MATLAB

erig@utfpr.edu.br

1 INTRODUÇÃO

Esta apostila tem o objetivo de servir como referência básica para um curso de técnicas de programação baseadas em Matlab e Simulink. Esta referência, aliada a atividades de laboratório, deve propiciar ao aluno uma base de conhecimento necessária para o bom aproveitamento em um curso de automação e controle. Este material tem não tem a pretensão de ser completo ou suficiente em qualquer dos temas abordados. Referências adicionais são feitas com o desejo que os alunos busquem o importante aprofundamento nos diversos temas aqui explorados.

O capítulo 2 apresenta alguns conceitos básicos para operação do Matlab. No capítulo 3 é explorada a operação com vetores e matrizes. O capítulo4 apresenta técnicas de geração de gráficos 2D e 3D. Finalmente o capítulo 5 descreve conceitos de programação no Matlab.

2 O AMBIENTE MATLAB

Introdução

Neste capítulo é apresentado o ambiente de programação Matlab, seus comandos básicos, operações aritméticas, operação com números complexos e formatação de dados.

Princípios Básicos

A tela de entrada do Matlab, apresentada na figura 1, permite observar uma série de janelas:

- Current Directory
- Command History
- Command Window

A janela mais importante é a Command Window, ou simplesmente janela de comandos. O Matlab trabalha com o conceito de linhas de comando. Através de uma seqüência de linhas de comando é possível resolver problemas matemáticos complexos, traçar gráficos e fazer uso de ferramentas dedicadas nas mais diversas áreas da engenharia.
Um comando muito importante é o comando >>help. Este comando lista uma série tópicos que podem ser explorados de forma detalhada. Por exemplo, digitando-se >>help general, tem-se uma descrição de comandos de uso geral usados pelo Matlab. Mais uma vez, digitando-se >> help demo, tem-se uma descrição do que o comando >>demo realiza se executado.

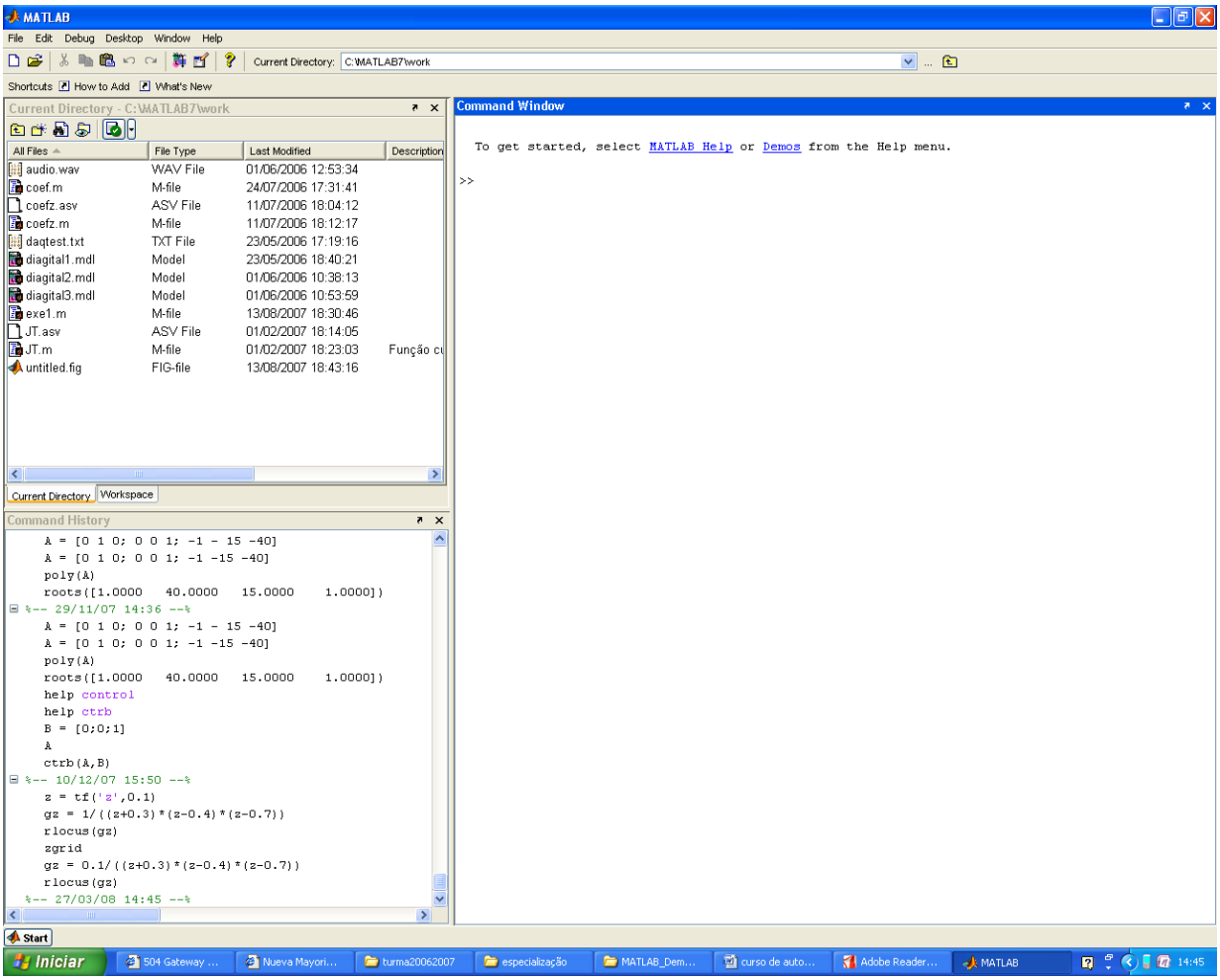

Figura 1 Tela inicial do Matlab.

Aritmética Básica

Antes de mais nada é importante aprender a usar os comandos de aritmética básica. Por exemplo, podemos executar a quatro operações básicas com os seguintes comandos:

Soma:

```
>> 1.2 + 2.33

ans =
   3.5300
```

Subtração:

```
>> 1.2 - 2.33

ans =
  -1.1300
```

Multiplicação:

```
>> 3.14* 11

ans =
  34.5400
```

Divisão:

```
>> 3.14/3

ans =
   1.0467
```

Combinação de diversas operações:

```
>> 1.23*9/(3-10.1)+9

ans =
   7.4408
```

Repare que o uso adequado dos parênteses na operação anterior deve ser observado.

O Matlab trabalha com o conceito de variáveis. Na verdade, o resultado das operações anteriores está armazenado na variável "ans". Assim digitando-se:

```
 >> ans

ans =
   7.4408
```

, ou seja, retorna o último valor armazenado.

É possível definir variáveis:

```
>> x = 3, y = 2
```

```
x =

    3

y =

    2
```

Ou:

```
>> x = 3; y = 2;
```

O caractere ";" inibe a apresentação do resultado da operação, mas não a operação.

Agora que x e y estão definidos pode-se fazer:

```
>> x*y

ans =
    6
```

Ou:

```
>> x = x*y

x =
    6
```

Outra operação muito usada:

Exponenciação:

```
>> x^y

ans =
   36
```

O comando >> help ops apresenta a lista de operadores disponíveis no Matlab, inclusive operadores aritméticos, operadores lógicos,

operadores relacionais, operadores de conjuntos e alguns caracteres especiais.

