UNIVERSIDADE FEDERAL DO CEARÁ

CENTRO DE TECNOLOGIA

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

PROGRAMA DE EDUCAÇÃO TUTORIAL

Apostila de MATLAB 7.3

Decio Haramura Junior

Guilherme Martins Gomes Nascimento

Luís Paulo Carvalho dos Santos

Luiz Fernando Almeida Fontenele

Pedro André Martins Bezerra

Fortaleza – CE

Maio / 2009

PET Eng. Elétrica

SUMÁRIO

1. PREFÁCIO

Esta apostila foi desenvolvida por alunos do Programa de Educação Tutorial (PET) do curso de Engenharia Elétrica da Universidade Federal do Ceará (UFC) para a realização do Curso de MATLAB.

Com o intuito de promover uma introdução ao MATLAB que viesse a facilitar o desempenho dos estudantes da graduação na realização de seus trabalhos e na sua vida profissional, o PET elaborou este Curso de MATLAB que está atualmente na quarta edição, sendo as três primeiras realizadas durante o ano letivo de 2008 e a última em 2009. Durante as quatro edições foram contemplados aproximadamente 250 estudantes dos mais variados cursos de Engenharia do Centro de Tecnologia da UFC.

Devido à sua boa repercussão, o Curso de MATLAB foi premiado no XVII Encontro de Iniciação à Docência nos Encontros Universitários de 2008.

2. APRESENTAÇÃO

O MATLAB (MATrix LABoratory) é uma linguagem de alto desempenho para computação técnica. Integra computação, visualização e programação em um ambiente de fácil uso onde problemas e soluções são expressos em linguagem matemática. Usos típicos:

- Matemática e computação;
- Desenvolvimento de algoritmos;
- Aquisição de dados;
- Modelagem, simulação e prototipagem;
- Análise de dados, exploração e visualização;
- Construção de interface visual do usuário.

2.1. Utilizando o HELP

Indubitavelmente, a melhor apostila tutorial sobre o MATLAB que possa existir é o HELP do próprio MATLAB. Todas as informações possíveis há no

HELP, principalmente sobre as *toolboxes*, sobre funções, SIMULINK e entre outros.

O HELP pode ser aberto de várias formas. A primeira é através da barra de menu, como mostrado na Figura 1:

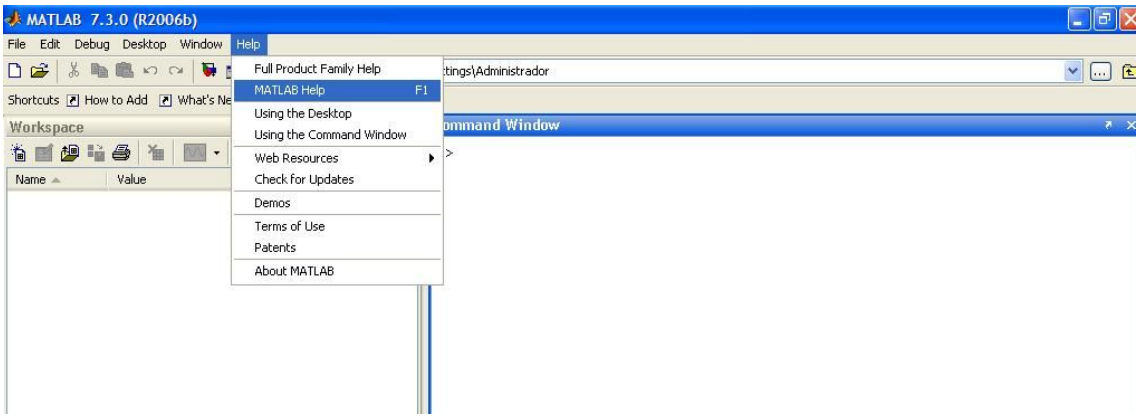


Figura 1 – HELP do MATLAB sendo acessado pela barra de menu.

Outra forma é pela tecla de atalho *F1*. Uma terceira forma é pelo botão *START*, posicionado logo abaixo do *COMMAND HISTORY,* de acordo com a Figura 2.

Figura 2 – HELP do MATLAB sendo acessado pelo botão *START*.

Dando continuidade, quando se deseja obter informações sobre uma dada função, é possível consultar diretamente no HELP ou pelo *COMMAND WINDOW*. Para isso, basta digitar *help* e em seguida a função requerida, de acordo com o exemplo abaixo:

```
>> help dirac
DIRAC Delta function.
DIRAC(X) is zero for all X, except X == 0 where it is
```

```
        infinite.
        DIRAC(X) is not a function in the strict sense, but rather
        a
        distribution with int(dirac(x-a)*f(x),-inf,inf) = f(a) and
        diff(heaviside(x),x) = dirac(x).
        See also heaviside.
        Overloaded functions or methods (ones with the same name in
        other directories)
        help sym/dirac.m
        Reference page in Help browser
        doc dirac
```

Veja que as informações sobre a função *dirac* aparecem no próprio *COMMAND WINDOW*. Se for necessário consultar a página do HELP, basta utilizar o comando *doc* e em seguida o nome da função. Por exemplo:

```
        >> doc dirac
```

Depois de efetuado este comando, irá aparecer a janela do HELP com o seguinte:

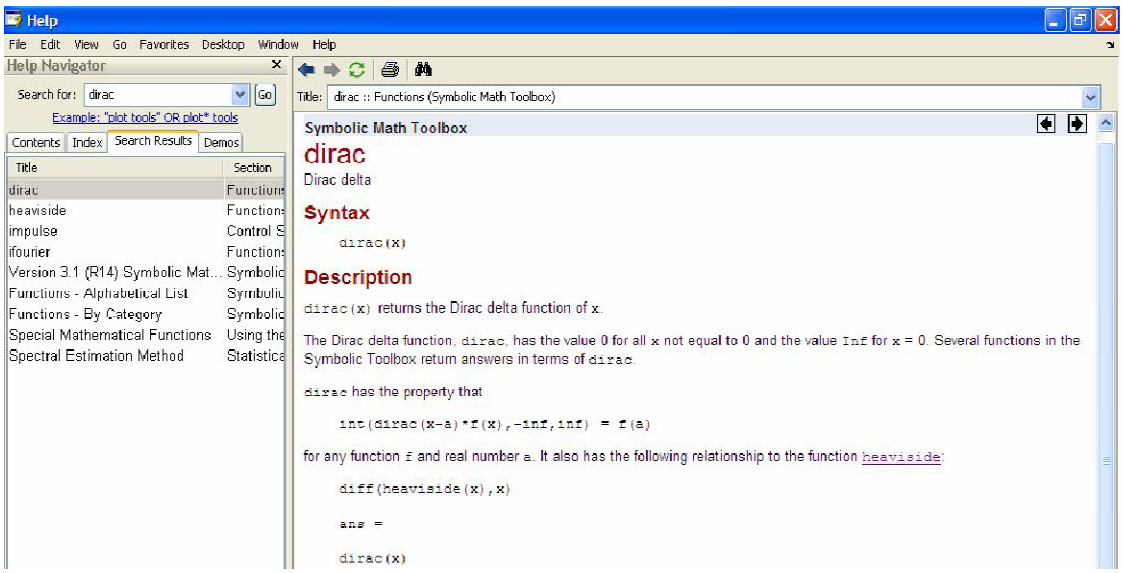


Figura 3 – HELP da função *dirac*.

3. FORMATAÇÃO

No MATLAB não há necessidade de declarar o tipo das variáveis utilizadas no programa, mas o usuário pode escolher qual o formato que vai ser utilizado. São usados os comandos mostrados na Tabela 1:

Tabela 1 - Formato das variáveis

Comando MATLAB	Variável	Descrição
Format long	3.141592653589793	Com 16 dígitos
Format short	3.1416	Com 5 dígitos
Format short e	3.1416e+000	Com 5 dígitos – notação científica
Format long e	3.141592653589793e+000	Com 16 dígitos em notação científica
Format +	+	Retorna “+” para valores positivos e “-” para valores negativos
Format rat	355/113	Aproximação racional
Format hex	400921fb54442d18	Formato hexadecimal

4. MATRIZES

4.1. Declaração

A declaração de matrizes é feita da seguinte maneira:

```
       >> a = [1:10]          %cria o vetor linha [1 2 3 4 5 6 7 8 9
10]
       >> b = [0:0.5:3]       %cria o vetor [0 0.5 1 1.5 2 2.5 3]
       >> A = [16 3 2 13; 5 10 11 8; 9 6 7 12; 4 15 14 1]
       A =
           16     3     2    13
            5    10    11     8
            9     6     7    12
```

```
     4    15    14     1
>>A(1,2);     %Elemento de linha 1 e coluna 2
>>A(:,3);     %Elementos da coluna 3
>>A(1,:);     %Elementos da linha 1
```

O MATLAB também aceita a concatenação de matrizes, por exemplo:

```
>> a=[ 4 1 ; 3 4];
>> b= [ 2  3; 4 5];
>> c=[a b];
c =
     4     1     2     3
     3     4     4     5
```

Obs.: É bom lembrar que o MATLAB tem como primeiro índice do vetor o número 1, diferente de outras linguagens que usam o primeiro índice como 0.

4.2. Soma

A soma de todos os elementos de uma matriz com um número é feita da seguinte maneira:

```
>> c =
     4     1     2     3
     3     4     4     5
>> c+1
ans =
     5     2     3     4
     4     5     5     6
```

A soma de matrizes é feita da maneira tradicional:

```
>> d=[ 1 2 7 8 ; 4 7 5 8] ;
>> e=[ 5 -4 7 0; 3 -1 6 -4];
>> d+e
ans =
     6    -2    14     8
```

```
     7     6    11     4
```

4.3. Multiplicação

Usa-se o sinal da multiplicação:

```
>> a=[1 4 2; 7 8 5; 9 5 4];
>>  b=[4 2 -5; 0 1 3; 8 -2 1];
>>  c=a*b
c =
    20     2     9
    68    12    -6
    68    15   -26
```

Obs.: Se for desejado realizar outra operação matemática (exceto a soma e a subtração) entre os elementos com mesmo índice das matrizes deve-se colocar um ponto antes do operador. Observe os exemplos abaixo:

```
>> a=[1 4 2; 7 8 5; 9 5 4];
>> b=[4 2 -5; 0 1 3; 8 -2 1];
>> c=a.*b
c =
     4     8   -10
     0     8    15
    72   -10     4
>> b./a
ans =
    4.0000    0.5000   -2.5000
         0    0.1250    0.6000
    0.8889   -0.4000    0.2500
>> a.^2
ans =
     1    16     4
    49    64    25
    81    25    16
```

Exercício 1- Declare as matrizes A, B e C abaixo:

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{bmatrix} \qquad B = \begin{bmatrix} 3 & 6 & 9 & 12 & 15 & 18 & 21 \end{bmatrix}$$

$$C = \begin{bmatrix} 0 & 5 & 10 & 15 & 10 & 5 & 0 \\ -1 & -2 & -3 & -4 & -5 & -6 & -7 \end{bmatrix}$$

Através das matrizes acima, determine as matrizes a seguir utilizando os comandos já mencionados.

$$D = \begin{bmatrix} 4 & 5 & 6 & 7 \end{bmatrix};$$

$$E = \begin{bmatrix} 7 & 6 & 5 & 4 \end{bmatrix};$$

$$F = \begin{bmatrix} 4 & 5 & 6 & 7 \\ 7 & 6 & 5 & 4 \end{bmatrix};$$

$$G = \begin{bmatrix} 4 & 5 \\ 7 & 6 \\ 6 & 7 \\ 5 & 4 \end{bmatrix};$$

$$H = \begin{bmatrix} 12 \\ 15 \\ 18 \\ 21 \end{bmatrix};$$

$$I = \begin{bmatrix} 5 \\ -6 \end{bmatrix};$$

$$J = \begin{bmatrix} 0 & 5 & 10 \\ -1 & -2 & -3 \end{bmatrix};$$

$$K = \begin{bmatrix} 0 & 25 & 100 \\ 1 & 4 & 9 \end{bmatrix};$$

$$L = \begin{bmatrix} 0 & 24 & 99 \\ 2 & 5 & 10 \end{bmatrix};$$

$$M = \begin{bmatrix} 0 & -24 & -99 \\ -2 & -5 & -10 \end{bmatrix};$$

4.4. Matrizes Pré-Definidas

- **ones**

Definição: Esta função gera uma matriz cujos valores são unitários.

Sintaxe:

ones(n) → Gera uma matriz quadrada de ordem *n* cujos termos são unitários.

ones(m,n) → Gera ma matriz *m* x *n* cujos termos são unitários.

```
>>ones(2)
ans =
     1     1
     1     1
```

- **zeros**

Definição: Esta função gera uma matriz cujos valores são nulos.

Sintaxe:

zeros(n) → Gera uma matriz quadrada de ordem *n* cujos termos são nulos.

zeros(m,n) → Gera ma matriz *m* x *n* cujos termos são nulos.

```
>>zeros(2)
ans =
     0     0
     0     0
```

- **eye**

Definição: Gera uma matriz identidade.

Sintaxe:

eye(n) → Gera uma matriz identidade *n* x *n*.

eye(m,n) → Gera uma matriz de ordem *m* x *n* cujos termos que possuem *i=j* são unitários.

```
>>eye(2)
ans =
     1     0
     0     1
```

- **vander**

Definição: Calcula a matriz de Vandermonde a partir de um vetor dado.

Sintaxe:

vander(A) → Calcula a matriz de Vandermonde a partir de *A*.

```
A=[1 2 3 4];
>> vander(A)
ans =
     1     1     1     1
     8     4     2     1
    27     9     3     1
    64    16     4     1
```

- **rand**

  Definição: Cria uma matriz com valores aleatórios.

  Sintaxe:

  rand(m) → Cria uma matriz *m* x *m* com valores aleatórios entre 0 e 1.

  rand(m,n) → Cria uma matriz *m* x *n* com valores aleatórios entre 0 e 1.

```
rand(2)
ans =
    0.9501    0.6068
    0.2311    0.4860
```

4.5. Propriedades de matrizes

- **’ (apóstrofo)**

  Definição Calcula a matriz transposta.

  Sintaxe:

  A’ → Gera a matriz transposta de *A*.

```
>>A=[1 1; 2 3]
>>A =
     1     1
     2     3
>> A'
ans =
     1     2
     1     3
```

- **det**

  Definição: Calcula o determinante de uma matriz.

  Sintaxe:

  det(A) → Calcula o determinante da matriz *A*.

```
>>det(A)

ans =
     1
```

- **trace**

Definição: Retorna um vetor com a soma dos elementos da diagonal principal de uma matriz.

Sintaxe:

trace(A) → Retorna a soma dos elementos da diagonal principal da matriz *A*.

```
A =
     1     4
     2     1
>> trace(A)
ans =
     2
```

- **inv**

Definição: Determina a matriz inversa dada.

Sintaxe:

inv(A) → Retorna a matriz inversa da matriz *A*.

```
>> A = [5 8; 4 9]
A =
     5     8
     4     9
>> inv(A)
ans =
    0.6923   -0.6154
   -0.3077    0.3846
```

- **eig**

Definição: Calcula os autovalores e autovetores de uma matriz.

Sintaxe:

eig(A) → Retorna os autovalores de uma matriz quadrada *A*.

[a, b] = eig(A) → Retorna em *a*, uma matriz com os autovetores e, em *b*, uma matriz com os autovalores associados.

```
>> A=[1 -1; 4 1]
```

```
A =
     1    -1
     4     1
>> [a,b]=eig(A)
a =
        0 - 0.4472i        0 + 0.4472i
  -0.8944            -0.8944
b =
   1.0000 + 2.0000i        0
        0             1.0000 - 2.0000i
```

Exercício 2- Resolva o seguinte sistema de equações lineares:

$$\begin{cases} 2x_1 + 1.5x_2 + x_3 = 13.20 \\ x_1 + 6x_2 - 2x_3 = 21.64 \\ 2x_2 + 4x_3 = 26.62 \end{cases}$$

4.6. Trabalhando com matrizes

- **size**

Definição: Retorna as dimensões de uma matriz.

Sintaxe:

[m,n] = size(A) → Retorna, em *m*, o número de linhas e, em *n*, o número de colunas da matriz *A*.

```
>> A=[1 1; 2 3];
>> [m,n]=size(A)
m =
     2
n =
     2
```

- **find**

Definição: Procura os elementos em uma matriz de tal modo a respeitar a lógica fornecida, retornando os índices que descrevem estes elementos.

Sintaxe:

ind = find(X) → Retorna os índices dos elementos não-nulos na matriz *X*.

[row,col] = find(X, ...) → Retorna, em *row*, uma matriz coluna com os índices das linhas dos elementos da matriz *X* e, em *col*, a matriz coluna contendo os índices correspondentes às colunas dos elementos da matriz *X*.

[row,col,v] = find(X, ...) → Retorna, em *row*, uma matriz coluna com os índices das linhas dos elementos da matriz *X* e, em *col*, a matriz coluna contendo os índices que descrevem as colunas dos elementos da matriz *X* e, em *v*, a matriz contendo os elementos de *X*.

```
A=[1 1; 0 3];
>> find(A)
ans =
     1
     3
     4
>> X = [3 2 0; -5 0 7; 0 0 1];
>> [r,c,v] = find(X>2);
>> [r c]
ans =
     1     1
     2     3
```

Veja no ultimo caso acima que *r* e *c* retornam em os índices das linhas e das colunas correspondentes aos elementos que respeitam a expressão oferecida.

Obviamente, os elementos a_{11} e a_{23} são os únicos maiores que 2.

- **sort**

Definição: Retorna o vetor dado ou elementos de uma matriz em ordem crescente ou decrescente.

Sintaxe:

sort(A,dim) → Retorna os elementos das colunas (*dim = 1*) ou da linha (*dim = 2*) da matriz *A* em ordem crescente.

sort(A,mode) → Retorna os elementos das colunas da matriz *A* em ordem crescente (*mode* = *'ascend'*) ou em ordem decrescente (*mode* = *'descend'*).

```
>> sort(A)
ans =
     1     1
     2     4
```

- **fliplr**

  Definição: Espelha as colunas de uma matriz.

  Sintaxe:

  fliplr(A) → Espelha as colunas da matriz *A*.

```
>> A=[1 2;3 4]
A =
     1     2
     3     4
>> fliplr(A)
ans =
     2     1
     4     3
```

- **flipud**

  Definição: Espelha as linhas de uma matriz.

  Sintaxe:

  flipud(A) → Espelha as linhas da matriz *A*.

```
>> A=[1 2;3 4]
A =
     1     2
     3     4
>> flipud(A)
ans =
     3     4
     1     2
```

Exercício 3- Crie um vetor *A* de 50 elementos aleatórios e em seguida crie a partir deste, outro vetor *B* obedecendo aos seguintes critérios:

a. Conter somente os elementos de *A* maiores que 0.5;

b. Os elementos devem de *B* estar em ordem decrescente.

