autovalores de C, H1 o produto da primeira coluna de A com a segunda linha de B e H2 o produto elemento a elemento da primeira coluna de A com a segunda linha de B transposta. O simbolo (“:”) significa “todos os elementos”. Assim, quando se escreve `A(:,1)` leia-se “todas as linhas de A da primeira coluna”, que neste caso é um vetor de dimensão 3 `x` 1. O operador “`.*`” faz multiplicação elemento a elemento, isto é, o primeiro elemento de um vetor multiplicado pelo primeiro elemento do outro, o segundo elemento de um pelo segundo do outro, e assim por diante.

Para saber a dimensão de uma determinada variável, deve-se usar o comando `size`, experimente por exemplo:

```
>> S=size(A(:,1))
```

Para verificar as variáveis já definidas utilize o comando `whos`:

```
>> whos
```

Se você seguiu todos os comandos até agora deve aparecer na janela de comandos:

```
  Name      Size                Bytes  Class     Attributes

  A         3x3                    72  double
  B         3x3                    72  double
  C         3x3                    72  double
  D         3x3                    72  double
  F         3x3                   144  double    complex
  G         3x3                   144  double    complex
  H1        3x3                    72  double
  H2        3x1                    24  double
  S         1x2                    16  double
  ans       3x3                    72  double
  b         1x3                    24  double
  n         1x111                 888  double
```

Para apagar as variáveis da área de trabalho, use o comando `clear`:

```
>> clear
```

Este comando pode ser usado para apagar apenas algumas variáreis específicas. Para maiores detalhes, veja o "help" desse comando.

Existem várias funções matemáticas já definidas no Matlab como a função co-seno, exponencial, etc. Para obter a lista dessas funções dê o seguinte comando:

```
>>  help elfun
```

O número π=3,1415923… é definido pela variável `pi` e `j` ou `i` representam $\sqrt{-1}$. Assim

```
>>  j*exp(j*11*pi/4)
ans =

  -0.7071 - 0.7071i
```

Em algumas situações é necessário usar comandos de linguagem de programação como `for`, `while`, `if`, etc. Para ver a lista desses comandos, dê:

```
>>  help lang
```

Cabe observar que esses comandos devem ser evitados sempre que possível, pois exigem maior tempo de processamento.

Arquivos denominados *M-files*, escritos na linguagem de programação do Matlab, podem ser de dois tipos: *functions* e *scripts*. *Scripts* são simplesmente sequências de comandos dispostos em um arquivo texto com extensão ".m". *Functions* são estruturas que também possuem extensão ".m", porém são mais complexas e apresentam certas características básicas:

- Passagem de parâmetros com o *workspace* (conjunto de variáveis que podem ser vistas na janela de comandos através do comando `whos` ou `who`);

- O ambiente de cada função é independente, isto é, uma função não reconhece variáveis definidas fora dela. Outras funções e/ou o *workspace* também não reconhecem variáveis definidas nela.

Em qualquer arquivo do tipo *M-file*, a linha iniciada com o símbolo "%" é considerada como comentário.

É importante inicializar as variáveis (escalares, vetores ou matrizes) no início do programa quando comandos do tipo **`for`** ou **`while`** são utilizados. Nestes casos, se uma variável não for inicializada e sua dimensão mudar a cada iteração, o tempo de processamento será excessivo e o programa ficará lento. Para inicializar um vetor ou uma matriz com zeros pode-se usar a função `zeros` do Matlab. Por exemplo, seja `y` um vetor de dimensão 100 `x` 1 que é saída de uma função. Esse vetor pode ser inicializado com zeros, inserindo no início da função o comando `y=zeros(100,1)`.

Para maiores detalhes, no endereço

http://www.mathworks.com/academia/student_center/tutorials/launchpad.html

você encontrará tutoriais do Matlab criados por professores e/ou alunos de diferentes universidades americanas.