Com a definição de vária variável pode ser necessário verificar quais variáveis já forma criadas. Para isto existem dois comandos:

```
>> who

Your variables are:

ans  x    y


>> whos
  Name      Size                Bytes  Class

  ans       1x1                     8  double array
  x         1x1                     8  double array
  y         1x1                     8  double array

Grand total is 3 elements using 24 bytes
```

O comando >>who simplesmente lista as variáveis já criadas, enquanto o comando >>whos detalha as mesmas, descrevendo inclusive sua dimensão e o número de bytes utilizados.

O comando >>clear limpa a lista de variáveis criada ou apenas um conjunto de variáveis desejadas. Por exemplo:

```
>> clear x, who

Your variables are:

ans  y
```

Exercício: Encontre o resultado da operação $x = 3\frac{1}{2}[(1.1^2 - 9)/13 - 15]$.

Formatando o resultado de saída

A função format permite a apresentação do resultado das operações de diferentes modos:

```
>> format long
>> x = 3 + 11/16 + 2^1.2

x =
5.98489670999407
>> format short
>> x = 3 + 11/16 + 2^1.2

x =
5.9849
```

Estes dois modos básicos da função format permitem a representação usando um maior ou um menor número de casas decimais. Tente as outras opções:

```
>>format long e
>>format short e
>>format long g
>>format short g
>>format long eng
>>format short eng
>>format bank
>>format rat
```

Representando números complexos

Os números complexos são gerados com o auxílio do caractere 'i''i'. Assim:

```
>> format
>> a = 2+3i;
>> b= 10+2i;
>> c = a+b
c =
  12.0000 + 5.0000i
```

Operando com arquivos

A área de trabalho, incluindo as variáveis que já foram criadas pode ser salva através dos passos: Menu File- Save WorkSpaceAs. Escolhendo-se um nome para o arquivo com extensão. MAT salva-se a área de trabalho corrente.

Exercício: Execute os seguintes passos:

1. Salve a área de trabalho
2. Execute o comando >>c = 1.11;
3. Execute >>c
4. Execute >>clear
5. Execute >>c
6. Carregue o arquivo em que foi salva a área de trabalho
7. Execute >>c

Pode-se observar que o valor da variável 'c' é perdido com o comando clear e recuperado com a carga do arquivo em que foi salva a área de trabalho. Outra forma muito útil de operar com o Matlab é através de M-files ou arquivos com extensão. M, onde uma seqüência de comandos pode ser armazenada. Isto permite inclusive a entrada e saída de dados, criando o conceito de função. Por exemplo, através dos passos: Menu File – New M-File é gerada uma nova janela. Nesta janela digite as seguintes linhas:

a =10

b = 20

a+b

Salve esta nova janela como exemplo1.m.

Agora, na linha de comandos do Matlab digite:

```
>> exemplo1
a =
   10
b =
   20
ans =
   30
```

Posteriormente será apresentado um tópico sobre programação de funções M.

Exercício: Com y = 2 e z = 1.3, crie um arquivo .m para resolver o problema

$$x = 3\frac{1}{2}[(1.1^2 - 9 * y)/13 - 15/z]$$

3 MATRIZES e VETORES

Muitas vezes, particularmente na área de automação e controle é necessária a representação de dados ou de um sistema através de vetores e matrizes.

Vetores

A representação de uma cadeia de números é conhecida como vetor. Um vetor linha $x = [1\ 2\ 3]$ pode ser representado como:

```
>> x = [1 2 3]

x =
    1    2    3
```

O segundo elemento deste vetor será:

```
>> x(2)

ans =
    2
```

Um vetor coluna $y = \begin{bmatrix} 13 \\ 14 \\ 15 \end{bmatrix}$ pode ser representado por:

```
>> y = [13; 14; 15]

y =
   13
   14
   15
```

Outra maneira é:

```
>> y = [13 14 15];
>> y= y'

y =
   13
   14
```

15

Onde y’ e o vetor transposto do vetor y original. O vetor linha é transposto em vetor coluna.

É possível a soma e a subtração de vetores com a mesma dimensão. Por exemplo:

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

c =
3
7
8


>> x = [3,0,3];
>> y = [2,1,1];
>> x = x – y

x =
1 –1 2
```

É possível a concatenação de vetores. Por exemplo, baseado nos vetores c e x já criados:

```
>> f = [c',x]

f =
   3    7    8    1   -1    2

>> f = [c ; x']

f =
    3
    7
    8
    1
   -1
    2
```

Outra maneira de se criar um vetor é através de elementos igualmente espaçados em um intervalo. Por exemplo, criando um vetor linha com elementos entre 0 e 10 com intervalo de 1:

```
>> x = [0:1:10]

x =
    0    1    2    3    4    5    6    7    8    9   10
```

Alguns comandos são usados para operar sobre os vetores. O comando length determina o tamanho do vetor:

```
>> length(x)

ans =
   11
```

Os comandos max e min determinam, respectivamente, o valor máximo e mínimo dentro do vetor:

```
>> max(x),min(x)
ans =
   10
ans =
    0
```

Matrizes

Matrizes são arranjos bidimensionais de elementos. Estes elementos podem ser números, funções, variáveis ou mesmo gráficos. Por exemplo, a matriz A = $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ pode ser representada no Matlab como:

```
>> A = [1, 2; 3, 4]

A =

    1    2
    3    4
```

Várias operações com matrizes podem ser realizadas. Multiplicação por escalar:

```
>> A = [1, 2; 3, 4];
>> 2*A

ans =
    2    4
    6    8
```

Soma de Matrizes:

```
>> A = [1, 2; 3, 4];
>> B = [3, -1;-1 5];
>> A+B

ans =
    4    1
    2    9
```

Multiplicação de Matrizes:

```
>> A = [1, 2; 3, 4];
>> B = [3, -1;-1 5];
>> A*B

ans =
    1    9
    5   17
```

Matriz inversa:

```
>> B = [3, -1;-1 5];
>> inv(B)

ans =
   0.3571    0.0714
   0.0714    0.2143
```

Matriz transposta:

```
>> B = [3, -1;-1 5];
>> B'
```

```
ans =
    3   -1
   -1    5
```

Determinante:

```
>> A = [1, 2; 3, 4];
>> det(A)

ans =
   -2
```

A matriz identidade pode ser gerada com o comando eye(n), onde n é a ordem da matriz. Assim:

```
>> eye(5)

ans =
    1    0    0    0    0
    0    1    0    0    0
    0    0    1    0    0
    0    0    0    1    0
    0    0    0    0    1
```

Uma matriz randômica pode ser gerada pelo comando rand(n), onde n é ordem da matriz. Assim:

```
>> rand(3)

ans =
   0.9501   0.4860   0.4565
   0.2311   0.8913   0.0185
   0.6068   0.7621   0.8214
```

Exercício: Resolva o sistema:

$$\begin{aligned} 5x + 2y - 9z &= 44 \\ -9x - 2y + 2z &= 11 \\ 6x + 7y + 3z &= 44 \end{aligned}$$

4 GRÁFICOS

É muito comum a necessidade de representar os resultados de um porblema na forma gráfica. O Matlab é bastante rico em opções de representação gráfica.

Gráficos 2D

A criação do gráfico de uma função segue os passos:

1. Definir a função.
2. Especificar a faixa de valores que será representada no gráfico.
3. Usar a função plot(*x*, *y*).

Por exemplo, a função $y = \cos(x)$ na faixa $0 \leq x \leq 10$, com incrementos de 1 pode ser traçada (na figura 2) usando os seguintes comandos:

```
>> x = [0:1:10];
>> y = cos(x);
>> plot(x, y)
```

A mesma função, com incrementos de 0.1 pode ser traçada (na figura 3) usando os seguintes comandos:

```
>> x = [0:0.1:10];
>> y = cos(x);
>> plot(x, y)
```

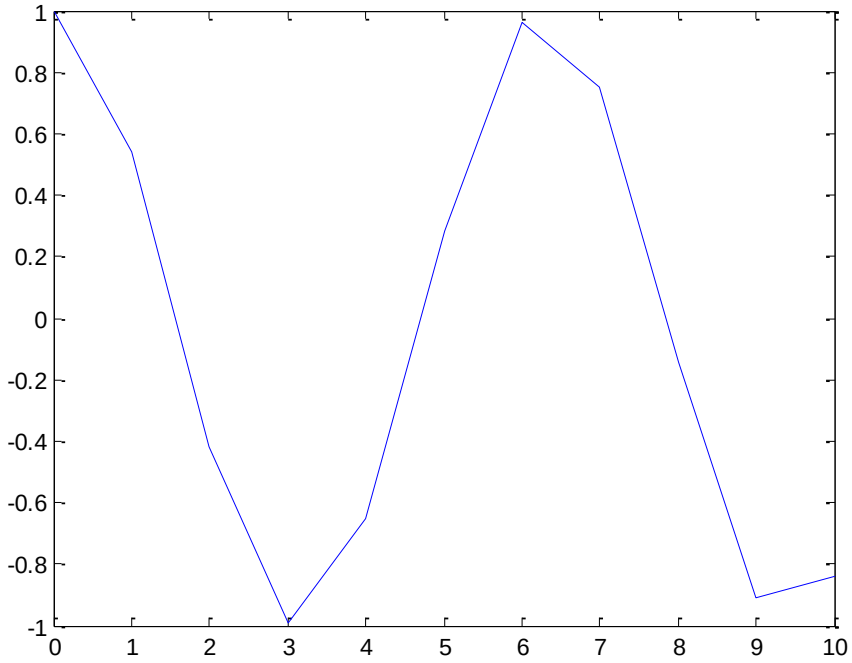

Figura 2 Traçado da função $y = \cos(x)$ com incrementos de 1.

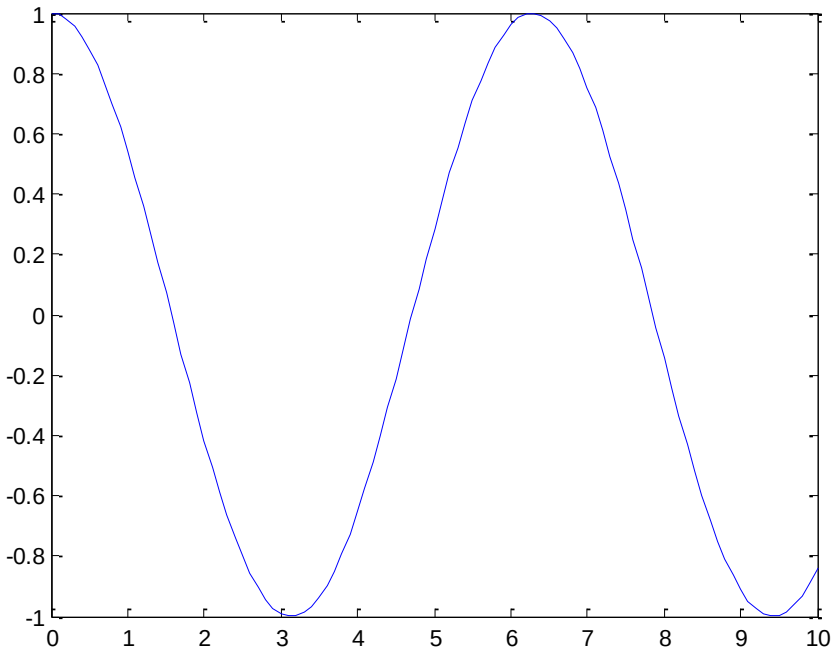

Figura 3 Traçado da função $y = \cos(x)$ com incrementos de 0.1.

Alternativamente, pode-se melhorar a representação com o uso de nomes nos eixos x e y, bem como de um título, representados na figura 4:

```
>>x=[0:0.01:10];
>>y=cos(x);
>> plot(x,y), xlabel('valores de x'), ylabel('valores de cos(x)'), title('PRIMEIRO GRÁFICO');
```

Outra maneira de se gerar o vetor x é através do comando linspace. Neste caso são gerados 100 pontos dentro de um intervalo especificado:

```
>>x = linspace(0,10);
```

Este comando é bastante útil para geração de pontos entre valores não inteiros, como por exemplo entre 0 e PI (3,1415).

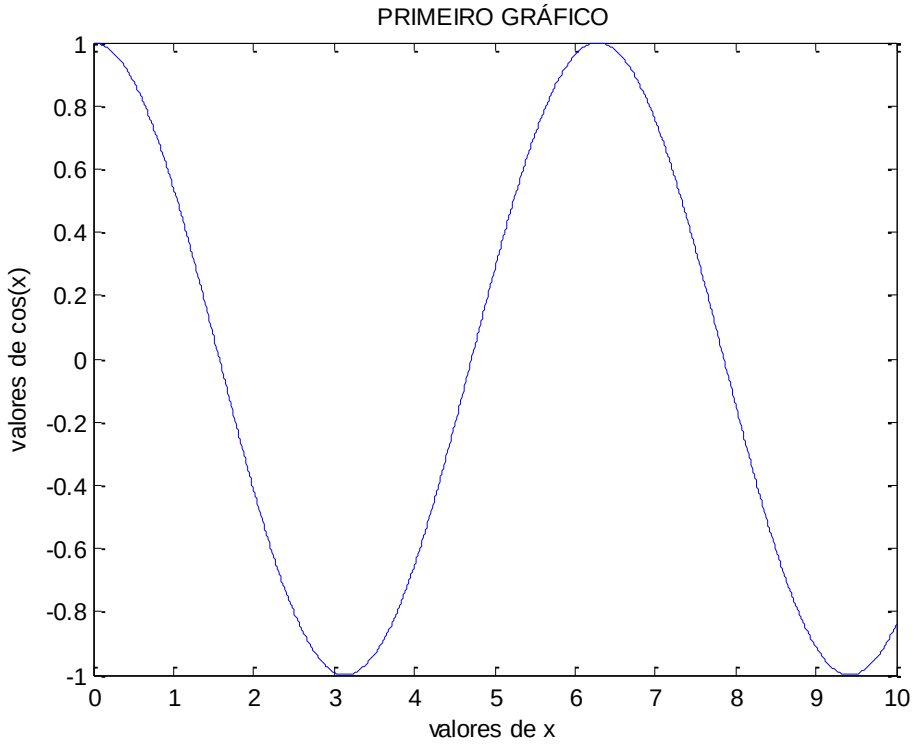

Figura 4 Traçado da função $y = \cos(x)$ com incrementos de 0.1, usando legendas nos eixos.

O comando fplot é uma alternativa para gerar o gráfico de uma função sem a definição prévia de um vetor de pontos de entrada. Basta definir a função e a faixa de valores a ser usada. A figura 5 ilustra a linha de comando:

```
>>fplot('exp(–2*t)*sin(t)',[0,4]), xlabel('t'), ylabel('f(t)'),title('Damped Spring Forcing')
```

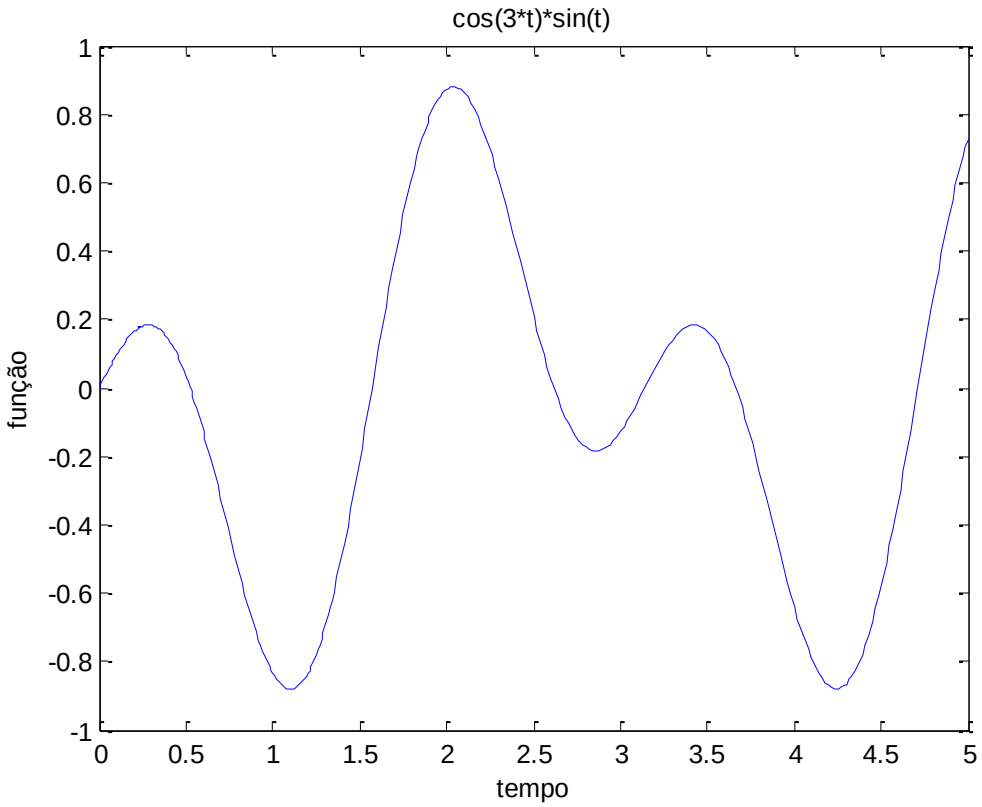

Figura 5 Função plotada com comando fplot.

A mesma função pode ser traçada na figura 6 usando-se grades:

```
>>fplot('cos(3*t)*sin(t)',[0,5]),xlabel('tempo'),ylabel('função'),title('cos(3*t)*sin(t)'),grid on
```

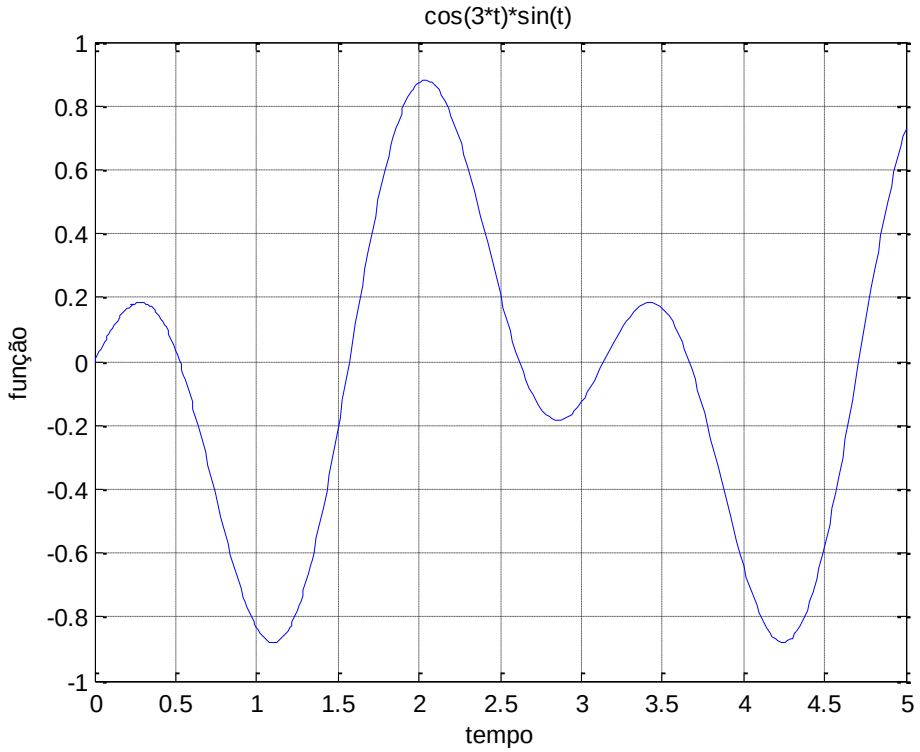

Figura 6 Função plotada com comando fplot e usando-se grid on.

Ainda é possível trabalhar com várias gráficas em uma única figura. Para tanto é comum usar caracteres especiais para diferenciar uma curva de outra. Alguns caracteres especiais usados são:

. ponto

- linha contínua (é o padrão)

o círculo

: dois pontos

x caracter x

-. traço-ponto

+ mais

-- tracejado

s quadrado

d diamante

Outra alternativa é o uso de cores para diferenciação das curvas. Alguns caracteres especiais usados para representar cores são:

b azul

g verde

r vermelho

y amarelo

k preto

w branco

Na figura 7 são representadas duas curvas, segundo os comandos:

```
>>plot(t,exp(-t),'s',t,exp(-3*t),'—'),xlabel('tempo'),ylabel('função 1 e função 2')
```

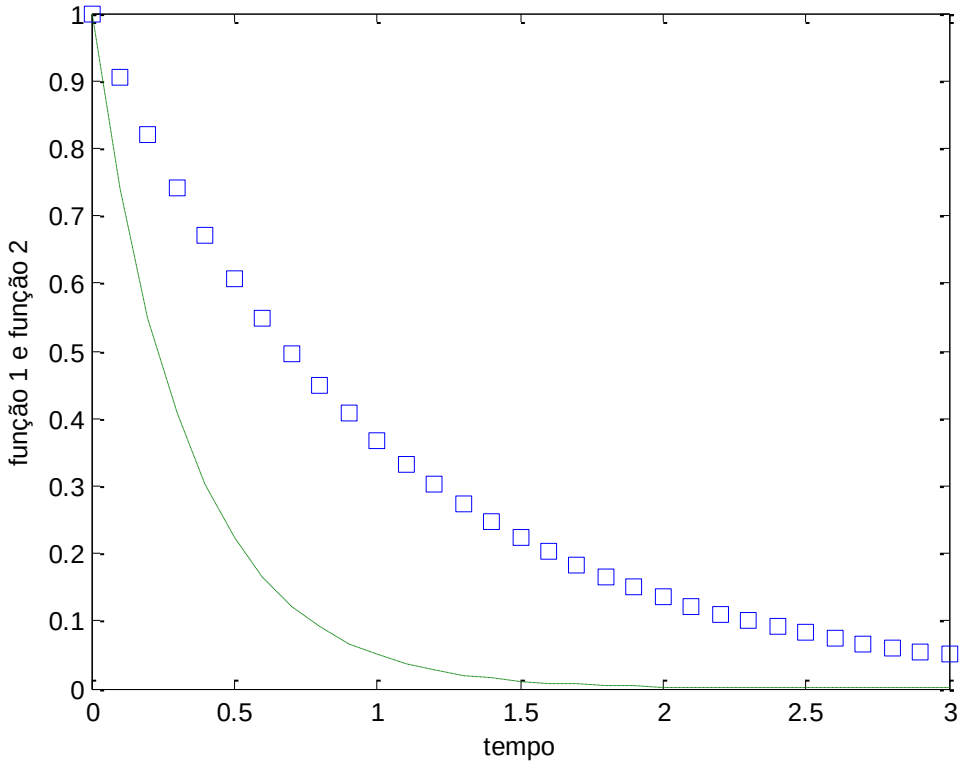

Figura 7 Exemplo de duas curvas plotadas em um mesmo gráfico.

No caso de múltiplas curvas em um mesmo gráfico é comum o uso de legendas. Isto pode ser visto na figura 8:

```
>> plot(t,exp(-t),'s',t,exp(-3*t),'—'),xlabel('tempo'),ylabel('função 1 e função 2')
>> legend('exp(-t)','exp(-3t)')
```

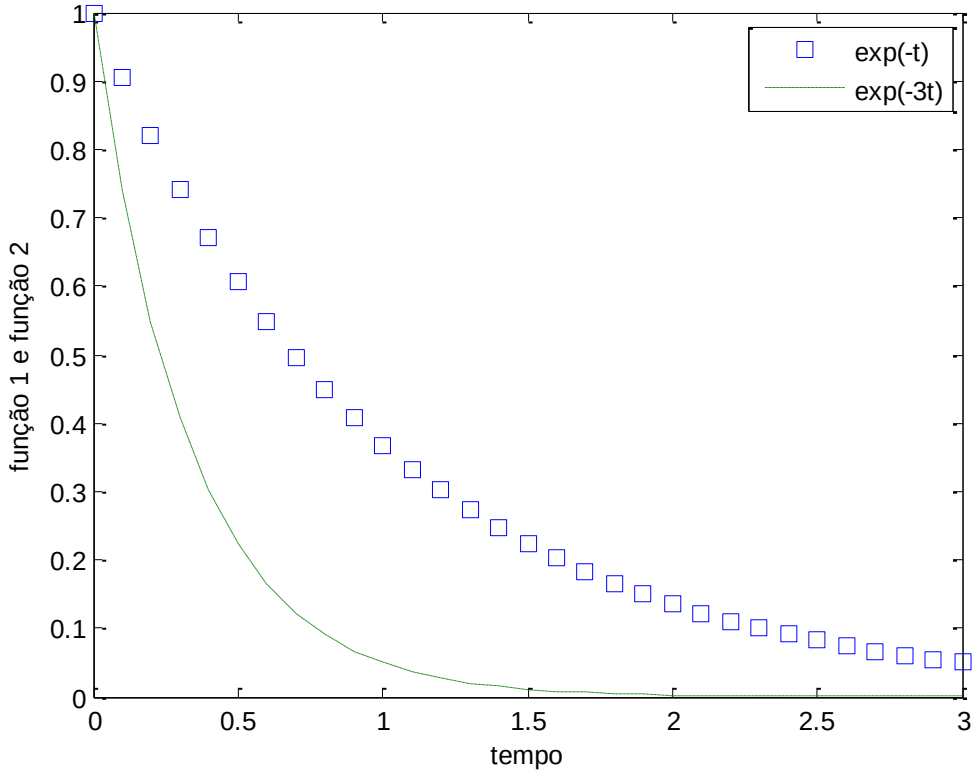

Figura 8 Múltiplas curvas com legendas.

Alternativamente é possível representar múltiplos curvas com múltiplos gráficos agrupados usando-se o comando subplot. Um subplot é um elemento de um vetor de gráficos. A notação usada é subplot (m,n,k), que significa o k-ésimo gráfico em uma matriz de gráficos mxn. O exemplo é apresentado na figura 9:

```
>> t = [0:0.1:3];
>> f1 = exp(-t);
>> f2 = exp(-3*t);
>> subplot(1,2,1);
>> plot(t,f1),xlabel('tempo'),ylabel('f1')
>> subplot(1,2,2);
>> plot(t,f2),xlabel('tempo'),ylabel('f2')
```

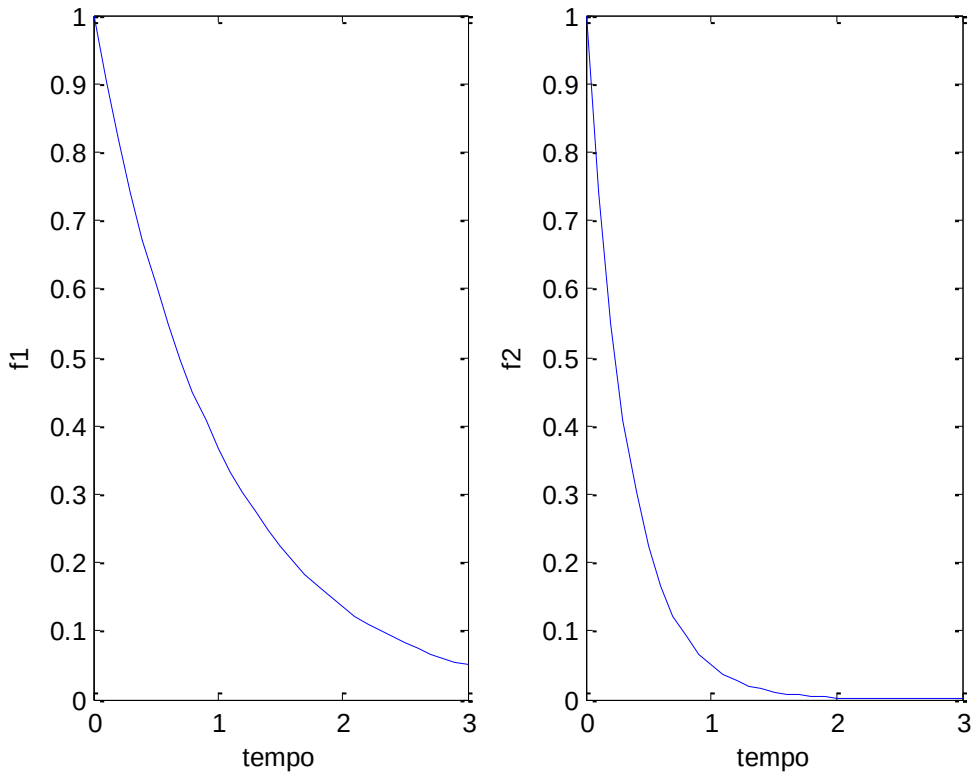

Figura 9 Exemplo de aplicação do comando subplot.

Outro arranjo é representado na figura 10:

```
>> t = [0:0.1:3];
>> f1 = exp(-t);
>> f2 = exp(-3*t);
>>subplot(2,1,1);
>> plot(t,f1),xlabel('tempo'),ylabel('f1')
>> subplot(2,1,2);
>> plot(t,f2),xlabel('tempo'),ylabel('f2')
```

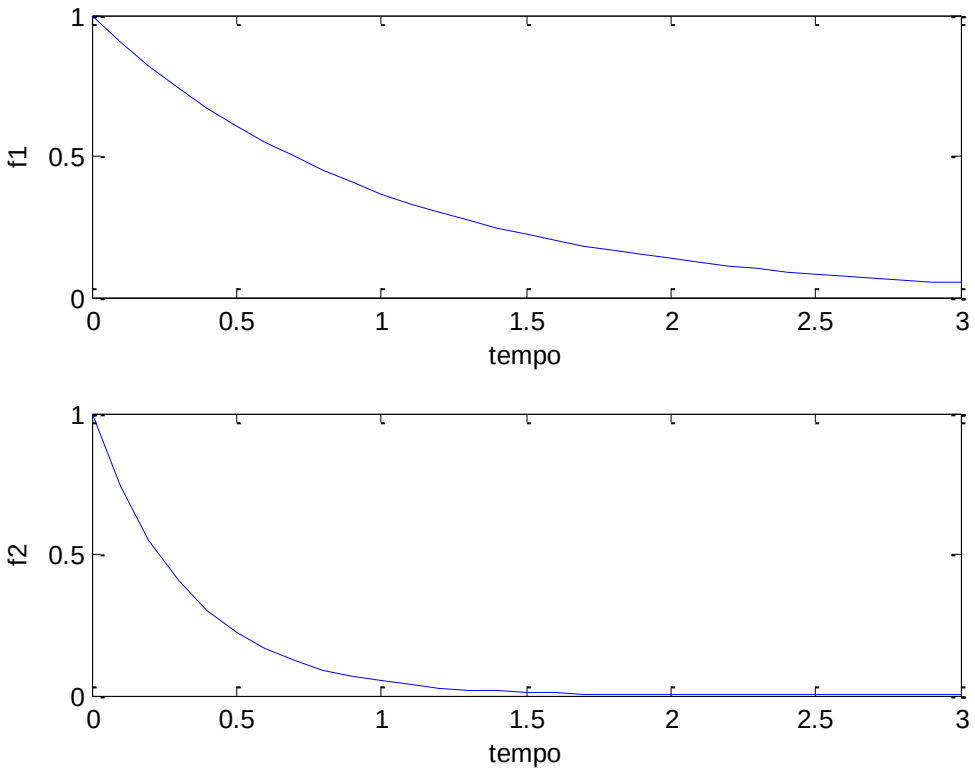

Figura 10 Exemplo de aplicação com comando subplot.

O comando hold também permite que curvas sejam plotadas em um mesmo gráfico. Assim, na figura 11, representa-se o uso do comando hold:

```
>> t = linspace(0,2*pi);
>> plot(cos(t),'.')
>> hold
Current plot held
>> plot(cos(2*t+pi/2))
>>legend('cos(t)','cos(2*t+pi/2)')
```

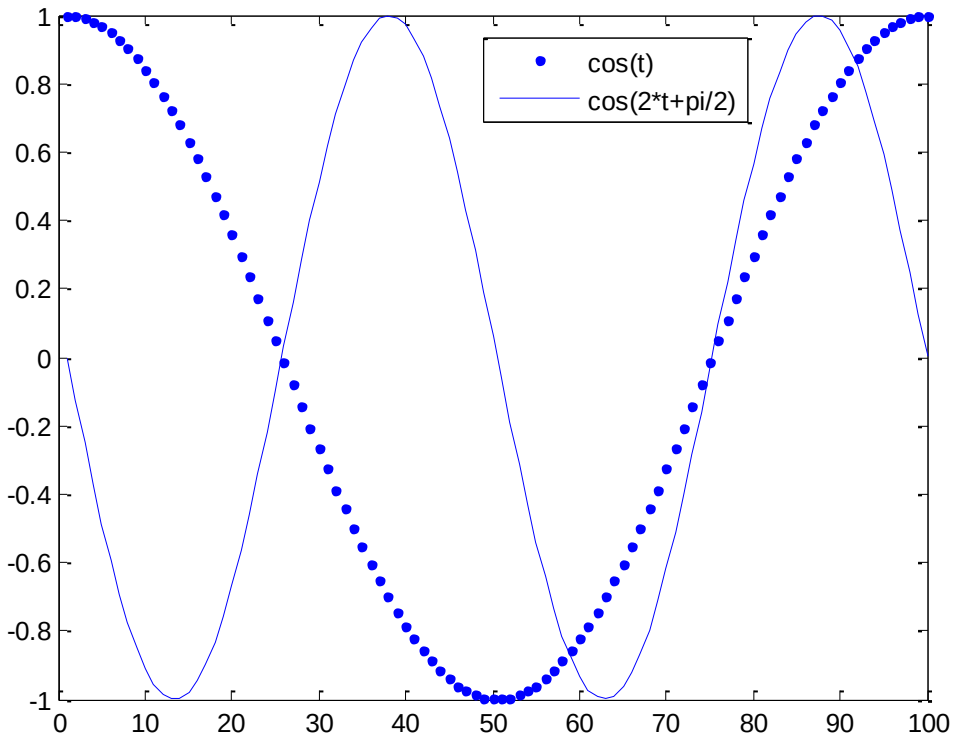

Figura 11 Uso do comando hold para geração de múltiplas curvas em um único gráfico.

Gráficos Polares

Gráficos polares são usuais em engenharia, particularmente em representação de comportamento de sistemas de controle. Um primeiro exemplo de um gráfico polar pode ser ilustrado na figura 12, segundo os comandos:

```
>> theta = linspace(0,2*pi);
>> modulo = 2*theta;
>> polar(theta,modulo)
```

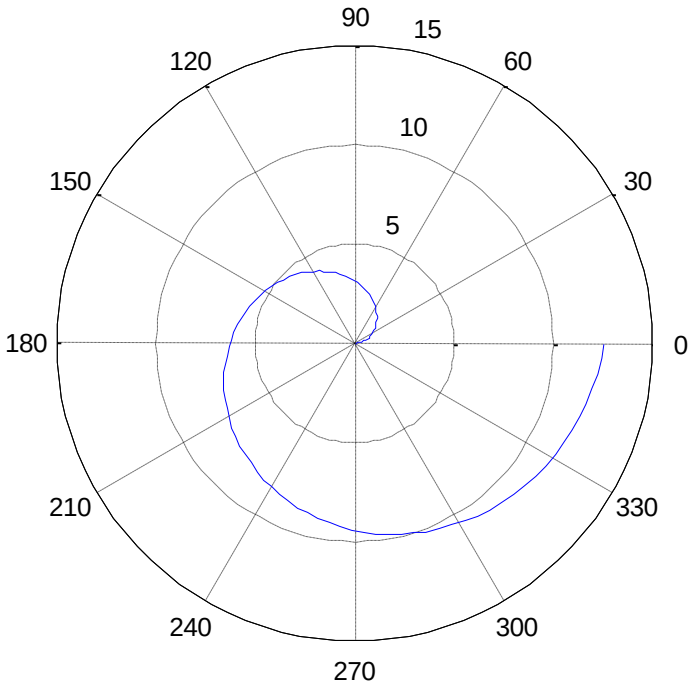

Figura 12 exemplo de gráfico polar.

Gráficos Discretos

Eventos discretos são usualmente representados com gráficos discretos. Um exemplo é a amostra de temperatura de um aquário (figura 13):