Exercício 4- Realize as seguintes operações no MATLAB, a partir das matrizes dadas, e interprete o resultado.

$$A=\begin{pmatrix}5 & 8 & 6\\ 9 & 2 & 10\\ 7 & 6 & 1\end{pmatrix} \quad B=\begin{bmatrix}7\\ 1\\ 6\end{bmatrix} \quad C=\begin{pmatrix}5.5 & 8.1 & 4.9\\ 2.1 & 7.4 & 9.2\\ 1.3 & 4.5 & 3.8\end{pmatrix} \quad D=\begin{bmatrix}4 & 1 & 0\end{bmatrix}$$

a) $E=\det\left(A-\lambda I\right)\ com\ \lambda=-6$

b) $F=A^{-1}B$

c) $G=A\backslash B$

d) $A\cdot F$

e) $B^{T}\cdot C$

f) $D\cdot B$

Exemplo 1- Dado o circuito da Figura 4, calcule as tensões nos nós 1 e 2:

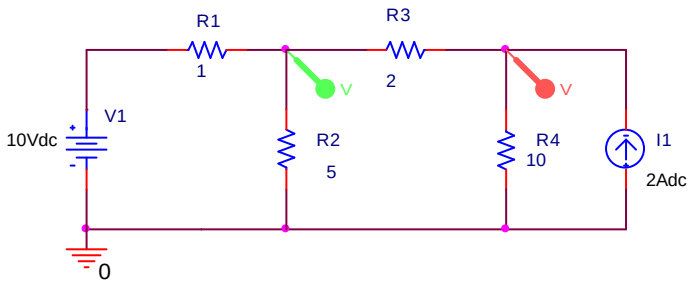


Figura 4 – Exemplo de circuito elétrico.

$$i=\frac{1}{R}\cdot v$$

$$i=G\cdot v$$

$$\begin{pmatrix}\frac{10}{1}\\ 2\end{pmatrix}=\begin{pmatrix}\frac{1}{1}+\frac{1}{5}+\frac{1}{2} & \frac{-1}{2}\\ \frac{-1}{2} & \frac{1}{2}+\frac{1}{10}\end{pmatrix}\cdot\begin{pmatrix}v_1\\ v_2\end{pmatrix}$$

$$G^{-1}\cdot i=G^{-1}\cdot G\cdot v$$

$$v=G^{-1}\cdot i$$

```
>> i=[10/1 ; 2]

i =
    10
     2

>> G=[1/1+1/5+1/2 -1/2 ; -1/2 1/2+1/10 ]

G =
    1.7000   -0.5000
   -0.5000    0.6000

>> v=inv(G)*i

v =
    9.0909
   10.9091
```

Os resultados obtidos podem ser confirmados por intermédio da Figura 5.

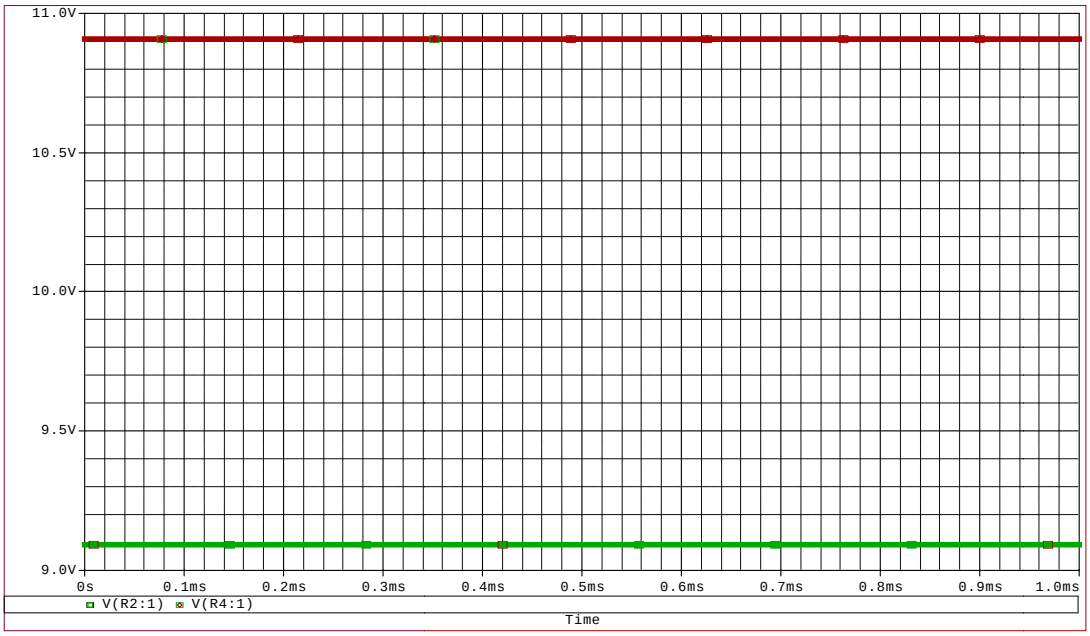


Figura 5 – Formas de onda das tensões dos nós 1 e 2.

5. VETORES

5.1. Declaração

É possível trabalhar com vetores no MATLAB, cuja representação é feita baseando-se numa matriz linha. Por exemplo, para obter o vetor *(1,3,8)*, basta iniciarmos com:

```
>> R=[1 3 8]
R =
     1     3     8
```

Portanto, todas as operações se tornam possíveis a partir do uso de funções apropriadas. É importante salientar que certas funções exigem a declaração de vetores por matriz coluna, entretanto, nada que uma consulta no *help* para ajudar.

Uma operação básica com vetores é na determinação do número de elementos, a partir da função *length*, assim como no cálculo do seu módulo, usando a função *norm*, ambas definidas abaixo. Logo depois, serão dadas algumas funções que trabalham com vetores.

- **length**

  Definição: Retorna o número de elementos que compõem o vetor.

  Sintaxe:

  length (A) → Calcula o numero de termos do vetor *A*.

```
>>A=[8 9 5 7];
>> length(A)
ans =
     4
```

- **norm**

  Definição: Retorna o módulo do vetor.

  Sintaxe:

  norm(A) → Calcula o módulo do vetor *A*.

```
>> x = [0 5 1 7];
>> sqrt(0+25+1+49)   % Forma Euclidiana
ans =
    8.6603
>> norm(x) % Usando norm
ans =
    8.6603
```

5.2. Operações

Quando se deseja calcular o produto vetorial (ou cruzado) de vetores, utiliza-se a função *cross,* apresentada a seguir:

- **cross**

  Definição: Calcula o produto vetorial entre *A* e *B*.

  Sintaxe:

C = *cross(A,B)* → Retorna, em *C*, o produto vetorial dos vetores tridimensionais *A* e *B*.

De modo análogo, define-se a função *dot* como a responsável pelo produto escalar de dois vetores dados, conforme definição a seguir.

- **dot**

  Definição: Determina o produto escalar entre dois vetores.

  Sintaxe:

C = *dot(A,B)* → Retorna, em *C*, o produto escalar dos vetores n-dimensionais *A* e *B*.

```
>> a = [1 7 3];
>> b = [5 8 6];
>> c = cross(a,b)
>> d = dot(a,b)
```

Além disso, qualquer outra operação é possível, como soma e subtração, mas se deve atentar-se ao fato de que ambos os vetores devem possuir a mesma dimensão.

Dando continuidade, serão definidas algumas funções que poderão ser úteis quando se trabalha com vetores ou até mesmo com matrizes.

- **min**

Definição: Retorna os valores mínimos de um vetor ou o das colunas de uma matriz.

Sintaxe:

min(A) → Retorna em um vetor linha os menores valores de cada linha da matriz *A*.

min(A,B) → Retorna uma matriz com os menores valores de cada posição correspondente de ambas as matrizes

[a,b]=min(A) → Retorna, em *a,* os menores valores de cada coluna e, em *b,* a posição dos mesmos nas suas respectivas colunas.

```
>> A=[1 4 5; 2 4 -2; 78 2 1]
A =
     1     4     5
     2     4    -2
    78     2     1
>> [a,b]=min(A)
a =
     1     2    -2
b =
     1     3     2
```

- **max**

Definição: Retorna os valores máximos de um vetor ou o das colunas de uma matriz.

Sintaxe:

max(A) → Retorna em um vetor linha os maiores valores de cada linha da matriz *A*.

max(A,B) →Retorna uma matriz com os maiores valores de cada posição correspondente de ambas as matrizes.

[a,b]=max(A) → Retorna, em *a*, os maiores valores de cada coluna e, em *b,* a posição dos mesmos nas suas respectivas colunas.

```
>> A=[1 4 5; 2 4 -2; 78 2 1]
A =
     1     4     5
     2     4    -2
    78     2     1

>> [a,b]=max(A)
a =
    78     4     5

b =
     3     1     1
```

- **sum**

Definição: Calcula o somatório dos elementos de um vetor ou o somatório das colunas de uma matriz.

Sintaxe:

sum(A) → Retorna a(o) soma/produto dos elementos de um vetor ou a(o) soma/produto das colunas de uma matriz

```
>> sum(A)
ans =
     3     8
```

- **prod**

Definição: Calcula o produtório dos elementos de um vetor ou o produtório das colunas de uma matriz.

Sintaxe:

prod(A) → Retorna a(o) soma/produto dos elementos do vetor *A* ou a(o) soma/produto das colunas da matriz *A*.

```
>> prod(A)
ans =
     2    16
```

Exercício 5- Os três vértices de um triângulo estão em *A (6,-1,2*), *B (-2,3,-4)* e *C (-3,1,5)*. Determine o vetor unitário perpendicular ao plano no qual o triângulo está localizado. Também determine o ângulo θ_{BAC} no vértice *A*.

Exercício 6- Declare a matriz *X* no MATLAB e determine:

$$X = \begin{bmatrix} 6 & 2 & 45 \\ 65 & 32 & 9 \\ 3 & -8 & 1 \end{bmatrix}$$

a. Um vetor *Y* que tenha o valor mínimo das três primeiras colunas;
b. A soma dos elementos de *Y*;
c. Um vetor *Z* que tenha o valor máximo das três primeiras linhas;
d. O produtório dos elementos do vetor *Z*;
e. Calcule o módulo dos elementos de cada linha.

5.3. Sistemas de Coordenadas

Existem funções, no MATLAB, que possibilitam as transformadas de coordenadas, conforme listadas a seguir:

- **cart2pol**

Definição: Converte do cartesiano para o polar/cilíndrico. Observe a Figura 6.

Sintaxe:

[theta,rho,z] = cart2pol(x,y,z) → Converte o ponto de coordenadas cartesianas *(x,y,z)* para coordenadas cilíndricas *(theta,rho,z)*.

[theta,rho] = cart2pol(x,y) → Converte o ponto de coordenadas cartesianas *(x,y)* para coordenadas polares *(theta,rho)*.

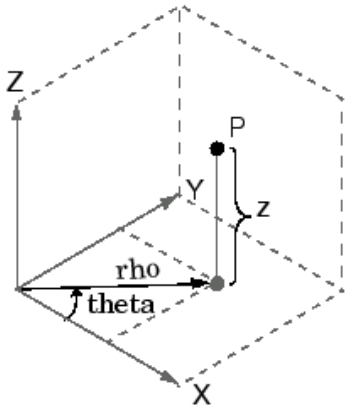


Figura 6 – Transformação entre coordenadas cartesianas e polares/cilíndricas.

- **pol2cart**

Definição: Converte do sistema de coordenadas polares/cilíndricas para o sistema cartesiano.

Sintaxe:

[x,y] = pol2cart(theta,rho) → Converte o ponto de coordenadas polares *(theta,rho)* para coordenadas cartesianas *(x,y)*.

[x,y,z] = pol2cart(theta,rho,z) → Converte o ponto de coordenadas cilíndricas *(theta,rho,z)* para coordenadas cartesianas *(x,y,z)*.

- **cart2sph**

Definição: Transforma do sistema de coordenadas cartesianas para o sistema de coordenadas esféricas. Observe a Figura 7.

Sintaxe:

[theta,phi,r] = cart2sph(x,y,z) → Converte o ponto de coordenadas cartesianas *(x,y,z)* para coordenadas esféricas *(theta,phi,r)*.

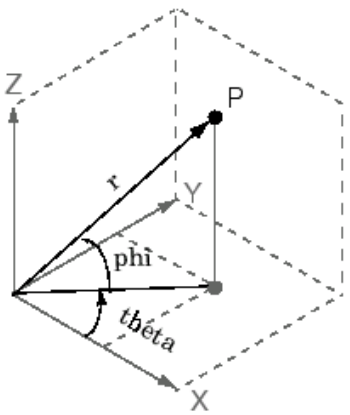


Figura 7 – Transformação entre coordenadas cartesianas e esféricas.

- **sph2cart**

Definição: Transforma do sistema de coordenadas esféricas para o sistema de coordenadas cartesianas.

Sintaxe:

[x,y,z] = sph2cart(theta,phi,r) → Converte o ponto de coordenadas esféricas *(theta,phi,r)* para cartesianas *(x,y,z)*.

Um exemplo para o uso destas funções é na utilização das equações de potenciais elétricos para determinadas distribuições de carga que são simétricos a um sistema de coordenadas. Vejamos o exemplo abaixo.

Exemplo 2- Um dipolo elétrico é formado colocando uma carga de *1nC* em *(1,0,0)* e uma outra carga de *-1nC* em *(-1,0,0)*. Determine as linhas equipotenciais geradas a partir dessa configuração.

```
>> [x,y,z] = meshgrid(-0.5:.012:0.5);
>> [teta,fi,r] = cart2sph(x,y,z);
v = (1e-9*0.2*cos(teta))./(4.*pi.*8.85e-12.*r.^2);
>> contourslice(x,y,z,v,[-0.5:0.5],[-0.5:0.5],[-0.5:0.5]);
>> colormap hsv
```

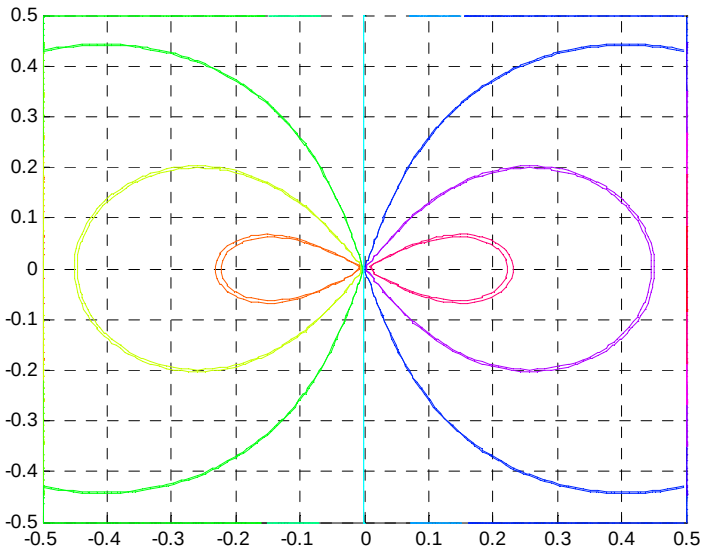


Figura 8 – Linhas equipotenciais.

6. M-FILE

6.1. Definição

O M-File é uma ferramenta do MATLAB que auxilia a criação de funções. Através dela podemos definir o nome da função, suas entradas e saídas. Para criar um novo M-file deve-se clicar em *File* → *New* → *M-File.* A declaração inicial é a seguinte:

```
function [saida1, saida2, ...] = nome(entrada1, entrada2, ...)

%declaração do código
...
```

Exemplo:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%           Função exemplo                                                    %
% A função recebe um vetor qualquer e retorna dois valores:                  %
%  vetor2 = vetor multiplicado por 2                                          %
% e v1 = o valor do primeiro elemento do vetor                               %
%                                                                            %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [vetor2, v1]= funcao(vetor)
vetor2=vetor*2;  %multiplica o vetor por 2
```

```
        v1=vetor(1);     %retorna  o  primeiro  elemento  do  vetor  de
entrada
```

Para chamar a função basta digitar na janela de comando o nome da função com as entradas entre parênteses**.** Lembrar de salvar o M-File com o mesmo nome da função!!

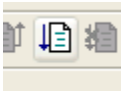
Na janela de comandos do MATLAB podemos colocar um vetor como exemplo:

```
        >> A=[2  5 -8 4 1 6]
        A =
             2     5    -8     4     1     6
        >> [x,y]=funcao(A)
        x =
             4    10   -16     8     2    12
        y =
             2
```

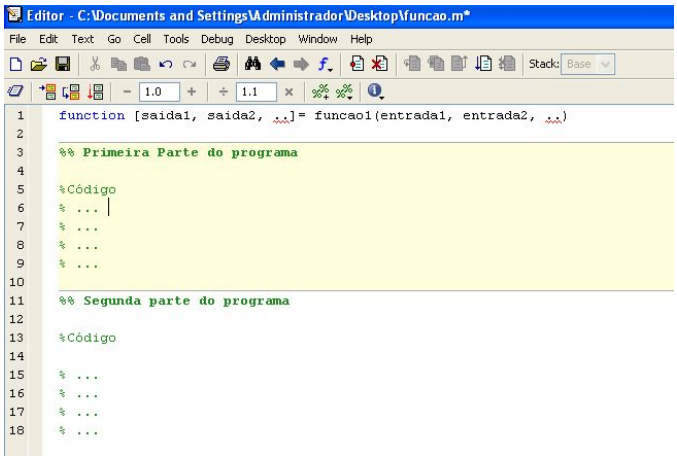
6.2. Organização

Para uma melhor organização podemos fazer comentários utilizando o símbolo '%', ou selecionando o texto inteiro e teclando *Crtl+R*, ou '%{' para abrir o comentário por bloco e '%}' para fechar.

Podemos ainda utilizar o símbolo '%%' para que, no mesmo M-File, o usuário possa rodar apenas algumas partes do programa. Para rodar somente a parte selecionada, tecle *Ctrl+Enter* e para rodar o programa inteiro clique em F5 ou em:



Exemplo:



Ao teclar *Ctrl+Enter* somente a Primeira Parte será executada

7. FUNÇÕES MATEMÁTICAS

7.1. Funções Elementares

O MATLAB contém um conjunto de funções que executam algumas funções matemáticas elementares, como módulo e raiz quadrada. A seguir disponibilizaremos uma lista de funções com uma breve descrição:

Função	Descrição
log(X)	Determina o logaritmo natural de *X*
log10(X)	Determina o logaritmo de *X* na base 10
log2(X)	Calcula o logaritmo de *X* na base 2
exp(X)	Determina a expressão de e^X
sqrt(X)	Retorna a raiz quadrada de *X*

7.2. Propriedades Fundamentais

O MATLAB possui também funções que possibilitam os cálculos elementares de matemática, tais como *mmc*, *mdc* e entre outros. Por exemplo, as funções *gcd* e *lcm* retornam o máximo divisor comum e o mínimo múltiplo comum, respectivamente. Vejamos abaixo essas e outras funções similares:

- **gcd**

  Definição: Determina o máximo divisor comum entre dois parâmetros.

  Sintaxe:

  G = gcd(A,B) → Retorna, em *G*, o máximo divisor comum entre *A* e *B*.

- **lcm**

  Definição: Determina o mínimo múltiplo comum entre dois parâmetros.

  Sintaxe:

  G = lcm(A,B) → Retorna, em *G*, o mínimo múltiplo comum entre *A* e *B*.

- **factorial**

  Definição: Retorna o fatorial de um argumento.

  Sintaxe:

  factorial(N) → Calcula o fatorial de *N*.

- **nchoosek**

  Definição: Desenvolve a fatoração binomial.**Sintaxe:**

  C = nchoosek(n,k) → Retorna o número de combinações de *n* tomada *k* a *k*.

- **primes**

  Definição: Devolve uma lista com uma quantidade desejada de números primos.

  Sintaxe:

  p = primes(n) → Devolve uma lista com *n* de números primos.

- **mod**

  Definição: Calcula a congruência entre dois argumentos.

  Sintaxe:

  M = mod(X,Y) → Retorna, em *M*, a congruência entre os argumentos *X* e *Y*.

- **rem**

  Definição: Determina o resto da divisão de dois argumentos.

  Sintaxe:

  R = rem(X,Y) → Retorna, em *R*, o resto da divisão de *X* por *Y*.

- **perms**

  Definição: Desenvolve todas as permutações possíveis dos argumentos dados.

  Sintaxe:

  P = perms(v) → *v* pode ser uma matriz com os números nos quais deseja permutá-los.

7.3. Números Complexos

O MATLAB proporciona um conjunto de funções que auxilia o manuseio de números complexos. Inicialmente, para definir um número complexo utilizam-se os operadores *i* e *j* (voltado mais para a engenharia). Por exemplo, para definir a=5+8i, faz-se:

```
>> X=2+3i
X =
   2.0000 + 3.0000i
>> Y=2+3j
Y =
   2.0000 + 3.0000i
```

Há outra forma de definir um número complexo no MATLAB, através da função *complex*. A sua vantagem está na maior liberdade que se tem para alterar a parte imaginária, real, módulo ou até mesmo a fase do número, no ponto de vista computacional. A definição dessa função é descrita como:

- **complex**

  Definição: Retorna um número complexo a partir da sua parte real e da sua parte imaginária.

  Sintaxe:

c = complex(a,b) → Retorna, em *c*, o número complexo de parte real *a* e parte imaginária *b*.

Quando se deseja trabalhar com módulo, ângulo de fase, conjugado, ou entre outros, tornam-se fáceis de serem calculados quando se utiliza a função adequada. Na Tabela 2 serão denotadas algumas funções que possibilitam isso.

Tabela 2 – Funções com números complexos.

Função	Descrição
abs(X)	Retorna o módulo do número complexo *X*
angle(X)	Retorna a fase do complexo *X*
conj(X)	Calcula o conjugado do número complexo *X*
imag(X)	Determina a parte imaginária de *X*
real(X)	Determina a parte real de *X*

7.4. Funções Trigonométricas

Quando trabalhamos com trigonometria, o MATLAB dispõe de funções que operam neste ramo matemático. Tabela 3 resume bem algumas funções que possuem este fim.

Tabela 3 – Funções trigonométricas.

Função	Descrição
cos(X)	Cosseno do argumento *X* em radianos
sin(X)	Seno do argumento *X* em radianos
tan(X)	Tangente do argumento *X* em radianos
sec(X)	Secante do argumento *X* em radianos
csc(X)	Cossecante do argumento *X* em radianos
cot(X)	Cotangente do argumento *X* em radianos

Veja acima que estas funções retornam um valor correspondente a um argumento em radianos. Quando for desejado entrar com um argumento em grau, basta utilizar o sufixo *d* em cada função. Por exemplo, o seno de 30°:

```
>> sind(30)
ans =
    0.5000
```

Entretanto, quando deseja calcular o arco correspondente a um valor pra uma dada função, basta utilizar o prefixo *a* diante as funções. Como exemplo, determinemos o arco-tangente de 1 em radianos:

```
>> atan(1)
ans =
    0.7854
```

Caso queiramos saber em grau, faríamos:

```
>> atand(1)
ans =
    45
```

Por fim, quando se deseja determinar a função hiperbólica, basta utilizar o sufixo *h* na função dada. Vejamos no comando a seguir:

```
>> cosh(3)
ans =
   10.0677
```

De fato, o resultado é coerente, pois:

```
>> (exp(3)+exp(-3))/2
ans =
   10.0677
```

A Tabela 4 resume bem o uso de sufixo e prefixo nas funções trigonométricas:

Tabela 4 – Uso de sufixo e prefixo nas funções trigonométricas.

Prefixo	Sufixo	Descrição	Exemplo
a	-	Determina o arco de um valor	*atan*
-	*d*	Determina com um argumento em graus	*cosd*
-	*h*	Determina a função hiperbólica	*sinh*

7.5. Aproximação Inteira

Na biblioteca de funções do MATLAB, há uma variedade que trabalha no intuito do arredondamento de números. Indubitavelmente, a mais importante é a *round*, que arredonda para o inteiro mais próximo. Obviamente, esta importância depende do ambiente prático no qual a função está sendo submetida. Abaixo segue uma lista de funções que tratam com aproximações numéricas.

- **round**

Definição: Arredonda os elementos de uma matriz ou de um vetor para o inteiro mais próximo desses elementos. Também é válido para números complexos.

Sintaxe:

Y = round(X) → Retorna, em *Y*, os inteiros mais próximos dos elementos de *X*.

- **ceil**

Definição: Arredonda os elementos de uma matriz ou de um vetor para o inteiro imediatamente maior que os respectivos elementos .

Sintaxe:

B = ceil(A) → Retorna, em *B*, os inteiros imediatamente maiores que os elementos de *A*.

- **floor**

Definição: Arredonda os elementos de uma matriz ou de um vetor para o inteiro imediatamente menor que os respectivos elementos.

Sintaxe:

B = floor(A) → Retorna, em *B*, os inteiros imediatamente menores que os elementos de *A*.

- **fix**

Definição: Arredonda os elementos de uma matriz ou de um vetor para o inteiro mais próximo de tal modo que esteja em direção ao zero.

Sintaxe:

B = fix(A) → Retorna, em *B*, os inteiros mais próximos, em direção ao zero, dos elementos de *A*.

8. GRÁFICOS

8.1. Gráficos Bidimensionais

- **ezplot**

Definição: Plota a expressão simbólica $f(x)$ no domínio padrão $-2\pi < x < 2\pi$. Observe a Figura 9.

Sintaxe:

ezplot(f) → Plota a expressão $f(x)$.

```
>> ezplot('sin(x)')
```

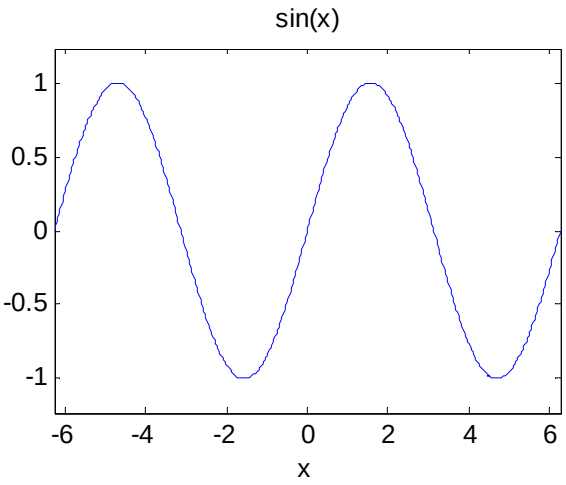


Figura 9 – Gráfico *sin(x)* gerado pela função *ezplot*.

- **plot**

Definição: Plota as colunas de um vetor *versus* os índices de cada elemento, se o vetor for real. Se for complexo, plota a parte real pela parte imaginária de cada elemento. Observe a Figura 10.

Sintaxe:

plot(X) → Se *X* for real, plota as colunas de *X* pelos índices de cada elemento.

plot(X) → Se *X* for complexo, plota a parte real pela parte imaginária de cada elemento. É equivalente a *plot(real(X),imag(X))*.

plot(X,Y) → Plota os elementos de *X* pelos de *Y*.

```
>> t = 0:pi/50:10*pi;
>> plot(t,sin(t))
```

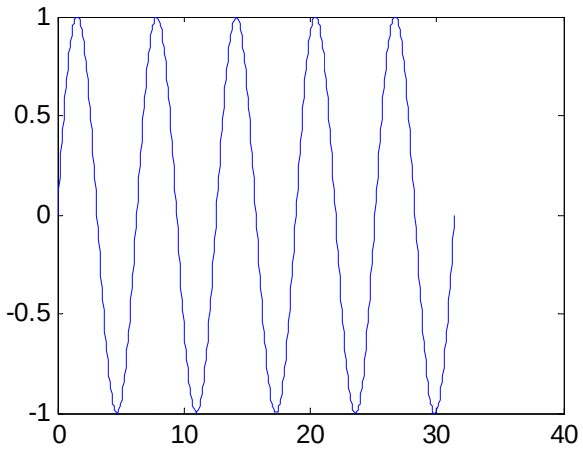


Figura 10 – Gráfico *sin(t)* gerado pela função *plot*.

- **line**

  Definição: Cria uma linha no gráfico atual. Observe a Figura 11.

  Sintaxe:

  line(X,Y) → Cria uma linha definida nos vetores *X* e *Y* no gráfico atual.

  line(X,Y,Z) → Cria uma linha no espaço tridimensional.

```
>> x=-2:0.01:5;
>> line(x,exp(x))
```

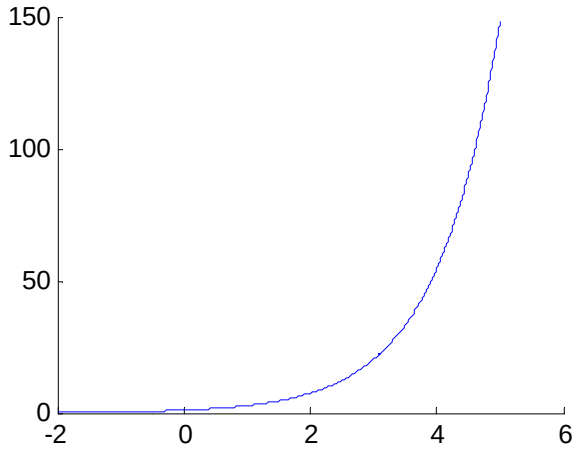


Figura 11 – Gráfico e^x gerado pela função *line*.

- **stem**

Definição: Plota uma seqüência de dados discretos. Observe a Figura 12.

Sintaxe:

stem(Y) → Plota a seqüência de dados do vetor *Y* em um domínio discreto ao longo do eixo-*x*.

stem(X,Y) → Plota *X* em função de *Y* em um domínio discreto. *X* e *Y* devem ser vetores ou matrizes de mesmo tamanho.

```
>> x=-4:4;
>> y=exp(x);
>> stem(x,y)
```

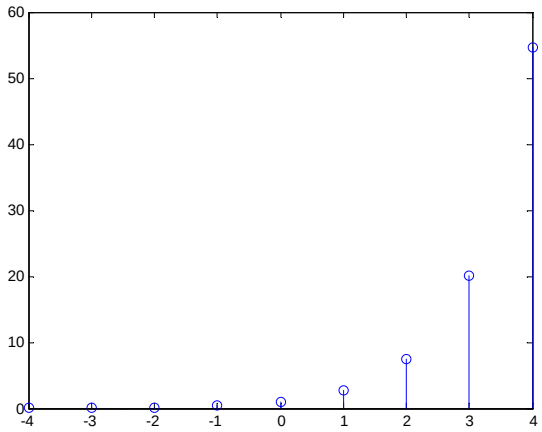


Figura 12 – Gráfico e^x gerado pela função *stem*.

- **compass**

Definição: Plota vetores de componentes cartesianas a partir da origem de um gráfico polar. Observe a Figura 13.

Sintaxe:

compass(U,V) → Plota o vetor de componentes cartesianas *U* e *V* partindo da origem do gráfico polar.

```
>> compass(2,3)
```

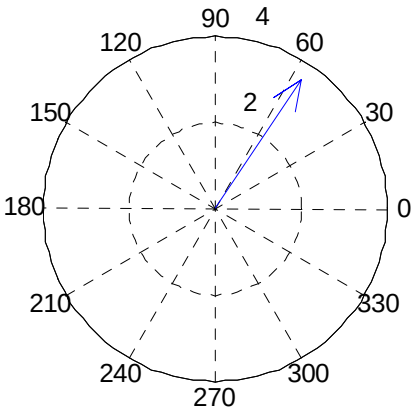


Figura 13 – Gráfico polar gerado pela função *compass*.

- **quiver**

Definição: Plota vetores de componentes cartesianas de acordo com posição e módulo definidos pelo usuário. Observe a Figura 13.

Sintaxe:

quiver(x,y,u,v,scale) → Plota um vetor com origem nos pontos *x* e *y* módulo proporcional à hipotenusa dos catetos *u* e *v*. Na variável *scale* o usuário pode definir a proporção do módulo do vetor com as variáveis *u* e *v*. Para *scale* igual a 0, a proporção será de 1:1. Veja os exemplos na Figura 14 e .

```
>> x=[1 2 3];
>> y=[3 4 7];
>> u=[-1 -1 -1];
>> v=[1 1 1];
>> quiver(x,y,u,v,0)
>> grid on
```

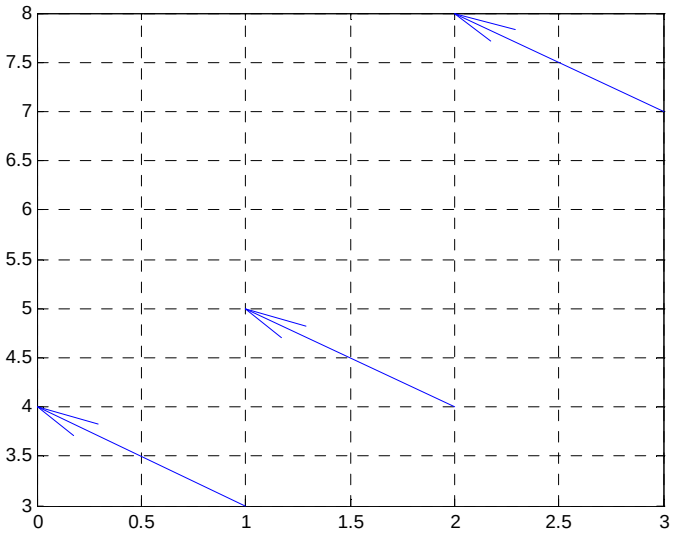


Figura 14 - Gráfico de vetores gerado pela função *quiver*

```
>> x=[1 2 3];
>> y=[3 4 7];
>> u=[-1 1 -1];
>> v=[1 -1 1];
>> quiver(x,y,u,v,0)
>> grid on
```

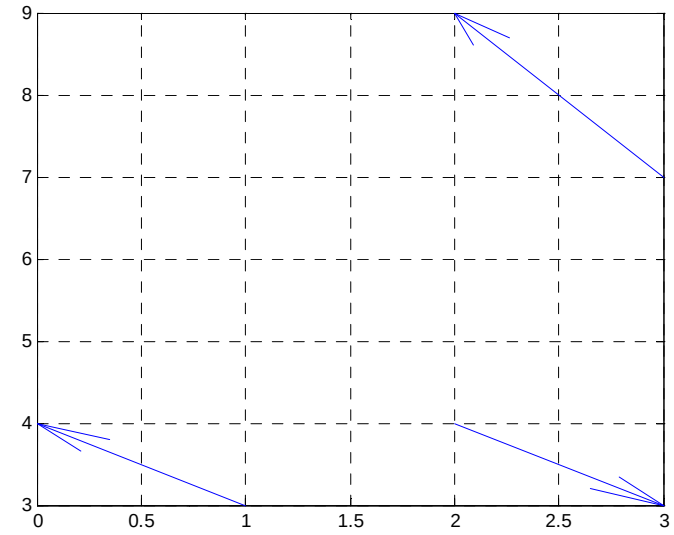


Figura 15 - Gráfico gerado pela função *quiver* com algumas variações nas variáveis dos módulos dos vetores

8.2. Gráficos Tridimensionais

- **ezplot3**

Definição: Plota uma curva espacial de três equações paramétricas no domínio padrão $0 < t < 2\pi$. Observe a Figura 16.

Sintaxe:

ezplot3(x,y,z) → Plota a curva paramétrica $x = x(t)$, $y = y(t)$ e $z = z(t)$.

```
>> ezplot3('cos(t)','sin(t)','t')
```

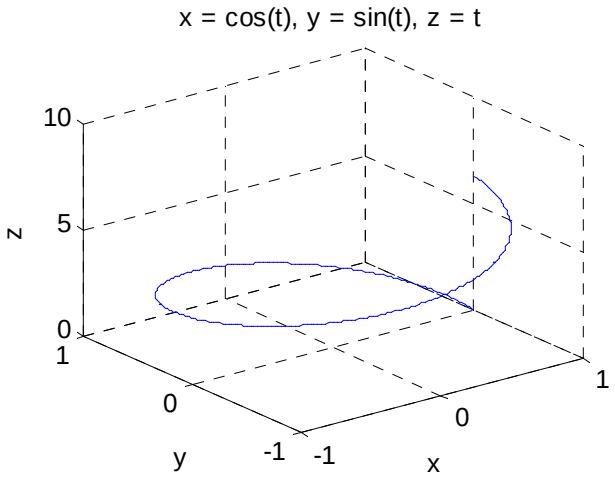


Figura 16 – Gráfico *cos(t), sin(t), t* gerado pela função *ezplot3*.

- **plot3**

Definição: Plota tridimensionalmente um gráfico. Observe a Figura 17.

Sintaxe:

plot(X,Y,Z) → Plota uma ou mais linhas no espaço tridimensional através de pontos cujas coordenadas são elementos dos vetores ou matrizes *X*,*Y* e *Z*.

```
>> t = 0:pi/50:10*pi;
>> plot3(cos(t),sin(t),t)
```

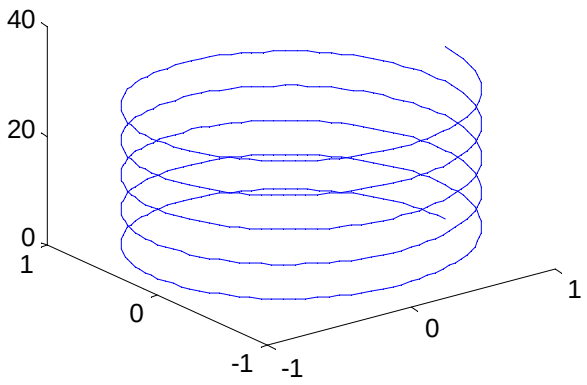


Figura 17 – Gráfico *cos(t), sin(t), t* gerado pela função *plot3*.

- **ezsurf**

Definição: Plota a superfície de um gráfico de uma função de duas variáveis no domínio padrão $-2\pi < x < 2\pi$ e $-2\pi < y < 2\pi$. Observe a Figura 18.

Sintaxe:

ezsurf(X,Y,Z) → Plota a superfície paramétrica $x = x(s,t)$, $y = y(s,t)$ e $z = z(s,t)$ no domínio $-2\pi < s < 2\pi$ e $-2\pi < t < 2\pi$.