1.2 Como gerar gráficos com o Matlab

Suponha que se deseje obter gráficos das seguintes funções de tempo discreto $f(n) = \frac{1}{2}\sin\left(\frac{\pi}{8}n + \frac{\pi}{3}\right)$ e $g(n) = \cos\left(\frac{\pi}{12}n\right)$ no intervalo $n = 0,1,2,\ldots,30$.

Primeiramente vamos gerar essas sequências através dos seguintes comandos:

```
»   n=0:1:30;
»   f=0.5*sin(pi*n/8+pi/3);
»   g=cos(pi*n/12);
```

Como se trata de duas sequências discretas, vamos utilizar a função `stem` do Matlab para obter o resultado da Figura 1.

```
» figure(1)
» subplot(211)
» stem(n,f)
» grid
» xlabel('n')
» ylabel('f(n)')
» subplot(212)
» stem(n,g)
» grid
» xlabel('n')
» ylabel('g(n)')
```

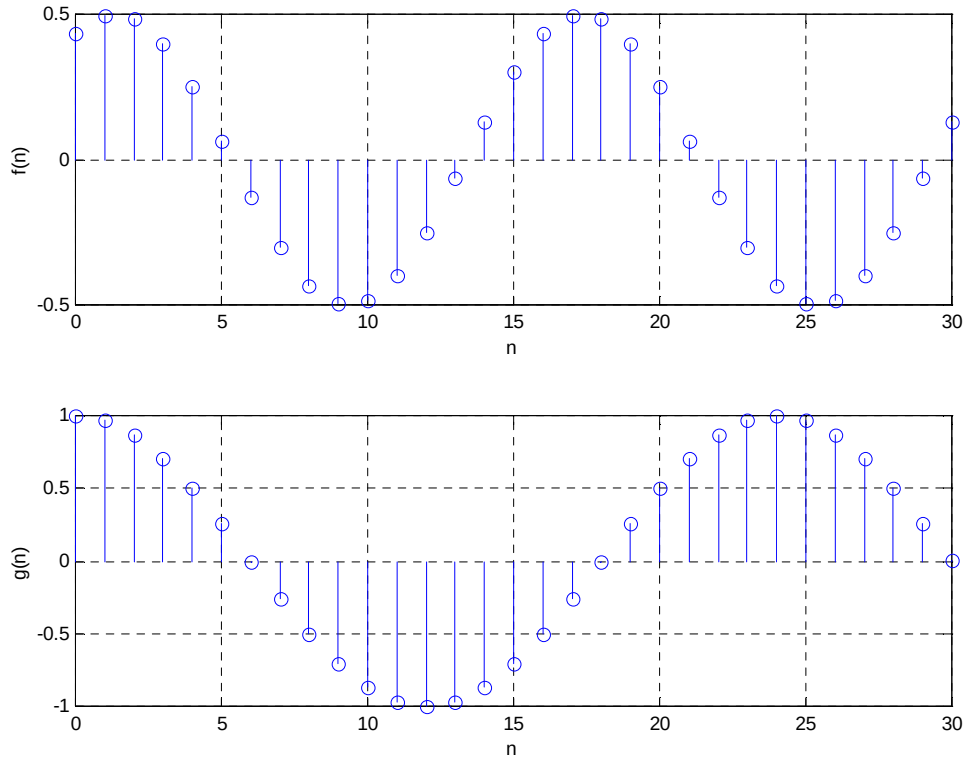

Figura 1

A função `subplot` divide o espaço de uma janela de figura em espaços menores criando subfiguras dentro de uma mesma janela. No caso anterior, foram criadas duas subfiguras dentro da `figure(1)`: uma na linha 1 e coluna 1 (`subplot(211)`) e outra na linha 2 e coluna 1 (`subplot(212)`). Os comandos `xlabel` e `ylabel` são usados para colocar legendas na abscissa e ordenada respectivamente.