```
>> ezsurf('1/sqrt(x^2 + y^2)')
```

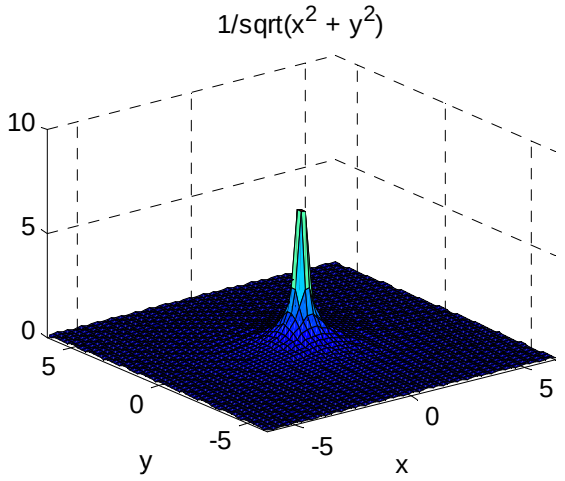


Figura 18 – Superfície $\frac{1}{\sqrt{x^2+y^2}}$ gerada pela função *ezsurf*.

- **meshgrid**

Definição: Prepara a criação de uma superfície de um gráfico tridimensional.

Sintaxe:

[X,Y] = meshgrid(x,y) → Transforma o domínio especificado pelos vetores *x* e *y* em matrizes de vetores *X* e *Y*, as quais podem ser usadas para preparar a plotagem de superfície de um gráfico tridimensional de uma função de duas variáveis.

```
>> [X,Y]=meshgrid(-6:0.1:6,-6:0.1:6);
>> Z=1./(sqrt(X.^2+Y.^2));
```

- **surf**

Definição: Plota a superfície de um gráfico de uma função de duas variáveis cujo domínio é determinado pelo usuário. Observe a Figura 19.

Sintaxe:

surf(X,Y,Z) → Plota a superfície paramétrica de *Z* em função de *X* e *Y*.

```
>> surf(X,Y,Z)
```

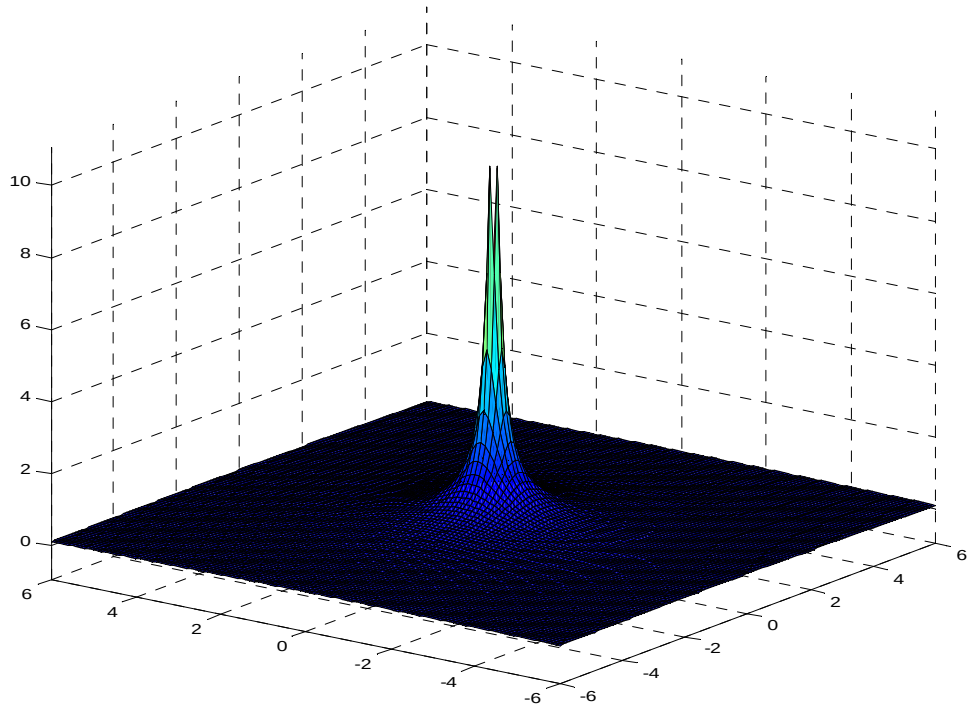


Figura 19 – Superfície $\frac{1}{\sqrt{x^2+y^2}}$ **gerada pela função *ezsurf*.**

8.3. Configuração

- **text**

Definição: Cria objetos de texto em locais específicos do gráfico. Observe a Figura 20.

Sintaxe:

text(x,y,'string') → Escreve *string* no local *(x,y)*. Pode-se modificar *string* das mais diversas formas.

```
>> plot(0:pi/20:2*pi,sin(0:pi/20:2*pi))
>> text(pi,0,' \leftarrow sin(\pi)','FontSize',18)
```

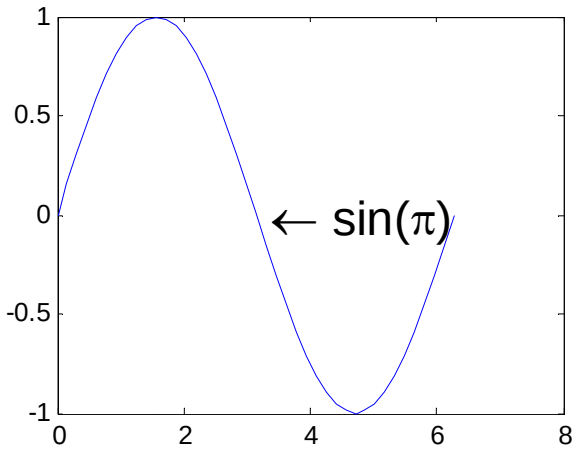


Figura 20 – Gráfico *sin(x)* gerado pela função *plot* com texto gerado pela função *text*.

- **title**

Definição: Dá um título ao gráfico. Observe a Figura 21.

Sintaxe:

title('string') → Dá o título *string* ao gráfico atual.

```
>> compass(2,3)
>> title('Gráfico Polar')
```

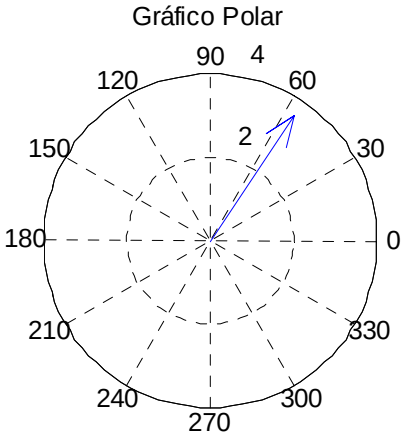


Figura 21 – Gráfico polar gerado pela função *compass* com título gerado pela função *title*.

- **axis**

Definição: Determina os limites dos eixos coordenados *X*, *Y* e *Z*. Observe a Figura 22.

Sintaxe:

axis([xmin xmax ymin ymax zmin zmax]) → define o eixo *X* de *xmin* a *xmax*, o eixo *Y* de *ymin* a *ymax* e o eixo *Z* de *zmin* a *zmax*.

```
>> t = 0:pi/50:10*pi;
>> plot3(cos(t),sin(t),t)
>> axis([-1.5 1.5 -1.5 1.5 -1 34])
```

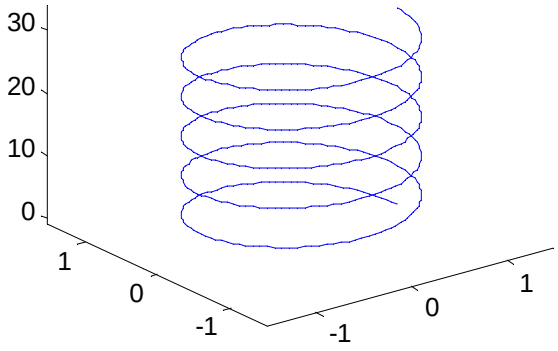


Figura 22 – Gráfico *cos(t), sin(t), t* gerado pela função *plot3* com eixos ajustados pela função *axis*.

- **grid**

Definição: Adiciona ou remove as linhas de grade em um gráfico. Observe a Figura 23.

Sintaxe:

grid on → Adiciona as linhas de grade em um gráfico.

grid off → Remove as linhas de grade em um gráfico.

```
>> t = 0:pi/50:10*pi;
>> plot3(cos(t),sin(t),t)
>> grid on
```

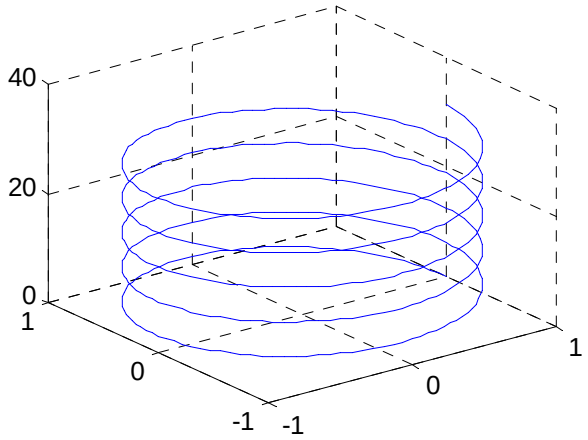


Figura 23 – Gráfico *cos(t), sin(t), t* gerado pela função *plot3* com linhas de grade geradas pela função *grid*.

- **hold**

Definição: Determina se objetos são adicionados ao gráfico ou se substituem o existente. Observe a Figura 24.

Sintaxe:

hold on → Adiciona objetos no mesmo gráfico

hold off → Substitui os objetos existentes em um gráfico pelos atuais.

```
>> x=-6:0.01:6;
>> y=sin(x);
>> plot(x,y)
>> hold on
>> t=-6:0.01:2;
>> k=exp(t);
>> plot(t,k)
```

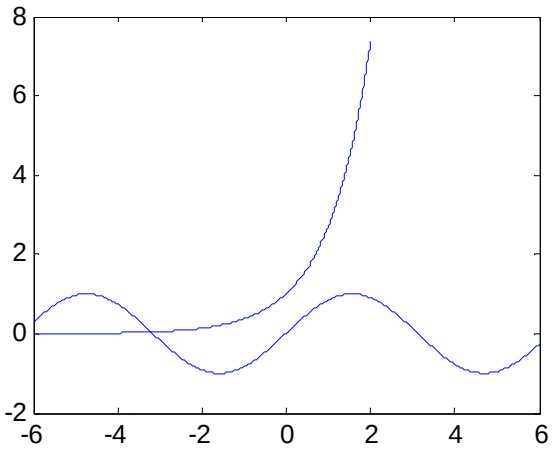


Figura 24 – Gráficos *sin(x)* e e^x gerados pela função *plot* e *ezplot* respectivamente e colocados na mesma janela de gráfico pela função *hold*.

- **legend**

Definição: Adiciona uma legenda ao gráfico. Observe a Figura 25.

Sintaxe:

legend('string1','string2') → Adiciona as legendas *string1* e *string2* ao gráfico atual.

```
>> x=-6:0.01:6;
>> y=sin(x);
>> plot(x,y)
>> hold on
>> t=-6:0.01:2;
>> k=exp(t);
>> plot(t,k,'k')
>> legend('Gráfico 1: y=sen(x)','Gráfico 2: y=exp(x)')
```

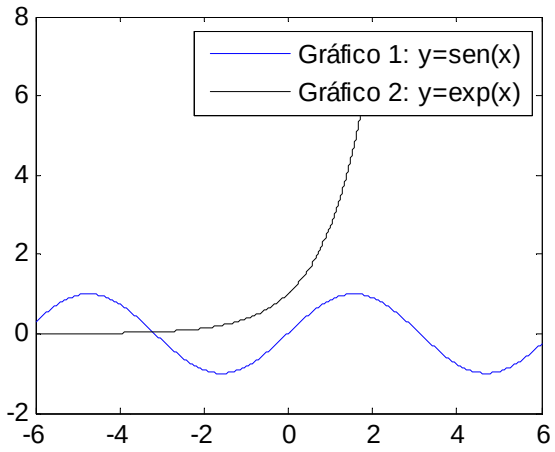


Figura 25 – Gráficos *sin(x)* e e^x gerados pela função *plot* com legenda.

- **xlabel, ylabel, zlabel**

  Definição: Dá um título aos eixos *X*, *Y* e *Z*. Observe a Figura 26.

  Sintaxe:

  xlabel('string') → Dá o título *string* ao eixo *X*.

  ylabel('string') → Dá o título *string* ao eixo *Y*.

  zlabel('string') → Dá o título *string* ao eixo *Z*.

```
>> t = 0:pi/50:10*pi;
>> plot3(cos(t),sin(t),t)
>> xlabel('x=cos(t)')
>> ylabel('y=sin(t)')
>> zlabel('z=t')
```

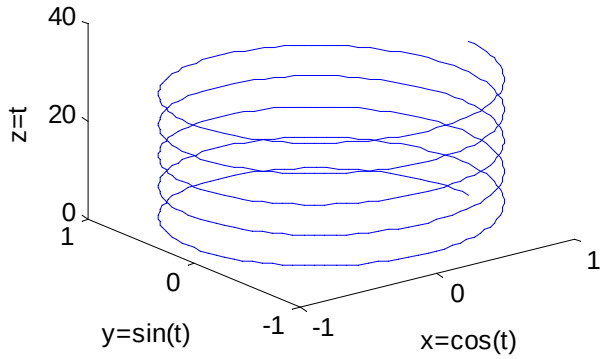


Figura 26 – Gráfico *cos(t), sin(t), t* gerado pela função *plot3* com títulos nos eixos coordenados.

- **xlim, ylim, zlim**

  Definição: Estipula os limites dos eixos *X*,*Y* e *Z*. Observe a Figura 27.

  Sintaxe:

  xlim([xmin xmax]) → define o eixo *X* de *xmin* a *xmax*.

  ylim([ymin ymax]) → define o eixo *Y* de *ymin* a *ymax*.

  zlim([zmin zmax]) → define o eixo *Z* de *zmin* a *zmax*.

```
>> t = 0:pi/50:10*pi;
>> plot3(cos(t),sin(t),t)
>> xlim([-1.5 1.5])
>> ylim([-1.5 1.5])
>> zlim([-1 34])
```

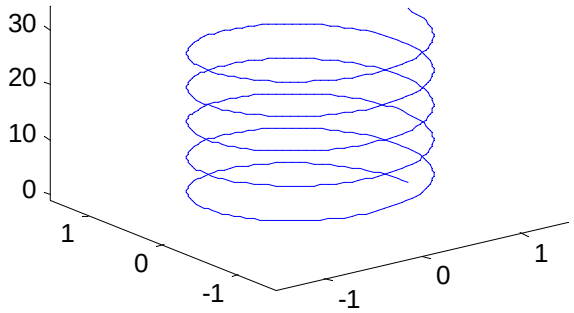


Figura 27 – Gráfico *cos(t), sin(t), t* gerado pela função *plot3* com eixos ajustados pelas funções *xlim*, *ylim* e *zlim*.

- **figure**

  Definição: Cria uma nova janela para plotar gráficos.

  Sintaxe:

  figure → Abre uma nova janela de gráfico, definindo-a como janela atual.

- **subplot**

  Definição: Divide a janela do gráfico em uma matriz definida pelo usuário, podendo trabalhar com qualquer um. Observe a Figura 28.

  Sintaxe:

h = subplot(m,n,p) (ou *subplot(mnp)*) → Divide em *m* linhas, *n* colunas, plotando o gráfico na posição *p*. Caso tenha uma matriz retangular, a contagem inicia-se no sentido anti-horário do gráfico superior esquerdo.

subplot(m,n,p,'replace') → Se o gráfico já existe, deleta o gráfico especificado, substituindo por outro gráfico desejado.

```
>> subplot(2,1,1),ezplot('sin(x)')
>> subplot(2,1,2),ezplot('exp(x)')
```

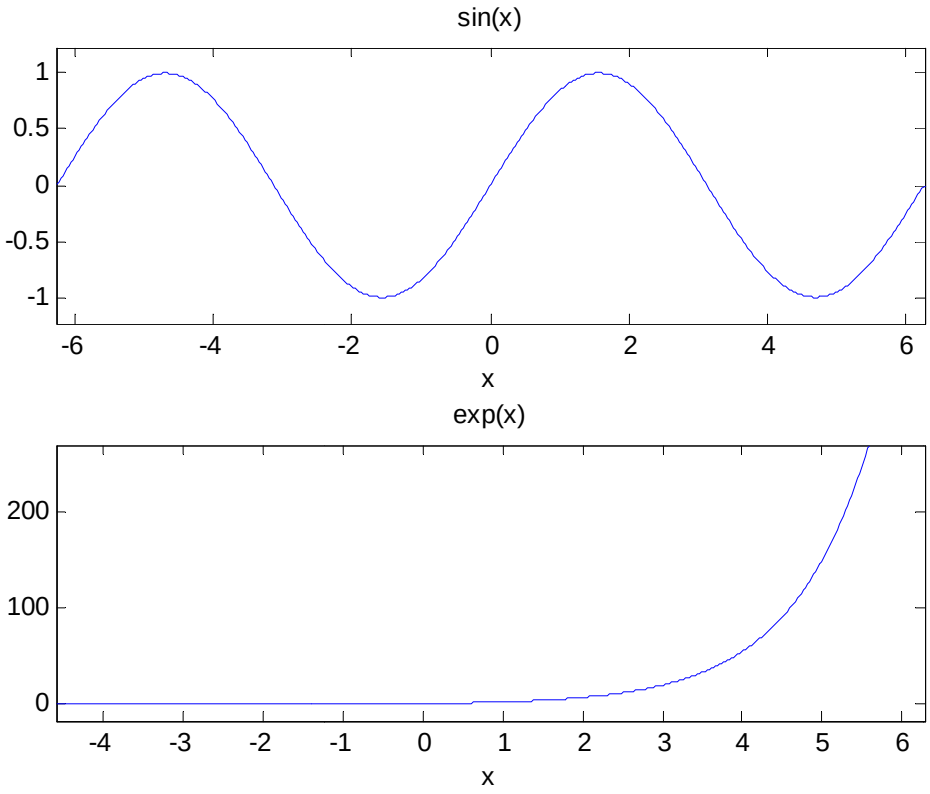


Figura 28 – Janela de gráfico dividida através da função *subplot*.

É possível configurar as propriedades do gráfico através da manipulação de uma variável. Isso é viável quando armazena a plotagem em uma variável, como no exemplo abaixo:

```
>> x=-6:0.01:6;
>> y=sin(x);
>> h=plot(x,y)
h =
  172.0143
```

As propriedades do gráfico ficam armazenadas na variável de objeto h. Assim, para configurá-lo utiliza-se a função set, de acordo com o exposto abaixo:

- **set**

  Definição: Configura as propriedades de objeto.

  Sintaxe:

  set(H,'PropertyName',PropertyValue,...) → configura o aspecto denotado por PropertyName (vide Tabela 5) para o valor especificado em PropertyValue no objeto identificador H.

Tabela 5 – Propriedades de Gráfico

Propriedade	Descrição
FontName	Especifica a fonte do texto
FontSize	Determina o tamanho da fonte
FontWeight	Altera a espessura do texto, como colocá-lo em negrito
FontAngle	Ajusta para fonte oblíqua
HorizontalAlignment	Especifica o alinhamento do texto
Color	Relacionada à cor
Rotate	Ajusta a orientação do texto
BackgroundColor	Aplica uma cor em um retângulo que envolve o texto
EdgeColor	Cor da borda desenhada em torno do texto
LineStyle	Especifica o estilo da linha
LineWidth	Determine a espessura da linha
Marker	Especifica o tipo de marcador para os pontos
MarkerEdgeColor	Ajusta a cor da borda do marcador
MarkerFaceColor	Ajusta a cor de preenchimento do marcador
MarkerSize	Especifica o tamanho do marcador

A seguir é dado um exemplo que possa contemplar o máximo de propriedades a fim de expor as suas aplicações.

```
% Fazendo a formatação do gráfico
t = 0:1/20:2;
h=plot(t,sin(2*pi*t));
set(h,'linewidth',2,'color','red','Marker','o','MarkerEdgeColor',...
    'k','MarkerFaceColor',[.49 1 .63],'MarkerSize',12)
grid on
title('Gráfico para exemplo','Fontsize',12,'Fontweight','bold',...
    'BackgroundColor',[0.49 1 0.63])
xlabel('tempo (s)','Fontweight','bold','Fontsize',12)
ylabel('f(t)=sin(2*\pi*t)','rotation',90,'Fontweight','bold',...
    'Fontsize',12)
```

Exercício 7- Esboce as funções a seguir com os respectivos comandos e de acordo com cada item:

Plot	**Ezplot**	**stem**
$5\sin(x)$	$4\sin\left(x+\frac{\pi}{3}\right)$	$3\sin(2x)$

a) Todas as funções no mesmo gráfico;

b) Cada função em uma janela diferente;

c) Todas as funções na mesma janela, mas em gráficos diferentes.

Exemplo 3- Plotar a velocidade tangencial de um projétil em lançamento oblíquo, lançamento horizontal e queda livre no espaço.

Observe o gráfico plotado do algoritmo abaixo na Figura 29.

```
      %Rotina para plotar um gráfico 3D da velocidade de um objeto
lançado ao ar

      z0=0;
      vz=50;          %velocidade
      a=-10;          %aceleração

      t=0:1:10;      %amostra de tempo
```

```
z=z0+vz*t+(a*t.^2)/2

%velocidade no eixo-x
vx=2;
x=vx*t;

%velocidade no eixo-y
vy=3;
y=vy*t;

%Cálculo dos gradientes
u=gradient(x);
v=gradient(y);
w=gradient(z);
scale=0.5;

plot3(x,y,z,'o','markerfacecolor','b');
hold on
quiver3(x,y,z,u,v,w,scale,'r')
hold off
grid on
```

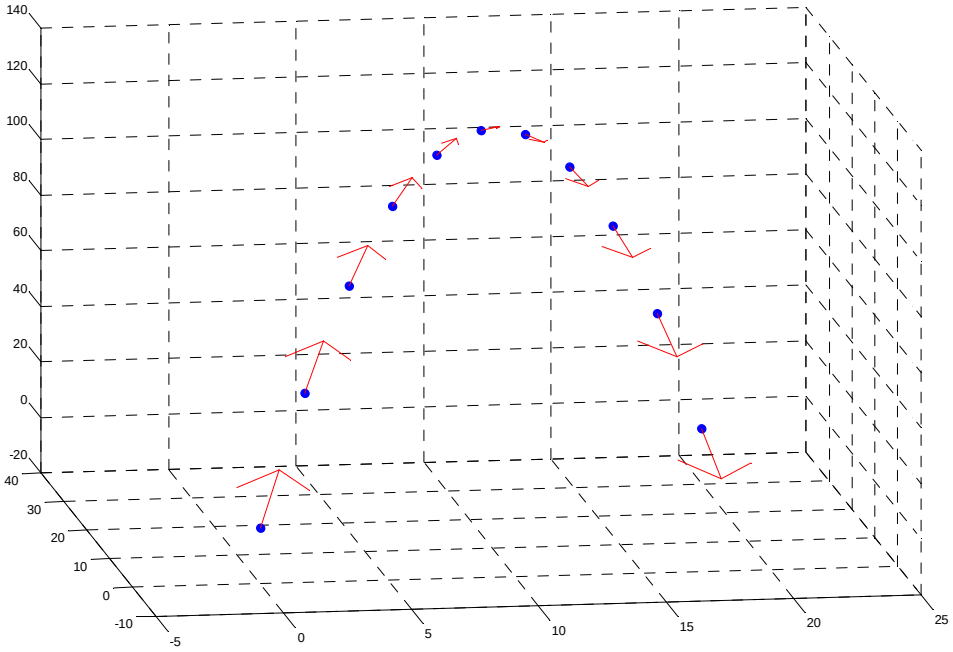


Figura 29 - Gráfico de um projétil em lançamento oblíquo

Já a Figura 30 e a Figura 31, comparam o movimento e velocidade de dois projéteis em trajetórias diferentes: lançamento horizontal e queda livre.

```
%rotina para plotar um gráfico 3D da velocidade de um objeto
lançado horizontalmente ao ar e um em queda livre

z0=0;
vz=0;          %velocidade
a=-10;          %aceleração
```

```
t=0:1:10;       %amostra de tempo
z=z0+vz*t+(a*t.^2)/2

%velocidade no eixo-x do lançamento oblíquo
vx=2;
x=vx*t;

%velocidade no eixo-y do lançamento oblíquo
vy=3;
y=vy*t;

%Cálculo dos gradientes do lançamento oblíquo
u=gradient(x);
v=gradient(y);
w=gradient(z);
scale=0.5;

%Em queda livre
xq=zeros(1,11)
yq=ones(1,11)*20
uq=gradient(xq);
vq=gradient(yq);

plot3(x,y,z,'o','markerfacecolor','b');
hold on
plot3(xq,yq,z,'o','markerfacecolor','b');
quiver3(x,y,z,u,v,w,scale,'r')
quiver3(xq,yq,z,uq,vq,w,scale,'r')
% hold off
grid on
```

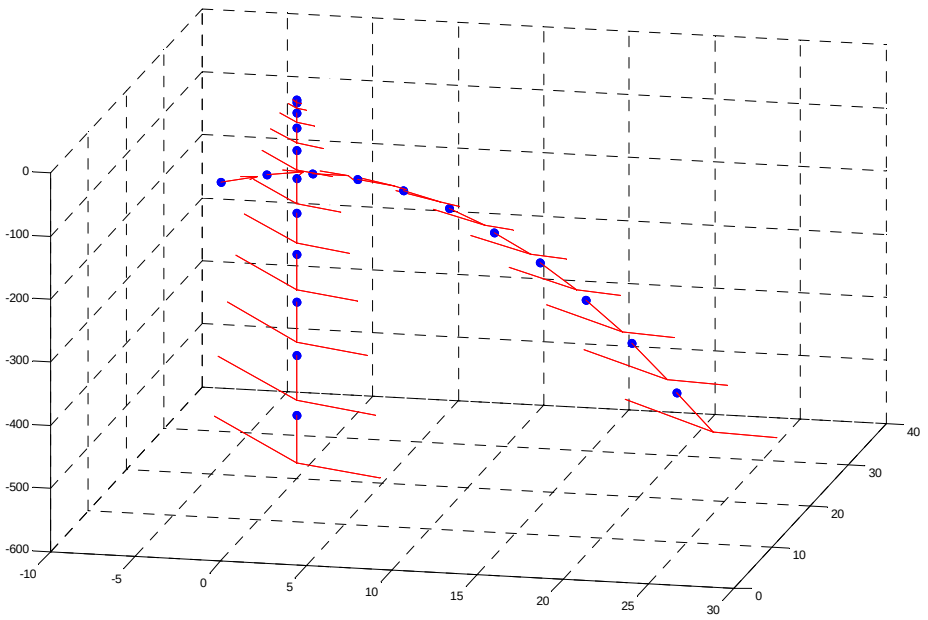


Figura 30 - Vista em 3 dimensões dos projéteis em queda livre e lançamento horizontal

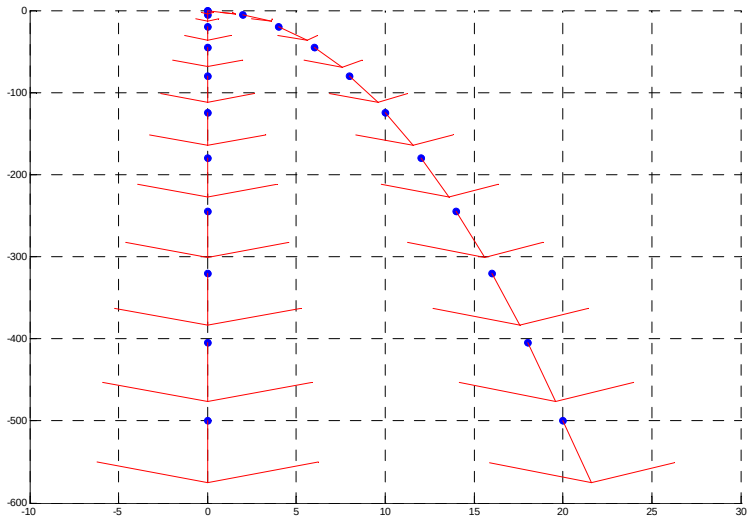


Figura 31 – Vista na dimensão X-Z dos projéteis em queda livre e lançamento horizontal

Veja que os dois projéteis estão à mesma altura e com mesma velocidade na direção *z* independente do tipo de lançamento.

Exemplo 4- Criação de arquivo em formato AVI. Observe as Figura 32 e Figura 33.

```
aviobj=avifile('Filme Seno.avi','fps',50);
hold on;
grid on;
x=-4*pi:0.1:4*pi;
for k=1:1:size(x,2)-1
    xx=[x(k) x(k+1)];
    yy=[sin(x(k)) sin(x(k+1))];
    h=plot(xx,yy);
    set(h,'EraseMode','xor');
    axis ([-10 10 -1.5 1.5]);
    frame=getframe(gca);
    aviobj=addframe(aviobj,frame);
end
aviobj=close(aviobj);
```

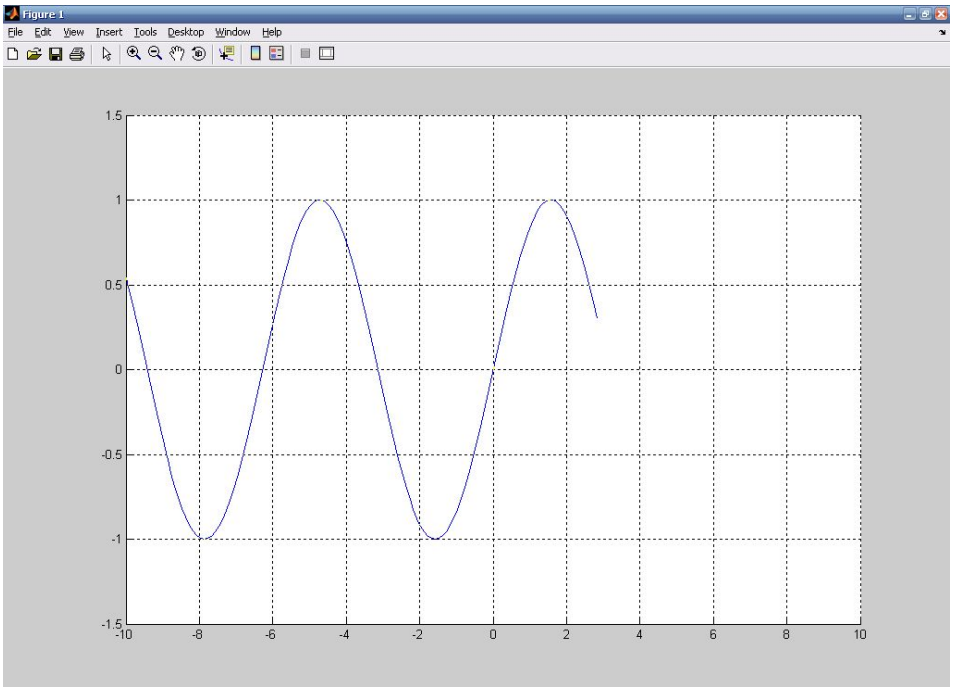


Figura 32 – Janela de criação de arquivo em formato AVI do gráfico *sin(x)*.

```
aviobj=avifile('Complexo.avi','fps',50);
hold off;
grid on;
t=0:0.01:4*pi;
x=cos(t);
y=sin(t);
for k=1:1:length(t)
    c=x(k)+i*y(k);
    h=compass(c);
    set(h,'EraseMode','xor');
    frame=getframe(gca);
    aviobj=addframe(aviobj,frame);
end
aviobj = close(aviobj);
```

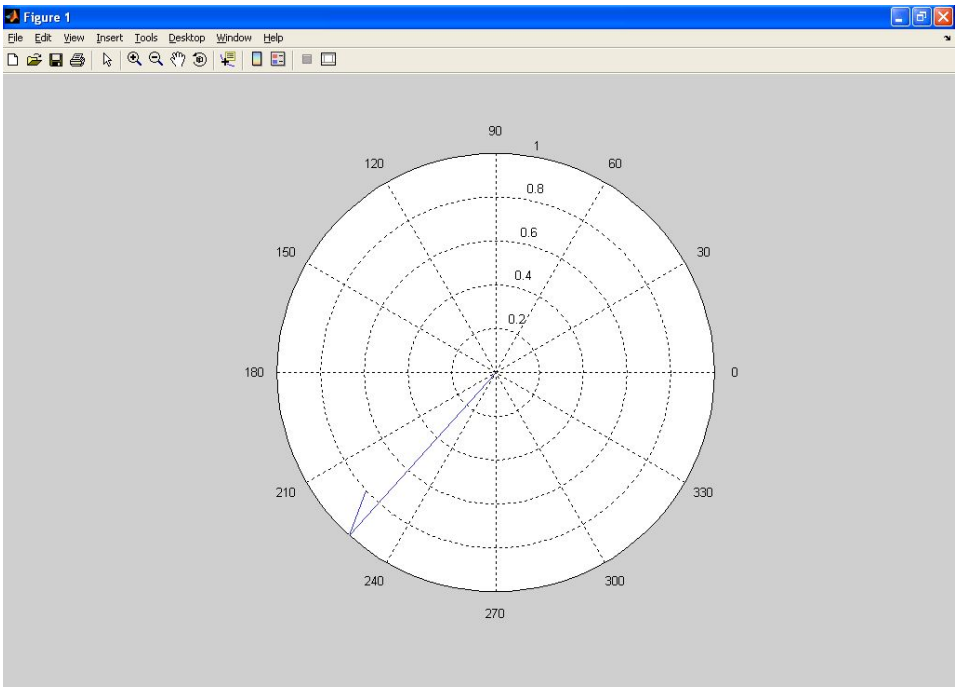


Figura 33 – Janela de criação de arquivo em formato AVI de gráfico polar.

9. MATEMÁTICA SIMBÓLICA

Há, em algumas situações, a necessidade de se trabalhar com variáveis simbolicamente, pois possibilita uma visão mais geral sobre o resultado de um problema. Neste contexto, uma função importante é a *syms*, que declara as variáveis como simbólica. Uma outra função é a *sym*, que transforma uma expressão para a forma literal. Mais detalhes dessas funções são dadas a seguir:

- **syms**

Definição: Determina que os argumentos acompanhados terão caráter simbólico .

Sintaxe:

syms arg1 arg2 ...

- **sym**

Definição: Definiçãoine variáveis, expressões e objetos como simbólicos.

Sintaxe:

S = sym(A)

x = sym('x')

Como exemplo, veja a diferença dessas duas funções executando os comandos a seguir:

```
>> rho = sym('(1 + sqrt(5))/2')
>> syms x y
>> f = x^2*y + 5*x*sqrt(y)
```

Em alguns casos, quando se desejar determinar quais as variáveis simbólicas numa expressão, usa-se a função *findsym*, que retorna os parâmetros que são simbólicos. Uma outra função é a *subs*, que substitui a variável declarada inicialmente simbólica por uma outra ou mesmo por um número. Suas Definiçãoinições estão listadas abaixo:

- **findsym**

  Definição: Determina as variáveis simbólicas em uma expressão.

  Sintaxe:

  findsym(S)

  findsym(S,n)

- **subs**

  Definição: Substituição simbólica em expressão simbólica ou em matriz.

  Sintaxe:

  R = subs(S)

  R = subs(S, new)

  R = subs(S,old,new)

  Exemplo: Dado o procedimento abaixo:

```
y=3;w=30;
syms a b n t x z
f = x^n+y; g = sin(a*t + b)-cosd(w);
```

Determine os parâmetros que são simbólicos em f e em g, assim como, de acordo com a ordem das variáveis simbólicas que aparecer, substituir todos pelo valor 2,3 para f e 4,7,1 para g, respectivamente.

Exercício 8- Dado o procedimento abaixo:

```
y=3;w=30;
syms a b n t x z
f = x^n+y; g = sin(a*t + b)-cosd(w);
```

Determine os parâmetros que são simbólicos em *f* e em *g*. Substitua, em *f*, *x* por 2 e *y* por 3, e em *g*, *a* por 4, *t* por 7 e *b* por 1.

10.OPERAÇÕES MATEMÁTICAS BÁSICAS

10.1. Expressões Numéricas

Uma curiosidade é que o MATLAB dispõe de um conjunto de funções que contribuem para a fatoração, expansão, simplificações e entre outros. O quadro abaixo resume bem cada função.

Função	Definição
collect	Reescreve a expressão como um polinômio
expand	Expande a expressão em produtos e somas
horner	Determina o fator em comum da expressão
factor	Fatora o polinômio, se os coeficientes são racionais
simplify	Simplifica as expressões, de forma mais geral.
compose	Calcula a composição das funções
finverse	Encontra a inversa funcional da função

O uso dessas funções é bastante semelhante, por exemplo, dada a expressão:

$$x\left(x\left(x-6\right)+11\right)-6$$

Para agrupá-la de tal modo que possa ter uma organização em relação ao grau do polinômio, faz-se:

```
>> syms x
>> collect(x*(x*(x-6)+11)-6)
ans =
-6+x^3-6*x^2+11*x
```

Exercício 9- Verifique a relação trigonométrica fundamental utilizando a função simplify, logo após, determine a forma expandida de $\tan(x+y)$.

Exercício 10- Dado o polinômio $f(x)=2x^2+3x-5$ e $g(x)=x^2-x+7$. Determine os seguintes polinômios: (a) (b) $f\left(g\left(x\right)\right)$

10.2. Polinômios

Agora trataremos com os polinômios. Para definir um polinômio no MATLAB, basta entrar com uma matriz linha, nos quais os elementos dela representam os coeficientes do maior para o menor grau. Por exemplo, o polinômio $5x^3-9x^2+\frac{8}{5}x+\frac{4}{7}$ é representado como p=[5 -9 8/5 4/7]. É bom lembrar que o polinômio pode ter elementos irracionais como, por exemplo, $\sqrt{2}$ ou π.

As principais funções destinadas para os polinômios são descritas a seguir.

- **poly**

Definição: Determina os coeficientes do polinômio a partir de suas raízes. Caso a entrada seja uma matriz, este calcula o polinômio característico da matriz.

Sintaxe:

p = poly(A)

p = poly(r)

```
>> y=[-2 -1]           % Declara um vetor linha [-2 -1]
y =
    -2    -1
>> z=poly(y)           %z é o polinômio (x+2)(x+1)=x²+3x+2
z =                    %que tem como raízes -2 e -1
     1     3     2
>> A=[1 5 3; 0 -2 9; 2 11 -1]
A =                    %Declara matriz
     1     5     3
     0    -2     9
     2    11    -1
>> poly(A)        %calcula o seu polinômio característico
```

```
ans =
    1.0000    2.0000 -106.0000   -5.0000
```

- **roots**

Definição: Retorna um vetor coluna com a(s) raiz(es) do polinômio fornecido.

Sintaxe:

r = roots(c)

```
>> c=[1 3 2]     % declara um vetor correspondente ao polinômio
                 % x²+3x+2
c =
     1     3     2
>> x=roots(c)    %Calcula as raízes desse polinômio, que são -2
e -1
                 %Observe a oposição entre as funções roots e poly
x =
    -2
    -1
```

- **polyval**

Definição: Determina o valor do polinômio para uma determinada entrada. Se a entrada for uma matriz, a função retorna o valor do polinômio para cada elemento.

Sintaxe:

y = polyval(p,X) → y receberá os valores do polinômio desenvolvido para cada elemento da matriz X.