Em muitas situações, mesmo se tratando de sequências discretas, para uma melhor visualização, pode-se desejar obter gráficos contínuos. Além disso, para efeito de comparação, pode ser útil obter vários gráficos numa mesma figura. Neste caso, para facilitar a distinção dos mesmos é necessário considerar cores e marcadores diferentes. Considerando as funções $f(n)$ e $g(n)$, obtém-se a Figura 2 através dos seguintes comandos:

```
» figure(2)
» plot(n,f,'-ob')
» hold on
» plot(n,g,'-*r')
» hold off
» grid
» xlabel('n')
» legend('f(n)','g(n)')
```

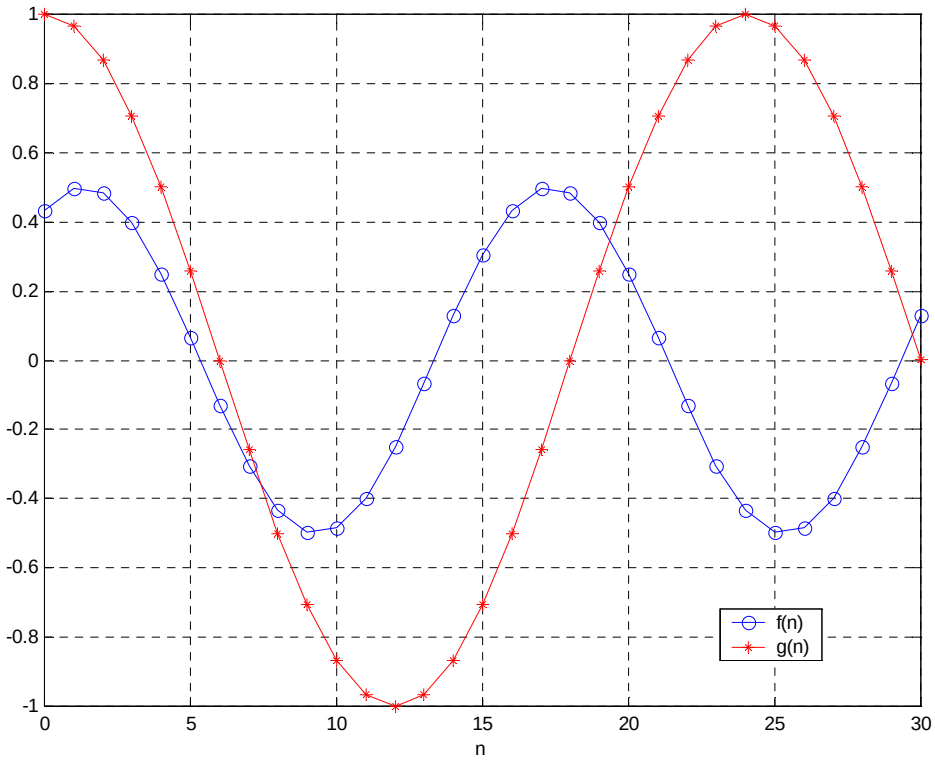

Figura 2

Na Figura 2, ao invés da função `stem`, foi utilizada a função `plot`. Além disso, para manter o gráfico anterior na mesma figura, foi utilizada a função `hold on`. No Matlab exitem uma série de funções relacioandas a gráficos 2D e 3D. Para obter uma lista destas funções, dê os comandos:

```
>> help graph2d
>> help graph3d
```

1.3 Geração de sinais de tempo discreto utilizando o Matlab

1.3.1 Pulso unitário

O pulso unitário deslocado é um sinal de tempo discreto definido como:

$$\delta(n-n_o)=\begin{cases}1 & n=n_o\\ 0 & n\neq n_o\end{cases}.$$

Para criar um pulso unitário no Matlab, deve-se decidir o quanto de sinal é de interesse. Suponha que se deseja obter L amostras da resposta ao pulso unitário de um sistema, então deve-se usar um pulso unitário de comprimento L. Para L=31, o seguinte código em Matlab criará o pulso unitário em questão:

```
>>  L=31;
>>  n=0:1:L-1;
>>  imp=zeros(L,1);
>>  imp(1)=1;
```

O índice $n = 0$, corresponde a `imp(1)=1`, pois no Matlab o índice dos vetores começa sempre em 1.