```
>> polinomio=[1 5 -2 8 3.2]        %polinômio=x⁴+5x³-2x²+8x+3.2
polinomio =
    1.0000    5.0000   -2.0000    8.0000    3.2000
>> a=[1 -1; 3 2.83]
a =
    1.0000   -1.0000
    3.0000    2.8300
>> valores=polyval(polinomio,a)
valores =
```

```
          15.2000  -10.8000
         225.2000  187.2906
       %valores(1,1)= a(1,1)^4+5a(1,1)^3-2 a(1,1)^2+8 a(1,1)+3.2
```

- **polyfit**

Definição: Determina o polinômio interpolador com os pontos dados por *x* e *y* com o grau *n.* Os coeficientes são retornados numa matriz linha.

Sintaxe:

p = polyfit(x,y,n)

```
>> x=[0: 0.1: 2.5];
>> y=sqrt(x);
>> polinomio_interpolador=polyfit(x,y,3);
>> pontos_interpoladores=polyval(polinomio_interpolador,x);
>> plot(x,y,'color','r')
>> hold on
>> plot(x,pontos_interpoladores)
```

Exercício 11- São dados os pontos (1;-1), (2;-7), (5;-8) e (8;10).

(a) Determine o polinômio que interpola estes pontos;

(b) Calcule as suas raízes e o esboce em um gráfico;

(c) Destaque o ponto no qual se tem o valor do polinômio para $x = 3$.

10.3. Solução de Equações ou Sistemas

Quando você tiver um emaranhado de equações, resultando em um sistema, o MATLAB poderá ser uma ótima solução. Ao utilizar a função *solve*, você será capaz de economizar tempo e evitar resolver um tedioso sistema braçalmente. A declaração desta função segue abaixo:

- **solve**

Definição: Determina o valor do polinômio para uma determinada entrada. Quando a solução é armazenada em uma variável, o retorno é dado em uma estrutura de dados.

Sintaxe:

solve(eq) → Resolve a equação *eq*=0

solve(eq,var) → Determina as soluções de *eq=0*, em função da variável *var*.

solve(eq1,eq2,...,eqn) → Resolve um sistema de equações definidas.

g = solve(eq1,eq2,...,eqn,var1,var2,...,varn) → Calcula as soluções de um sistema de soluções em função das variáveis pré-definidas.

Partindo para um âmbito mais complexo, quando se trata de equações diferenciais, a função destinada para este caso é a *dsolve*, definida abaixo:

- **dsolve**

Definição: Soluciona simbolicamente uma equação ou sistema de equações diferenciais ordinárias.

Sintaxe:

r = dsolve('eq1,eq2,...', 'cond1,cond2,...','v')

r = dsolve('eq1','eq2',...,'cond1','cond2',...,'v')

dsolve('eq1,eq2,...','cond1,cond2,...', 'v')

Exercício 12- Eu tinha o triplo da idade que tu tinhas, quando eu tinha a idade que tu tens. Quando tu tiveres a minha idade, a diferença de nossas idades será de duas décadas. Determine nossas idades utilizando a função solve.

Exercício 13- Sabe que a aceleração de um carro em uma estrada é $a = -4x$, em que x representa a posição no instante t. Determine a posição no instante π, sabendo que este carro parte, no instante 0, do ponto 1, sendo que o motorista parou instantaneamente enquanto estava em $\frac{\pi}{2}$. Considere todas as unidades no S.I.

11. CÁLCULO DIFERENCIAL

O MATLAB disponibiliza funções que facilitam a operação de certos cálculos que são difíceis para o usuário. Por exemplo, a função *diff()*, *int()* e *limit* são algumas delas, nas quais diferenciam, integram ou calculam o limite de uma função de acordo com os parâmetros oferecidos, respectivamente. Vejamos essas e outras funções a seguir:

11.1. Limites

- **limit**

Definição: Determina o limite de uma expressão simbólica

Sintaxe:

limit(F,x,a) → calcula o limite de uma expressão simbólica *F* com *x* tendendo a *a*;

limit(F,a) → determina o limite de *F* com uma variável simbólica tendendo a *a*;

limit(F) →determina o limite com *a* = 0 como default;

limit(F,x,a,'right') → calcula o limite com *x* tendendo a *a* pela direita;

limit(F,x,a,'left') → calcula o limite com *x* tendendo a *a* pela esquerda;

Exemplo 5- Faça o seguinte limite pelo MATLAB: $\lim_{x\to 1^+} \frac{|x^2|-1}{x^2-1}$

```
>> limit('(abs(x^2)-1)/(x^2-1)',x,1,'right')
```

11.2. Diferenciação

- **diff**

Definição: Calcula a diferencial de uma função/matriz dada.

Sintaxe:

diff(S) → diferencia a expressão simbólica *S* em função de uma variável simbólica;

diff(S,'v') → diferencia *S* em torno de uma variável simbólica *v*;

diff(S,n) → diferencia, para um *n* inteiro positivo, *S* por *n* vezes;

diff(S,'v',n) → diferencia em torno de uma variável *v*, *S* por *n* vezes.

Exemplo 6- Para determinar a derivada de 1ª ordem de $f(x)=\sqrt{\ln(x)+e^x}$, faz-se:

```
>> syms x;
>>f=sqrt(log(x)+exp(x));
>> diff(f)
 ans =
 1/2/(log(x)+exp(x))^(1/2)*(1/x+exp(x))
>> pretty(ans)
```

11.3. Integração

- **int**

Definição: Calcula integral de uma função simbólica dada.

Sintaxe:

int(S) → integração indefinida a função *S* em respeito a uma variável simbólica já definida;

int(S,a,b) → integra de forma definida a função *S* de *a* a *b*;

int(S,v,a,b) → integra de *a* a *b* em função de uma variável *v*;

Exemplo 7- Dado a função $f(x)=\sqrt{x^2+5}$, calcule a integral:

- Indefinida

```
>> syms x
>> y=sqrt(x^2+5);
>> f=int(y,x)
1/2*x*(x^2+5)^(1/2)+5/2*asinh(1/5*5^(1/2)*x)
```

- Definida de 2 a 5

```
>>g = int(y,x,2,5)


5/2*30^(1/2)+5/2*log(5^(1/2)+6^(1/2))-3-5/4*log(5)
```

Caso a visualização de f não seja satisfatória, usa-se a função *pretty*, que transforma a saída de acordo com a representação matemática, conforme ilustra abaixo:

```
>> pretty(f)
```

Como a integral é calculada de forma simbólica, não é calculado o valor numérico da função *g*. Entretanto, o MATLAB não deixa a desejar neste ponto, como ao utilizar a função *eval*. Por exemplo, fazendo a instrução abaixo, você obtém:

```
>> eval(g)
```

11.4. Integrais definidas pela Regra Trapezoidal

É um método utilizado quando a área sob uma curva é representada por trapézios entre um intervalo *[a,b]* pré-definido, sendo o número de divisões *n*. Em representação matemática, tem-se:

$$\int_a^b f(x)dx = \frac{b-a}{2n}\left(f(x_0) + 2f(x_1) + ... + 2f(x_{n-1}) + f(x_n)\right)$$

em que x_i representa o ponto no final de cada trapézio, sendo $x_0 = a$ e $x_n = b$. No MATLAB a função que possibilita a partir deste método é o *trapz*, definida abaixo:

- **trapz**

Definição: Determina a integração de uma função a partir da Regra do Trapézio.

Sintaxe:

Z = trapz(Y)

Z = trapz(X,Y)

11.5. Integrais definidas pela Regra de Simpson

O método de Simpson é baseado, dado três pontos sobre a curva da função, na aproximação desses pontos em uma parábola. Então, tomados *n* subintervalos, onde *n* é par, e cuja extremidade da curva é delimitada por $f(a)$ e por $f(b)$, logo, a integral de uma função $f(x)$ é denotada por:

$$\int_a^b f(x)dx \approx \frac{b-a}{3n}\left[f(x_0)+4f(x_1)+2f(x_2)+4f(x_3)+...+2f(x_{n-2})+4f(x_{n-1})+ f(x_n)\right]$$

A maioria das calculadoras programadas utiliza esta é a regra, que é mais utilizada em termos computacionais. No MATLAB, a função encarrega para esse fim é a *quad*, mostrada abaixo:

- **quad**

Definição: Determina a integração de uma função a partir da Regra de Simpson.

Sintaxe:

q = quad(fun,a,b)

q = quad(fun,a,b,tool) à tool corresponde ao erro que a integral retornará, sendo o default de 10 3 - .

É bom destacar que fun deve ser uma função do tipo arquivo.m. Por exemplo, para calcular a integral de $y = e^{-x^2}$ no intervalo de 0 a 3, faz-se o seguinte:

Primeiro se cria o arquivo.m correspondente a função que deseja integrar, ou seja:

```
function y=funcao(x)
y=exp(-x^2);
```

Em seguida, basta utilizar a função quad, conforme modelo abaixo:

```
>> quad('funcao',0,3)
ans =
0.8862
```

11.6. Integração Dupla

O MATLAB possui a função *dblquad* que determina a integral dupla de uma função, conforme definição abaixo:

- **dblquad**

  Definição: Determina a integração dupla de uma função.

  Sintaxe:

  q = dblquad(fun,xmin,xmax,ymin,ymax)

  q = dblquad(fun,xmin,xmax,ymin,ymax,tol) → *tol* é a precisão que deseja calcular, sendo o default 10^{-6}.

11.7. Integração Tripla

Também é possível calcular a integral tripla de uma função no MATLAB, utilizando neste caso a função *triplequad*, cuja definição segue abaixo:

- **triplequad**

  Definição: Determina a integração tripla de uma função.

  Sintaxe:

  triplequad(fun,xmin,xmax,ymin,ymax,zmin,zmax)

  triplequad(fun,xmin,xmax,ymin,ymax,zmin,zmax,tol)→ *tol* é a precisão que deseja calcular, sendo o default 10^{-6}.

11.8. Outras funções

- **gradient**

  Definição: Determina o gradiente numericamente.

  Sintaxe:

  FX = gradient(F) → retorna o coeficiente do gradiente de *F* em relação a *∂x*(default).

  [FX,FY,FZ,...] = gradient(F) → retorna a matriz com os valores de *∂x, ∂*y, *∂*z..., respectivamente.

- **divergence**

Definição: Determina o divergente de um campo vetorial.

Sintaxe:

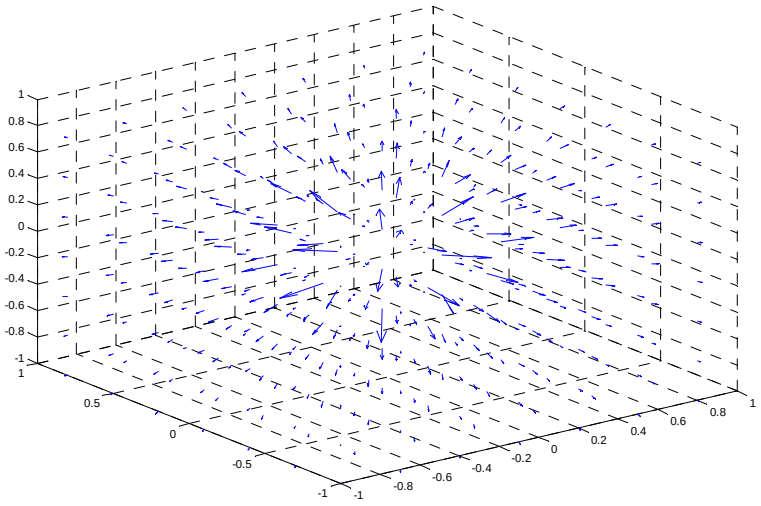
div = divergence(X,Y,Z,U,V,W) → determina o divergente do campo vetorial 3D *U*, *V* e *W*. *X*, *Y* e *Z* definem os limites de *U*, *V* e *W*, respectivamente.

div = divergence(X,Y,U,V) → calcula agora para 2D.

div = divergence(U,V,W) → usa como default para *X*, *Y* e *Z* o valor de *meshgrid(1:n,1:m,1:p).*

Exemplo 8- Um exemplo clássico para o uso de gradiente seria na determinação do Campo Elétrico devido ao efeito de uma carga pontual de 18ηC. O código segue abaixo.

```
>> [x,y,z]=meshgrid(-1:0.3:1);
>> V=(18e-12)./(4.*pi.*8.85e-12.*sqrt(x.^2+y.^2+z.^2));
>> [px,py,pz]=gradient(V,0.3,0.3,0.3);
>> Ex=(-1).*px;
>> Ey=(-1).*py;
>> Ez=(-1).*pz;
>> quiver3(x,y,z,Ex,Ey,Ez)
>> axis([-1 1 -1 1 -1 1])
```



Veja que a função *quiver* plota o que representaria o campo elétrico devido à carga.

12.SÉRIES NUMÉRICAS

12.1. Somatório

Uma função muito utilizada em Séries Numéricas é a *symsum*, que encontra o somatório simbólico de uma expressão. A sua descrição segue abaixo:

- **symsum**

  Definição: Determina o somatório simbólico de uma expressão.

  Sintaxe:

  r = symsum(s) → encontra o somatório da função *s* em função de uma variável simbólica pré-definida.

  r = symsum(s,v) → fornece o somatório em função da variável *v*.

  r = symsum(s,a,b) → determina o somatório de s variando a incógnita de a até b.

  r = symsum(s,v,a,b) → determina o somatório de s variando a incógnita *v* de a até b.

Exercício 14- Determine os seguintes somatórios:

a) $\sum_{0}^{x-1} x^2$

b) $\sum_{1}^{\infty} \frac{1}{(2n-1)^2}$

12.2. Série de Taylor

A Série de Taylor é definida como sendo:

$$P(x) = \sum_{n=0}^{\infty} (x - x_0)^n \frac{f^{(n)}(x_0)}{n!}$$

Considerando $P(x)$ como sendo o polinômio de Taylor, de ordem n, em torno do ponto x_0, então $P(x)$ é o único polinômio de grau no máximo n

que aproxima localmente f em volta de x_0 de modo que o erro $E(x)$ tenda a zero mais rapidamente que $(x-x_0)^n$, quando $x \to x_0$.

O MATLAB dispõe da função *taylor*, conforme pode ser visto abaixo:

- **taylor**

Definição: Expande em série de Taylor.

Sintaxe:

taylor(f) → faz a aproximação pelo polinômio de Taylor até a quinta ordem para a função simbólica f.

taylor(f,n,v) → retorna o polinômio de Taylor para a função simbólica f até o grau n-1 para a variável especificada por v.

taylor(f,n,v,a) → retorna a aproximação de Taylor de f em torno do ponto a, que pode ser simbólica ou um valor numérico.

Por exemplo, calculando o polinômio de Taylor para a função $f(x) = \frac{1}{5+4\cdot\cos(x)}$, tem-se:

```
>> syms x
>> f = 1/(5+4*cos(x))
f =
1/(5+4*cos(x))
 >> T = taylor(f,8)
T =
1/9+2/81*x^2+5/1458*x^4+49/131220*x^6
>> pretty(T)
```

Exercício 15- Determine, pelo polinômio de Taylor de ordem 2, o valor aproximado de $\sqrt[3]{8,2}$.

13.PROGRAMANDO EM MATLAB

O MATLAB também oferece um ambiente para a programação assim como a linguagem C. As funções são bem parecidas, modificando apenas a forma de declará-la.

- **if**

Definição: Operação condicional. Executa as funções contidas no comando. Pode ser utilizado com **else**, que executa caso a condição declarada for falsa, e com **elseif**, que executa a função caso outra condição posteriormente declarada for verdadeira.

Para mais de uma condição, utiliza-se para “e”, &&, e para “ou”, ||.

Sintaxe:

```
if  <condição>
        [Comandos]
  elseif  <condição>
        [Comandos]
  else
        [Comandos]
  end
```

- **for**

Definição: Comando de iteração. Permite que um conjunto de instruções seja executado até que a condição seja satisfeita.

Sintaxe:

```
for <condição>
[Comandos]
end
```

- **while**

Definição: Comando de iteração. Executa um bloco de instruções enquanto a condição for verdadeira.

Sintaxe:

while <condição>
[Comandos]
end

- **switch**

Definição: Operação condicional. Testa sucessivamente o valor da expressão dada e direciona para o caso especificado. Funciona como um bloco de “if’s”

Sintaxe:

switch <condição>
case caso1
[Comandos]
case {caso1, caso2, caso3, ...}
[Comandos]
otherwise (Caso não seja nenhuma das outras condições)
[Comandos]
end

```
a = 3;
switch a
   case {2}
      disp('Resposta um')
   case {3}
      disp(' Resposta dois')
   case '5'
      disp(' Resposta tres')
   otherwise
      disp('Resposta ?')
end
```

- **disp**

Definição: “Escreve” no *command window* um texto ou o valor de um vetor, sem escrever seu nome.

Sintaxe:

disp(x)

- **input**

  Definição: Pede uma entrada do usuário pelo *command window*.

  Sintaxe:

  entrada = input(‘O que deseja?’)