Exercício proposto 1:

Pulsos unitários deslocados, $\delta(n - n_o)$, podem ser usados para construir um trem de pulsos com período *P*, largura total *MP* e pesos A_l:

$$s(n) = \sum_{l=0}^{M-1} A_l \ \delta(n - lP).$$

Se os pesos A_l forem iguais, o trem de pulsos é periódico de período *P*. Faça um *script* para obter um gráfico do trem de pulsos, considerando $A_l = 2$, $l = 0, \ldots, M-1$, *P=5 e M=10*. Não utilize nenhum comando de linguagem de programação como `for`, `while`, `if`, entre outros.

1.3.2 Sinal Senoidal

Outro tipo de sinal bastante útil é o sinal senoidal. Apenas três parâmetros são necessários para sua completa descrição: amplitude (A), frequência (ω_o) e fase (ϕ).

$$x(n) = A\cos\left(\omega_o n + \varphi\right)$$

Exercício proposto 2

Faça um programa do tipo *script* que obtenha um gráfico de cada uma das seguintes sequências:

$$x_1(n) = \sin\frac{\pi}{17}n, \qquad 0 \le n \le 25$$

$$x_2(n) = \sin\frac{\pi}{17}n, \qquad -15 \le n \le 25$$

$$x_3(n) = \sin\left(3\pi\, n + \frac{\pi}{2}\right), \qquad -10 \le n \le 10$$

$$x_4(n) = \cos\left(\frac{\pi}{\sqrt{23}}n\right), \qquad 0 \le n \le 50$$

Use a função `stem` para a obtenção dos gráficos. Em cada caso, o eixo horizontal deve-se estender apenas no intervalo considerado e ser legendado de forma correspondente. Escreva $x_3(n)$ sem usar funções trigonométricas. Explique porque $x_4(n)$ não é uma sequência periódica.

Exercício proposto 3

Utilizando a função `square` do Matlab gere uma onda quadrada com período $T = 3\,\text{ms}$, frequência de amostragem $f_a = 10\text{KHz}$ e amplitudes 0 ou 1. Obtenha um gráfico de 3 períodos deste sinal em função do tempo em ms.

Dica: Seja $s(t) = \sin(2\pi ft)$ um sinal senoidal de tempo contínuo com frequência f. O correspondente sinal amostrado com frequência de amostragem f_a é dado por $s(n) = \sin\left(2\pi\, n \ \ f / f_a\right)$ com $n = 1,2,\ldots$

1.3.3 Ruído Branco

O ruído é um sinal presente em quase todos os sistemas de comunicação. Geralmente, ele é considerado gaussiano, branco, de média nula e variância σ^2. Para gerar esse sinal pode-se utilizar a função `randn`. No Matlab dê o comando

```
>> help randn
```

para ver como esta função funciona. Note que ela gera uma sequência branca e gaussiana com média nula e variância unitária. Para adequar essa sequência à variância σ^2 do ruído, o resultado desta função deve ser multiplicado pelo desvio padrão σ.

1.4 Média Móvel

Um exemplo de sistemas de tempo discreto LIT é a chamada média móvel que pode ser descrita através da equação:

$$y(n) = \frac{1}{M_1 + M_2 + 1} \sum_{k=-M_1}^{M_2} x(n-k).$$

Exercício proposto 4

a) Complete o programa em Matlab a seguir para calcular a média móvel do sinal $x(n)$ constituído de 100 amostras. Foi considerado ainda $M_1 = 0$ e $M_2 = 3$. Explique o resultado obtido.

b) Aplique o programa do item a) considerando que $x_2(n)$ é um ruído branco de média zero e variância $\sigma_{x_2}^2 = 0,01$.

```
clear
M1=0;
M2=3;
cte=1/(M1+M2+1);
% Sinal original acrescido de zeros
n=0:99;
a=ones(1,5);
b=[ones(1,10) zeros(1,10)];
x1=kron(a,b);
x2=.05*sin(3*pi*n+pi/2);
x0=x1+x2;
L=length(x0);
x=x0;

figure(1)
stem(n,x0);
title('Sinal original')
ylabel('x[n]')
xlabel('n')

x=[zeros(1,M2),x,zeros(1,M1)];

for i=......:L+......
   xx=......;
   for k=-M1:M2
      xx=xx+x(i-k);
   end
   y(i-M2)=cte*......;
end

figure(2)
stem(n,y)
title('Sinal obtido com a média móvel')
ylabel('y[n]')
xlabel('n')
```

1.5 Função `conv` e função `filter` do Matlab

Exercício proposto 5

Usando a sequência $x(n)$ do exercício proposto 4 a) e b), calcule a média móvel ($M_1 = 0$, $M_2 = 3$) de duas maneiras:

a) **Usando a função `conv` do Matlab;**
b) **Usando a função `filter` do Matlab.**

Qual a diferença entre as funções `conv` e `filter` do Matlab? Compare os resultados obtidos com os do exercício proposto 4.

Dica: Na janela de comandos do Matlab digite:

```
>> help conv
>> help filter
```

para saber como essas funções funcionam.

1.6 Resposta ao pulso unitário e em frequência de um sistema LIT.

Neste item, será obtida a resposta ao pulso unitário de um sistema linear invariante no tempo, cuja equação de diferenças linear com coeficientes constantes é do tipo:

$$\sum_{k=0}^{N_a} a_k y(n-k) = \sum_{l=0}^{N_b} b_l x(n-l)\,.$$

No Matlab, equações de diferenças são representadas por dois vetores, um contendo os coeficientes b_l dos termos *x* e outro os coeficientes a_k dos termos *y*. O coeficiente a_o é usualmente tomado igual a 1. A saída do sistema no instante de tempo *n* pode ser escrita como

$$y(n) = -\frac{1}{a_o}\sum_{k=1}^{N_a} a_k y(n-k) + \sum_{l=0}^{N_b} b_l x(n-l)$$

A função `filter` faz uma divisão por a_o e por isso esse coeficiente deve ser não nulo.

Exercício proposto 6

Seja a seguinte equação de diferenças:

$$y(n) - 2r\cos(\omega_o)y(n-1) + r^2 y(n-2) = x(n) - r\cos(\omega_o)x(n-1)\,,$$

sendo $r = 0.9$ e $\omega_o = \pi/16$. Deve-se usar a função `filter` para obter a resposta ao pulso unitário e um gráfico da mesma no intervalo $-10 \le n \le 100$. Deve-se ainda usar a função `freqz` para obter a resposta em frequência do sistema.

Dica: Na janela de comandos do Matlab digite :

```
>> help filter
>> help freqz
```

para saber como essas funções funcionam.

Exercício proposto 7

Escreva uma função no Matlab para obter a resposta ao pulso unitário e em frequência de um sistema de tempo discreto a partir dos vetores `a` e `b` que descrevem sua equação de diferenças.

A função deve ter como parâmetros de entrada:

`a,b->` vetores de coeficientes que descrevem a equação de diferenças da seguinte forma:

```
a(1)*y(n)+a(2)*y(n-1)+…+a(Na)*y(n-Na-1)=
b(1)*x(n)+b(2)*x(n-1)+…+b(Nb)*x(n-Nb-1)
```

e como parâmetros de saída:

`H->` Resposta em frequência

`W->` Frequência em radianos

`h->` Resposta ao pulso unitário no intervalo `[-10,100]`

Dicas:

Na janela de comandos do Matlab, digite

```
>> help function
```

para ver como definir uma função. Os comandos `keyboard` e `return` podem ser úteis quando se deseja verificar erros nas funções. Para parar uma simulação digite Ctrl+C.