```
X = input('Entre um número\n')
num = 10*X
```

Command Window:

```
Entre um número
23
X =
    23
num =
   230
```

- **break**

  Definição: Quebra um laço **for** ou **while**.

  Sintaxe:

  break

```
for i = 0:5
    if i == 1
        break
    end
    i = i + 1
end
```

Command Window:

```
i =
     0
i =
     1
```

- **continue**

  Definição: Passa para o próximo laço de um **for** ou **while**.

  Sintaxe:

  continue

```
for i = 0:3
    if i == 1 && i == 2
        continue
    end
    i = i + 1
end
```

Command Window:

```
i =
     1
i =
     4
```

- **Operadores Lógicos**

  Definição: Operadores lógicos

Entradas		and	or	not	Xor
A	*B*	A & B	A | B	~A	Xor(A,B)
0	*0*	0	0	1	0
0	*1*	0	1	1	1
1	*0*	0	1	0	1
1	*1*	1	1	0	0

Exercício 16- Utilizando matemática simbólica, crie um programa que calcule as raízes de uma dada função através dos métodos abaixo. A entrada deve conter uma função simbólica e a precisão da raiz.

a) Bissecção
b) Newton-Raphson

13.1. Verificação de Erros

É muito comum que depois de implementado, um programa apresente erros de lógica ou de sintaxe. O MATLAB, como em compiladores C, também apresenta uma ferramenta de “Debug”. Com essa ferramenta é possível se executar o código do programa passo a passo, de modo a tornar mais fácil a localização e de um erro e a sua correção.

Abaixo segue alguns procedimentos para se Debugar:

1. Primeiro se adiciona break points, clicando com o botão direito do “mouse“ nas linhas em que temos interesse de começar a verificação, e escolhendo a opção “Set/Clear Breakpoint”. Quando utilizamos um “breakpoint” numa linha, surge um ponto vermelho à sua esquerda.
2. Depois de se criar os “breakpoints” pode-se chamar o m.file digitando o seu nome no “workspace”
3. O programa será rodado até que o primeiro “breakpoint” seja encontrado. A partir daí pode-se pressionar F10 para a execução da linha seguinte do programa.
4. Caso se queira pular para o próximo “breakpoint”, basta pressionar F5

echo on/off → Mostra o código do M-file que está sendo executado no momento.

14. ANÁLISE DE SINAIS

14.1. Transformação de Variável Independente

$x(t) \rightarrow x(\alpha t + \beta)$

$|\alpha| < 1$ → Expansão

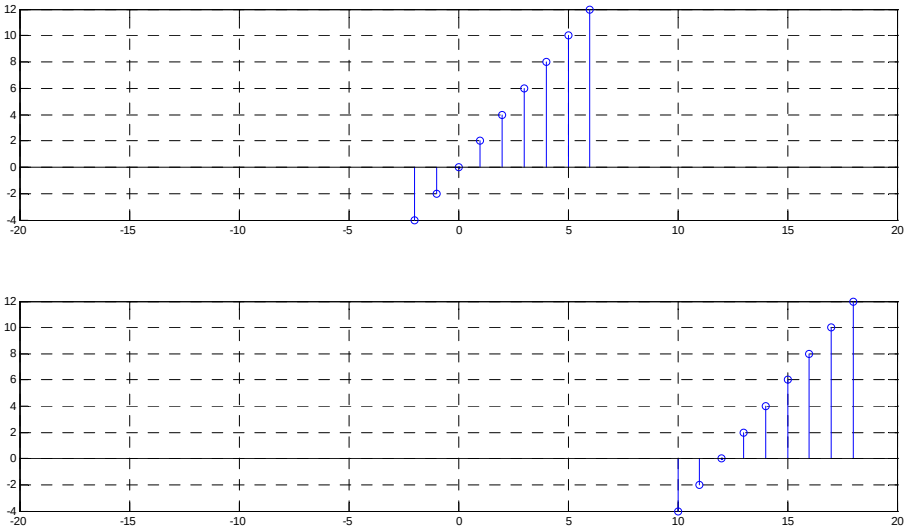
$|\alpha| > 1$ → Compressão

$\alpha < 0$ → Reflexão

$\beta \neq 0$ → Deslocamento

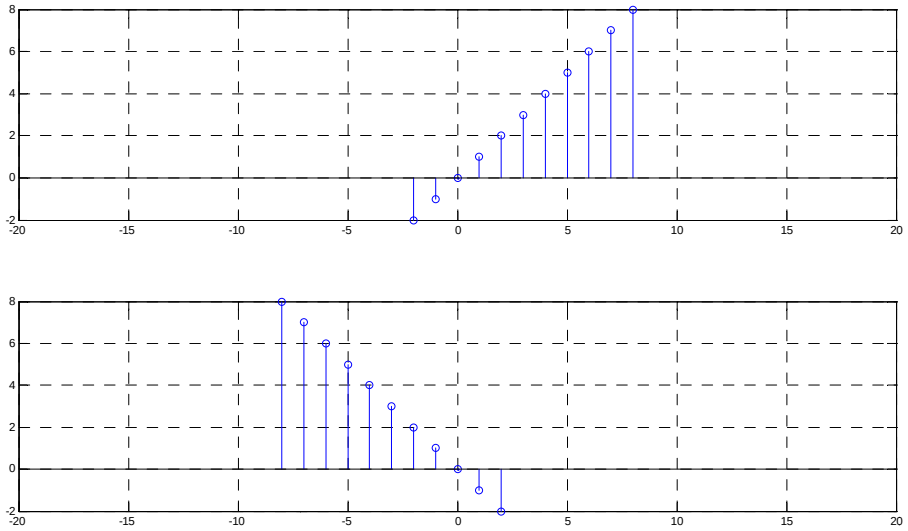
- Deslocamento no tempo:

```
clear, clc, clf
x=-2:6;
y=2*x;
n0=input('n0= ');
subplot(2,1,1),stem(x,y), grid on, xlim([-20 20])
hold on
xnovo=x+n0;
subplot(2,1,2),stem(xnovo,y);grid on, xlim([-20 20])
hold off
```



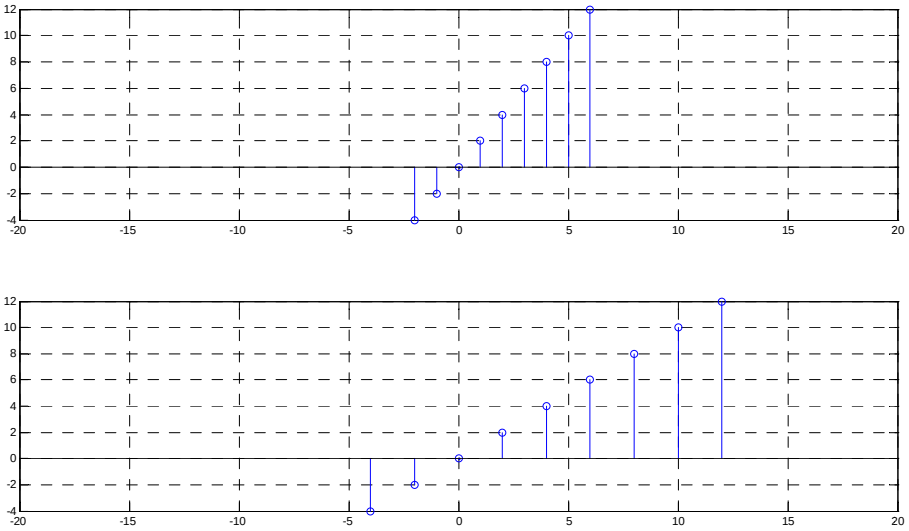
- Reflexão

```
clear, clc, clf
x=-2:8;
y=x;
subplot(2,1,1),stem(x,y), grid on, xlim([-20 20])
hold on
xl=-x;
subplot(2,1,2),stem(xl,y);grid on, xlim([-20 20])
hold off
```



- Escalonamento

```
clear, clc, clf
x=-2:6;
y=2*x;
a=input('a= ')
subplot(2,1,1),stem(x,y), grid on, xlim([-20 20])
hold on
xl=x/a;
subplot(2,1,2),stem(xl,y);grid on, xlim([-20 20])
hold off
```



14.2. Funções Pré-definidas

- Impulso:

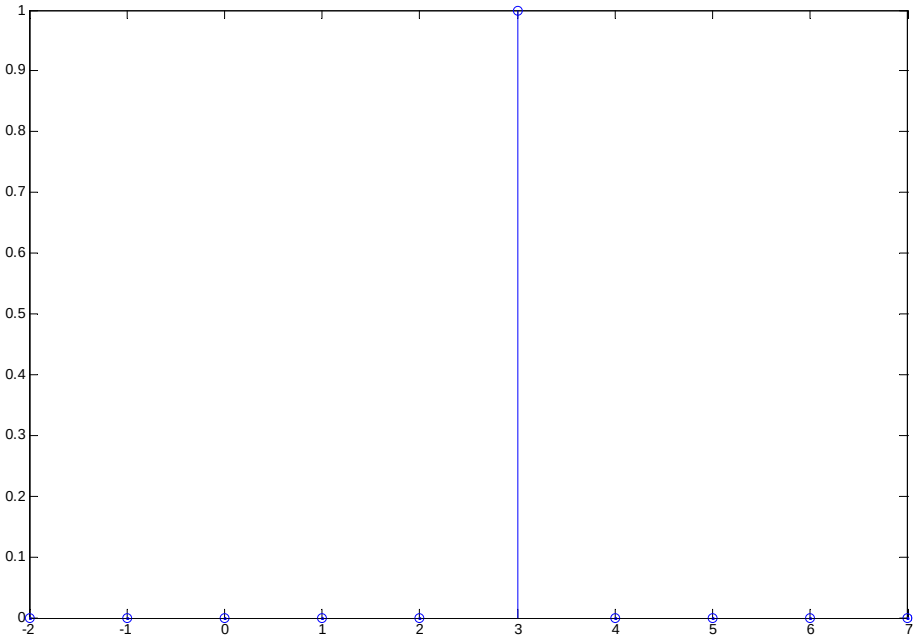
$$\delta[n] = \begin{cases} 0, & n \neq 0 \\ 1, & n = 0 \end{cases}$$

$$\delta(t) = \begin{cases} 0, & t \neq 0 \\ 1, & t = 0 \end{cases}$$

```
function [u] = impulso(n,N)
for k=1:length(n)
    if n(k)~=N
        u(k)=0;
    else
        u(k)=1;
    end
end
```

Command Window:

```
>> x=-2:7;
>> y=impulso(x,3);
>> stem(x,y)
```



- Degrau

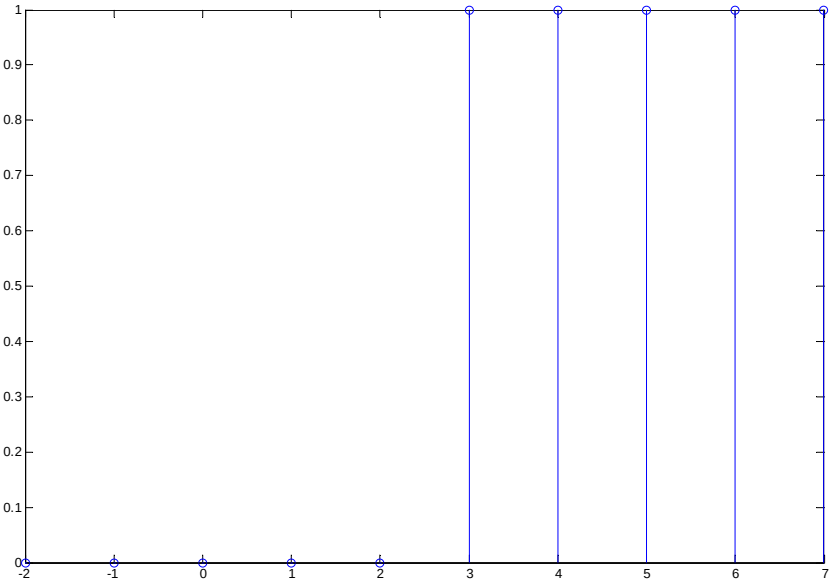
$$u[n]=\begin{cases}0, & n<0\\ 1, & n\geq 0\end{cases}$$

$$u(t)=\begin{cases}0, & t<0\\ 1, & t\geq 0\end{cases}$$

```
function [u] = degrau(n,N)
for k=1:length(n)
    if n(k)<N
        u(k)=0;
    else
        u(k)=1;
    end
end
```

Command Window:

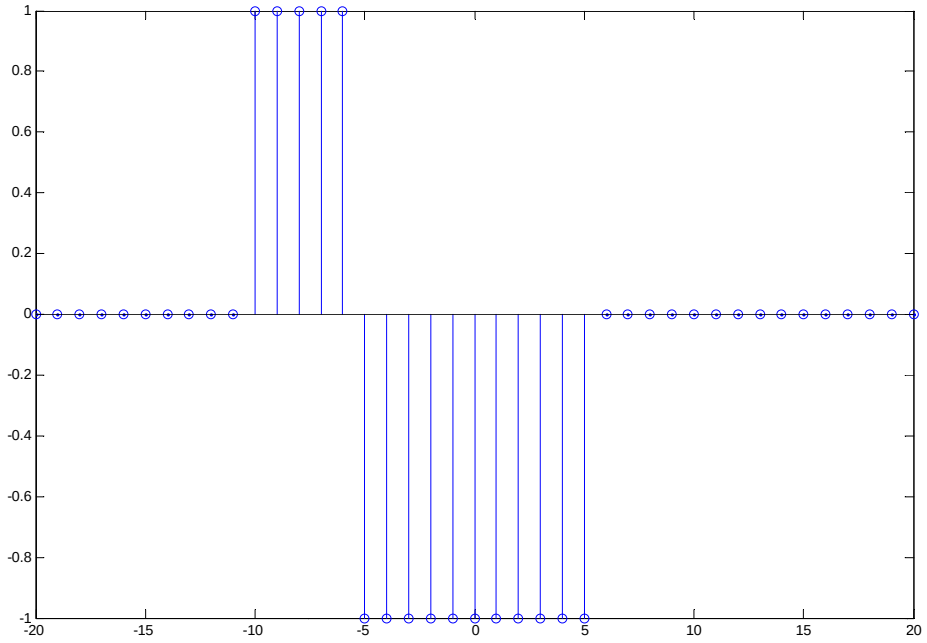
```
>> n=-2:7;
>> y=degrau(n,3);
>> stem(n,y)
```



Exercício: Determine:

a) $y[n] = u[n+10] - 2u[n+5] + u[n-6]$

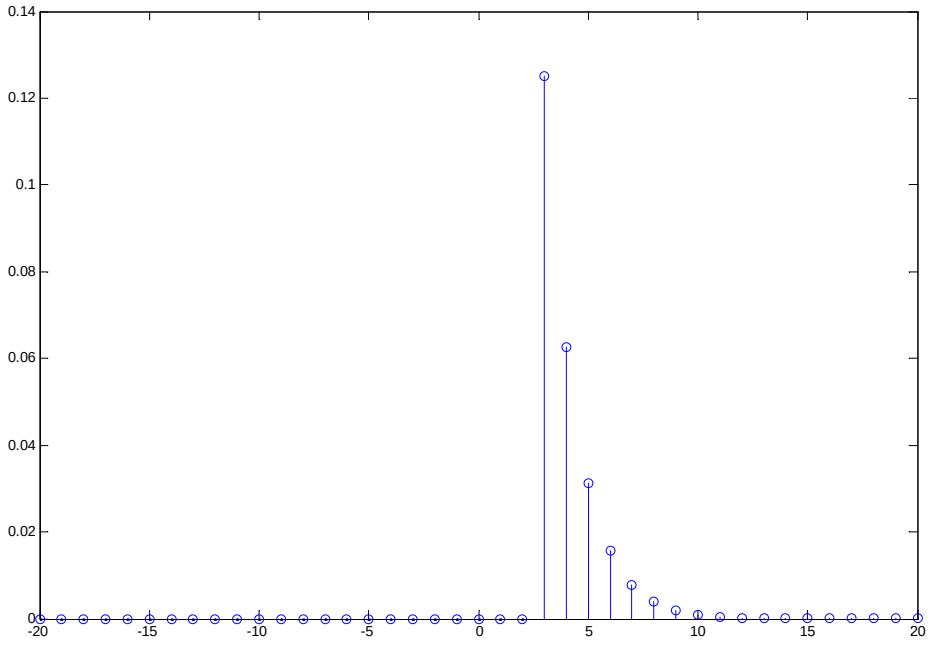
```
>> n=-20:20;
>> y=degrau(n,-10)-2*degrau(n,-5)+degrau(n,6);
>> stem(n,y)
```



b) $y[n] = \left(\frac{1}{2}\right)^n u[n-3]$

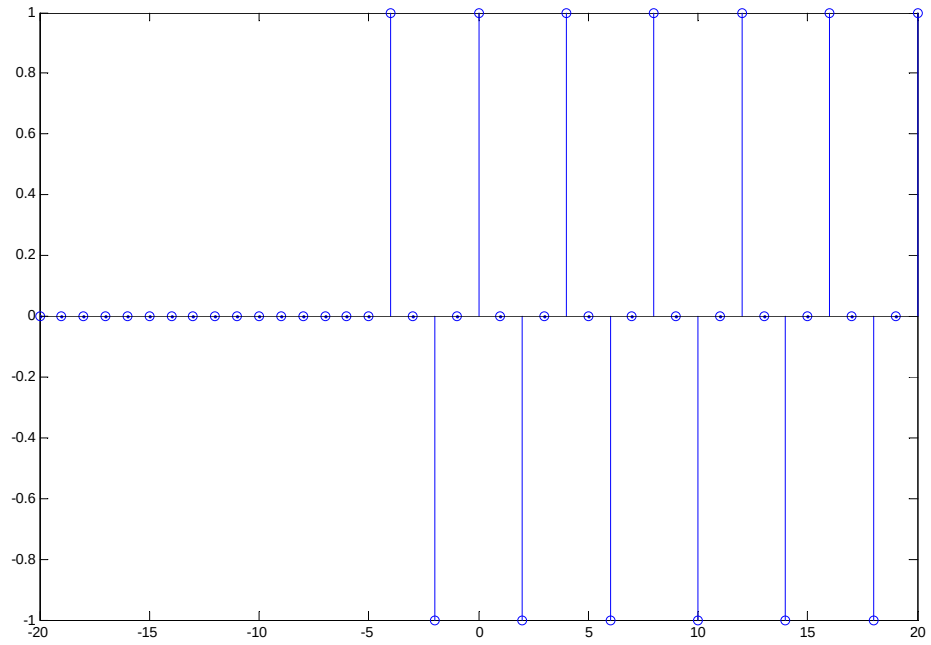
```
>> n=-20:20;
>> y=((1/2).^(n)).*degrau(n,3);
```

```
>> stem(n,y)
```



c) $y[n] = \cos\left(\frac{1}{2}\pi n\right)u[n+4]$

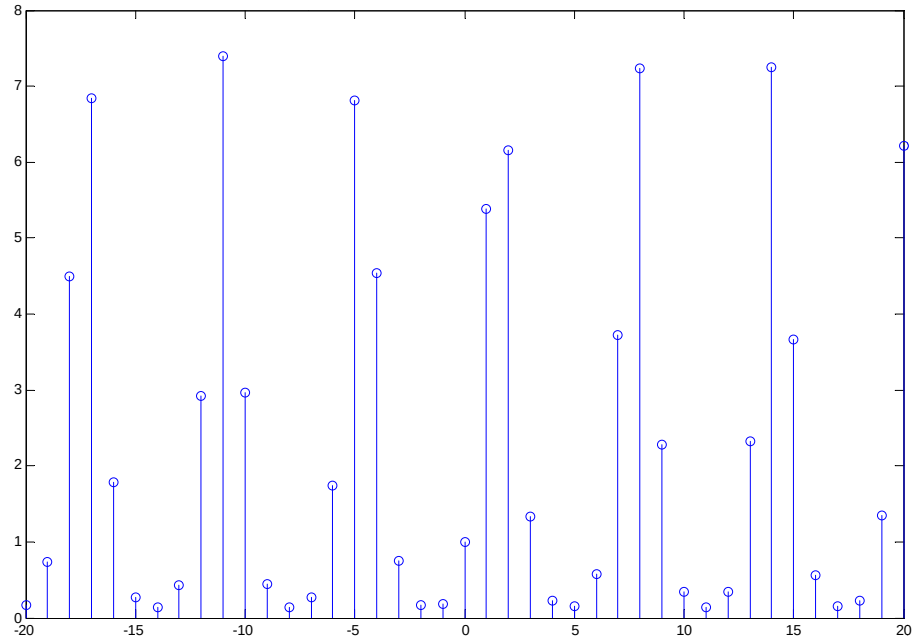
```
>> n=-20:20;
>> y=cos(0.5*pi*n).*degrau(n,-4);
>> stem(n,y)
```



d) $y[n] = e^{2sen(n)}$

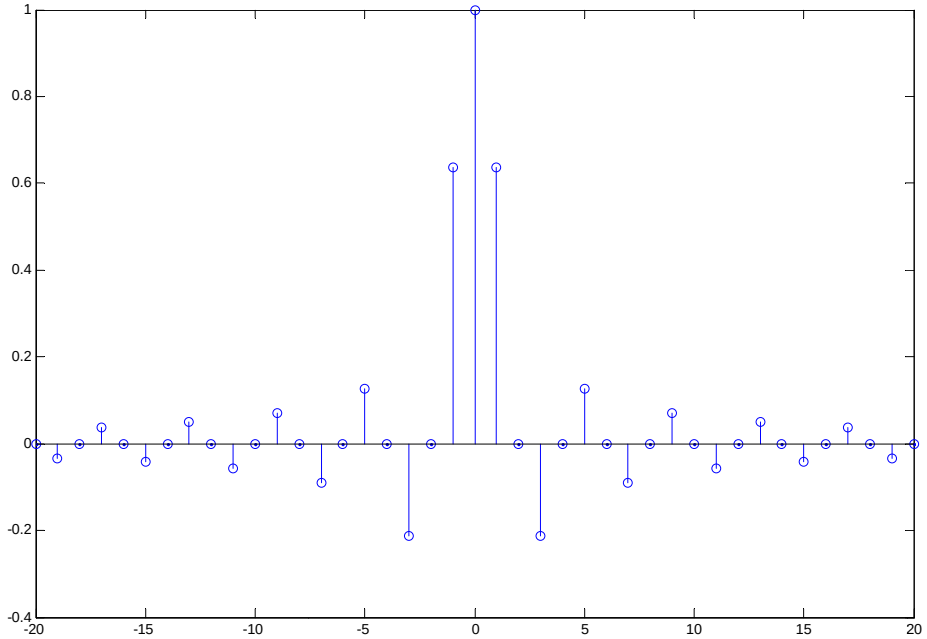
```
>> n=-20:20;
```

```
>> y=exp(2*(sin(n)));
>> stem(n,y)
```



e) $y[n] = \text{sinc}\left(\frac{n}{2}\right)$

```
>> x=-20:1:20;
>> y=sinc(x/2);
>> stem(x,y)
```



É bom salientar que nos exemplos anteriores foram dados exemplos de programas no qual se obtém as funções impulso e degrau. Entretanto, o MATLAB também possui funções que possibilitam isso de forma mais rápida, que são as funções *dirac* e a *heaviside,* conforme veremos a seguir:

- **dirac**

Definição: Obtém a função delta de Dirac, ou seja, a função impulso no intervalo determionado por *x.*.

Sintaxe:

dirac(x)

```
>> x=-10:10;
>> y=dirac(x-5);  %Impulso no instante 5
>> stem(x,y)
```

- **heaviside**

Definição: Determina a função degrau no intervalo determinado por *x*.

Sintaxe:

dirac(x)

```
>> x=-10:10;
>> y=dirac(x-5);  %Impulso no instante 5
>> stem(x,y)
```

Exemplo 9- Sabe-se que a função impulso é a derivada da função degrau. Determina este fato utilizando o MATLAB.

```
>> syms x
>> diff(heaviside(x),x)
ans =
dirac(x)
```

Exercício 17- Verifique a integral de $sen(x)\cdot\delta(x-5)$.

14.3. Convolução

A convolução é uma ferramenta matemática que expressa a saída de um sistema de tempo, seja este discreto ou contínuo, em função de uma entrada pré-definida e da resposta ao impulso do sistema.

O MATLAB possui uma função chamada de *conv* que realiza a convolução de sinais de duração finita. Por exemplo, sejam dois vetores *x* e *h* representando sinais. O comando *y = conv(x, h)* gera um vetor *y* que denota a convolução dos sinais *x* e *y*.

Veja que o número de elementos em *y* é dado pela soma do número de elementos em *x* e *y* menos um, devido ao processo de convolução. O vetor *ny* dado pelo espaço de tempo tomado pela convolução é definido pelo intervalo entre a soma dos primeiros elementos de *nx* e *nh* e a soma dos últimos elementos de *nx* e *nh,* sendo *nx* o espaço tempo definido para o vetor *x* e *nh* o espaço de tempo definido para o vetor *h*.

(ny = [(min(nx) + min(nh)):(max(nx) +max(nh))];)

. Vejamos a sintaxe de *conv* abaixo:

- **conv**

Definição: Determina a convolucão de dois sinais ou a multiplicação de dois polinômios.

Sintaxe:

w = conv(u,v)

```
h=[1,2,1];
x=[2,3,-2];
y=conv(x,y)
```

Exemplo 10- Determine os coeficientes do polinômio obtido ao multiplicar os polinômios $5x^2 + 3x$ com $2x + 2$.

```
>> a=[3 3 0];
>> b=[2 2];
>> y=conv(a,b)
```

```
y =
     6    12     6     0
```

Logo, o polinômio obtido seria $6x^3 + 12x^2 + 6x$.

Exemplo 11- Determine a resposta de um sistema com a entrada $x[n] = u[n-2] - u[n-7]$, sabendo que a resposta desse sistema ao impulso é $h[n] = u[n] - u[n-10]$.

```
h=ones(1,10);
x=ones(1,5);
n=2:15;
y=conv(x,h);
stem(n,y);
```

Exercício 18- Use o MATLAB para determinar a saída do sistema com entrada $x[n] = 2\{u[n+2] - u[n-12]\}$ sabendo que a resposta ao impulso desse sistema é $h[n] = 0,9^n\{u[n-2] - u[n-13]\}$.

14.4. Equações de Diferenças

As Equações de Diferenças é uma forma de expressarmos um sistema na forma recursiva que permita que a saída do sistema fosse computada a partir do sinal de entrada e das saídas passadas.

Um comando que é possível realizar uma função similar seria o *filter*, definida a seguir:

- **filter**

Definição: Expressa a descrição em equação de diferenças de um sistema em uma forma recursiva que permita que a saída do sistema seja computada a partir do sinal de entrada e das saídas passadas.

Sintaxe:

y = filter(b,a,X)

y = filter(b,a,X,zi)

Veja acima que apareceu o parâmetro *zi*, que determina a condição inicial de *y*. Este *zi* é uma matriz com as condições iniciais, sendo os valores passados de *y*.

Exemplo 12- Um exemplo clássico no uso de *filter* é determinar a seqüência de Fibonacci, definida como o número atual ser igual a soma dos dois números anteriores. Em linguagem matemática, tem-se $y[n]-y[n-1]-y[n-2]=0$ em que y é a saída do sistema.

Veja que ele não depende de uma entrada, mas ao usarmos o *filter*, é necessário usar a entrada apenas para definir o número de elementos da seqüência no qual se deseja obter, assim como é um parâmetro indispensável para o uso da função *filter*.

Será dado como condição inicial a matriz [1 0], correspondentes a entrada não desejada y[-1] e y[-2], indispensável para obter os outros valores. O código do programa que pode ser implementado no M-file segue abaixo. Neste caso, se deseja adquirir 20 valores.

```
a=[1, -1, -1];
b=[0];
x=ones(1,20);
y=filter(b,a,x,[1 0])
```

Exercício 19- Determine, utilizando filter, a seqüência tribonacci.

Quando se trabalha com sistemas de equações de diferenças, no qual precisa determinar a resposta desse sistema ao impulso, o comando *impz* se torna bastante útil. A sua sintaxe segue abaixo:

- **impz**

Definição: Determina a resposta ao impulso de um sistema de equações de diferenças.

Sintaxe:

[h,t] = impz(b,a)

[h,t] = impz(b,a,n)

O comando [h,t] = impz(b,a,n) avalia *n* valores da resposta ao impulso de um sistema descrito por uma equação de diferenças. Os coeficientes da equação de diferenças estão contidos nos vetores *b* e *a* no que se refere a *filter*. O vetor *h* contém os valores da resposta ao impulso e *t* contém os índices de tempo correspondentes.

14.5. FFT (Transformada Rápida de Fourier)

A Transformada de Fourier leva uma função no domínio do tempo para o domínio da freqüência, no qual podemos analisar as freqüências mais importantes (com maior amplitude) de uma função. A transformada inversa de Fourier faz o processo inverso, passa uma função do domínio da freqüência para o domínio do tempo.

A Transformada de Fourier e sua inversa podem ser calculadas a partir das expressões abaixo, respectivamente:

$$F(\omega) = \int_{-\infty}^{\infty} S(t) e^{-j\omega t} dt$$

$$S(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} F(\omega) e^{j\omega t} d\omega$$

Onde ω é a freqüência fundamental.

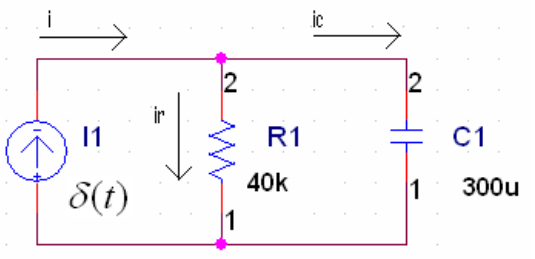
A FFT (Transformada rápida de Fourier) é um algoritmo computacional otimizado que calcula a Transformada Discreta de Fourier mais rapidamente. A FFT também pode servir de aproximação para a Transformada de Tempo Discreto de Fourier, Série de Fourier e a própria Transformada de Fourier.

Uma propriedade da Transformada de Fourier é que a transformada da convolução de duas funções equivale à multiplicação das duas no domínio da freqüência. Portanto para calcular a convolução de uma função levamos os

dois sinais para o domínio da freqüência, multiplicamos e voltamos para o domínio do tempo. Veja a expressão abaixo:

$$y(t) = x(t) * h(t) = IFFT[FFT(x(t)).FFT(h(t))]$$

Exemplo 13- Dado o circuito RC abaixo, determine a resposta ao impulso e a corrente no capacitor $i_C(t)$ quando a entrada $x(t)$ é igual a e^{-t}.



Resolução:

Cálculo da resposta ao impulso:

Lei dos nós:

$i = i_R + i_C$

$$i = i_R + C\frac{dV}{dt}$$

Em t=0, $i = \delta(t)$ e $i_R = 0$

$$\frac{1}{C}\delta(t) = \frac{dV}{dt} \rightarrow \boxed{V(0) = V(0^+) = \frac{1}{C}}$$

$Em\ t = 0^+,\ i(0^+) = 0$

$$(1)\quad 0 = \frac{V}{R} + C\frac{dV}{dt} \xrightarrow{\text{Resolução equação diferencial}} V(t) = \frac{1}{C}e^{-t/RC}$$

$$i(t) = h(t) = C\frac{dV}{dt} = -\frac{1}{RC}e^{-t/RC}$$

Para R=40kΩ e C=300 μ F $\rightarrow$ $\boxed{h(t) = -\frac{1}{12}e^{-t/12}A}$

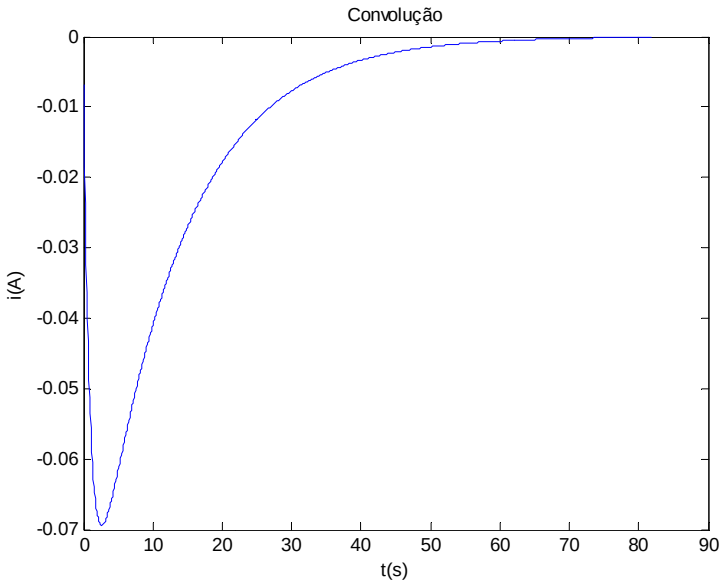
Cálculo da convolução analiticamente:

$$i_C(t) = y(t) = \int_{-\infty}^{t} x(\lambda)h(t-\lambda)\,d\lambda = \int_{0}^{t} e^{-\lambda}\left(-\frac{1}{12}e^{-(t-\lambda)/12}\right)d\lambda$$

$$= -\frac{e^{-t/12}}{12}\int_{0}^{t} e^{-11\lambda/12}d\lambda = \frac{e^{-t/12} - e^{-t}}{11}$$

Cálculo da convolução através do MATLAB:

```
n=[0:0.08:81.84];        %amostragem
x=exp(-n);               %definição da entrada
h=-exp(-n/12)/12;        %definição da saída
fftx=fft(x);             %cálculo da fft
ffth=fft(h);
ffty=fftx.*ffth;         %multiplicação
y=ifft(ffty);            %inversa
plot(n,-abs(y)*0.08)
title('Convolução');
xlabel('t(s)');
ylabel('i(A)');
```



Exercício 20- Calcule a convolução das formas de onda $\begin{cases} x(t) = e^{-5t} \\ h(t) = \cos(2.5t) - t \end{cases}$

14.6. Filtros Digitais

O Matlab possui inúmeras funções que permitem ao usuário descobrir a função transferência de diferentes tipos de filtros digitais:

A função de transferência digital é definida por H(z) onde $z = e^{jwt}$. Na forma geral a função de transferência H(z) é a seguinte:

$$H(z) = \frac{B(z)}{A(z)}$$

$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + ... + b_n z^{-n}}{a_0 + a_1 z^{-1} + a_2 z^{-2} + ... + a_n z^{-n}}$$

- **Butter**

Definição: Determina os coeficientes de um filtro Butterworth. Esse filtro pode ser passa baixa, passa alta, passa banda, rejeita banda.

Sintaxe:

[B,A] = Butter(N, Wn, 'tipo') → Determina os coeficientes da função transferência dada a freqüência de corte e o tipo de filtro. Caso nada seja posto em 'tipo', o Matlab interpreta filtro passa baixa como padrão.

- **Freqz**

Definição: Calcula os valores de uma função complexa H(z)

Sintaxe:

Freqz(B,A,n) → Utiliza 3 argumentos de entrada. O primeiro é um vetor contendo os coeficientes do polinômio B(z) da Equação (1). O segundo é um vetor contendo os coeficientes do polinômio A(z). O terceiro é para especificar o número de valores de freqüências normalizadas que se quer no intervalo de 0 a π.

Exemplo 14-

- Gerar um sinal com duas senóides de freqüências 5 e 80 Hz, com fs=200 Hz.
- Projetar um filtro para fs=200 Hz. Usar filtro de 2a ordem, Butterworth.
- Filtrar o sinal.
- Plotar a resposta em freqüência.

```
% Exemplo de filtros

fs=200;      % Freqüência de amostragem
t=0:1/fs:1;   % Tempo de amostragem
T=1/fs;
x=sin(2*pi*5*t)+sin(2*pi*80*t);   % sinal de entrada
figure(4)
plot(t,x)
title('Sinal de Entrada')
xlabel('tempo (s)')
ylabel('amplitude')

[B,A]=butter(2,20/(fs/2));   % Determinar os coeficientes
B     % Mostrar coeficientes B
A     % Mostrar coeficientes A

% Plotagem da resposta em freqüência
h1=freqz(B,A,100)
figure(1)
plot(abs(h1))
grid
title('Resposta em freqüência')
xlabel('freqüênca (Hz)')
ylabel('amplitude')

% Filtragem
figure(2)
y=filter(B,A,x);
plot(t,y,'k-')
title('Sinal de Entrada')
xlabel('tempo (s)')
ylabel('amplitude')
```

Exercício 21- Projete um filtro passa-alta de Butterworth de ordem 6, com freqüência de corte de 10 Hz. Use f_s=400 Hz. Sinais a serem filtrados: senóides de 1 e 20 Hz. Use as funções butter, filter e freqz.

15.REFERÊNCIAS BIBLIOGRÁFICAS

[1] **CARNAHAN**, Brice, **LUTHER**, H. A. & **WILKES**, James O. *Applied numerical methods*. John Wiley & Sons, Inc. Nova Iorque – 1969.

[2] **GÓES**, Hilder & **TONAR**, Ubaldo. *Matemática para Concursos*. 6ª Edição. ABC Editora. Fortaleza – CE. 2001.

[3] **HAYKIN**, Simon & **VEEN**, Barry Van. *Sinais e Sistemas*. Editora Bookman. Porto Alegre – RS. 2001

[4] **HAYT**, William H. Jr. & **BUCK**, Jonh A. *Eletromagnetismo*. 6ª Edição. Editora LTC. Rio de Janeiro – RJ. 2001.

[5] **LEITHOLD**, Louis. *O Cálculo com Geometria Analítica*. 3ª Edição. Volume I. Editora Habra. São Paulo – SP. 1994.

[6] **NILSSON**, James W & **RIEDEL**, Susan A. *Circuitos Elétricos*. 6ª Edição. Editora LTC. Rio de Janeiro – RJ. 2003.