```
>> temp = [27, 32, 33, 31, 29, 28, 27, 32];
>>x = [1:length(temp)];
>> plot(x,temp),xlabel('amostra'),ylabel('temperatura')
```

E sua variante (figura 14):

```
>> temp = [27, 32, 33, 31, 29, 28, 27, 32];
>>x = [1:length(temp)];
>> bar(x,temp),xlabel('amostra'),ylabel('temperatura')
```

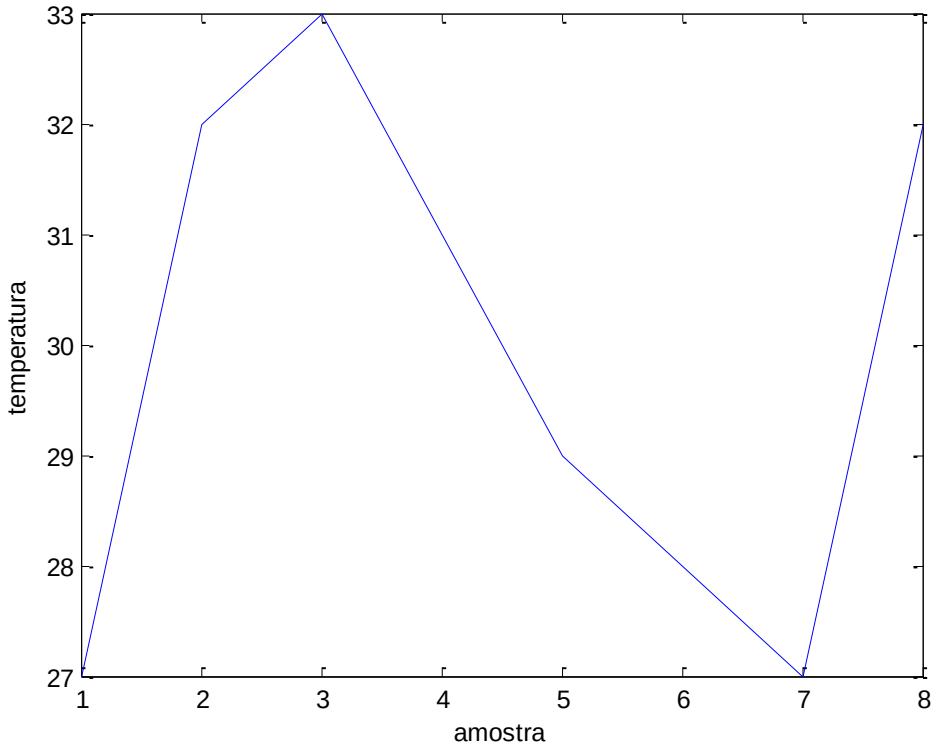

Figura 13 Exemplo de gráfico de amostras de temperatura.

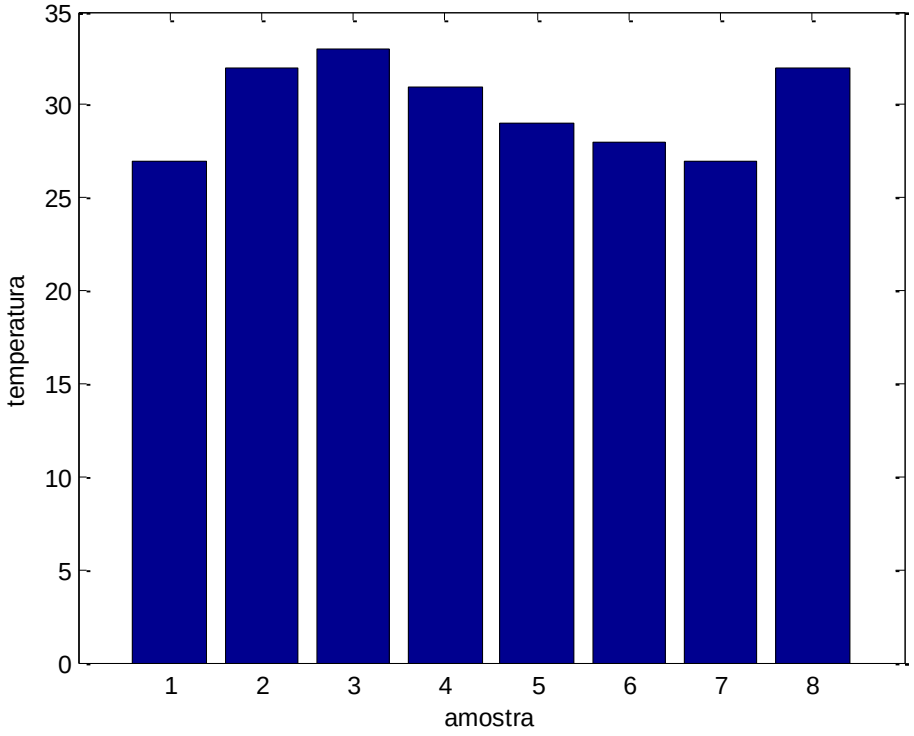

Figura 14 Exemplo de gráfico de amostras de temperatura.

As seguintes linhas de comando mostram o comando stem, o qual permite uma representação alternativa de um gráfico discreto, mostrado na figura 15.

```
>> t = [0: 5: 200];
>> f = exp(-0.01*t).*sin(t/4);
>> subplot(2,1,1);
>> plot(t,f);
>> subplot(2,1,2);
>> stem(t,f);
```

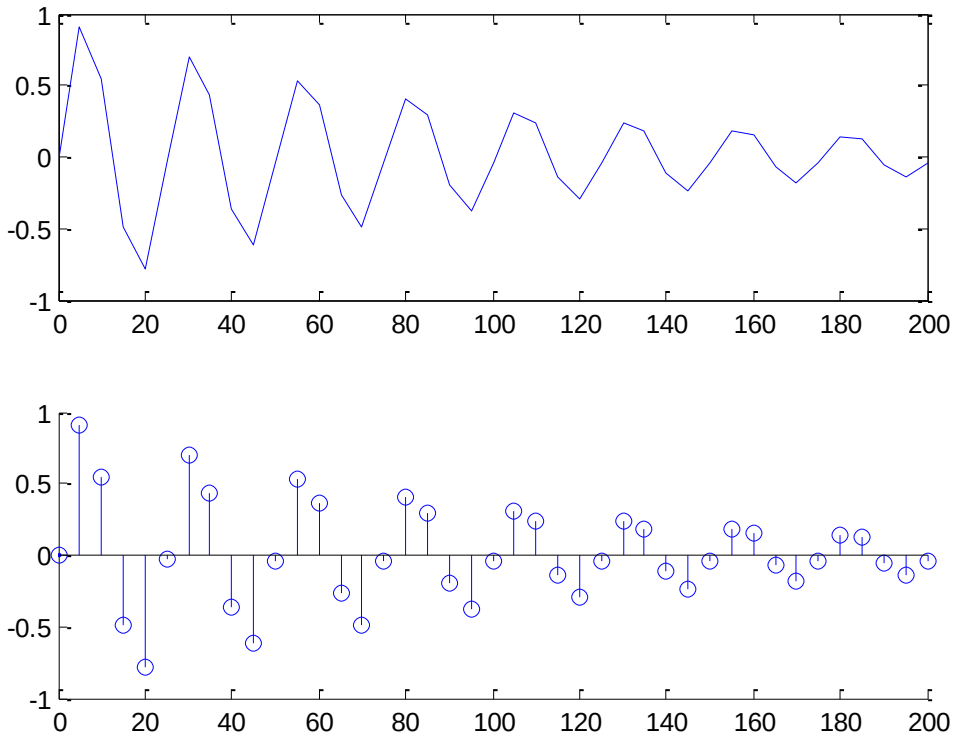

Figura 15 – Representação alternativa com o comando stem.

Outra maneira de representar um gráfico discreto é com o comando step. Por exemplo, a figura 16 apresenta o resultado dos seguintes comandos:

```
>> t = [0: 5: 200];
>> f = exp(-0.01*t).*sin(t/4);
>> stairs(t,f),xlabel('amostras'),ylabel('valores'),title('exp(-0.01*t).*sin(t/4)')
```

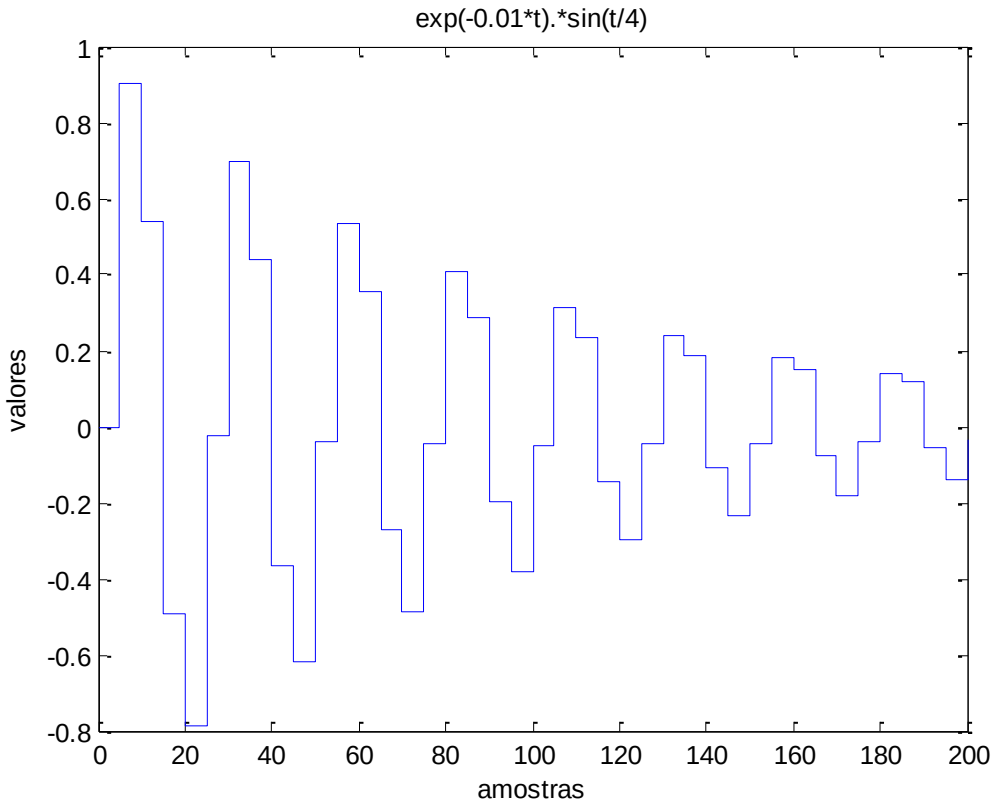

Figura 16 Exemplo do uso do comando stairs.

Gráficos 3D

O comando meshgrid pode ser usado para transformar vetores associados aos eixos x e y de sistemas tridimensionais em estruturas que podem ser usadas pelos comandos de traçado tridimensional. O comando mesh, por sua vez, permite o traçado de gráficos tridimensionais. Por exemplo, a figura 17 pode ser obtida:

```
> [x,y]= meshgrid(0:10,0:10);
>> z = x+y;
>> mesh(x,y,z)
```

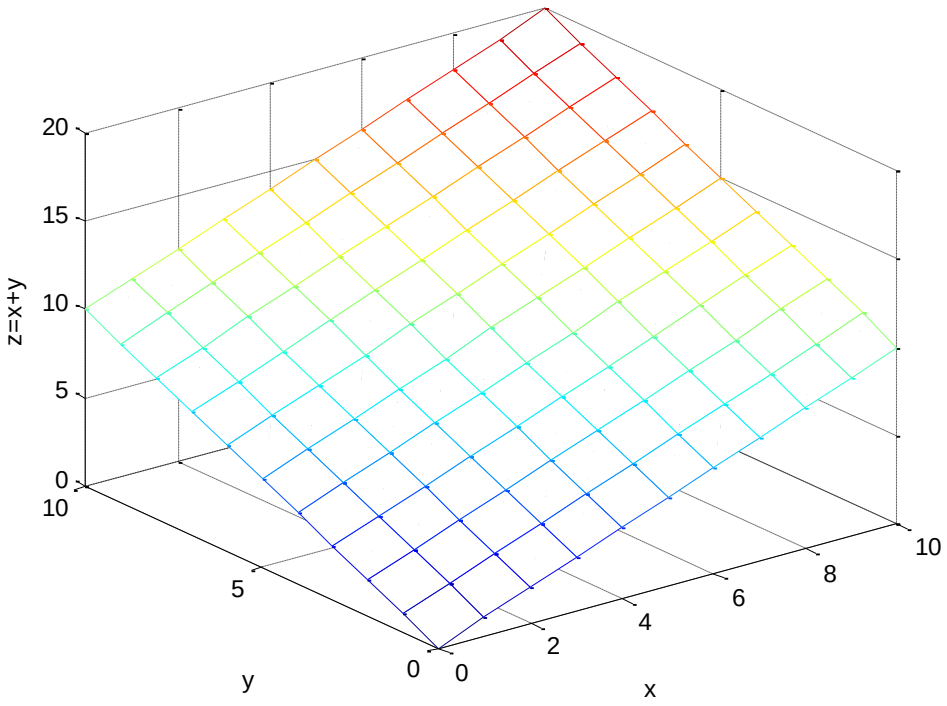

Figura 17 Exemplo de gráfico 3d obtido com os comandos meshgrid e mesh.

O comando surf pode também ser usado para traçar superfícies 3D. A figura 18 compara os resultados do comando mesh e surf, segundo as linhas de comando:

```
>> [x,y]= meshgrid(-2:0.1:2);
>> z = y.*exp(-x.^2-y.^2);
>> subplot(2,1,1)
>> mesh(x,y,z),xlabel('x'),ylabel('y'),zlabel('z')
>> subplot(2,1,2)
>> surf(x,y,z),xlabel('x'),ylabel('y'),zlabel('z')
```

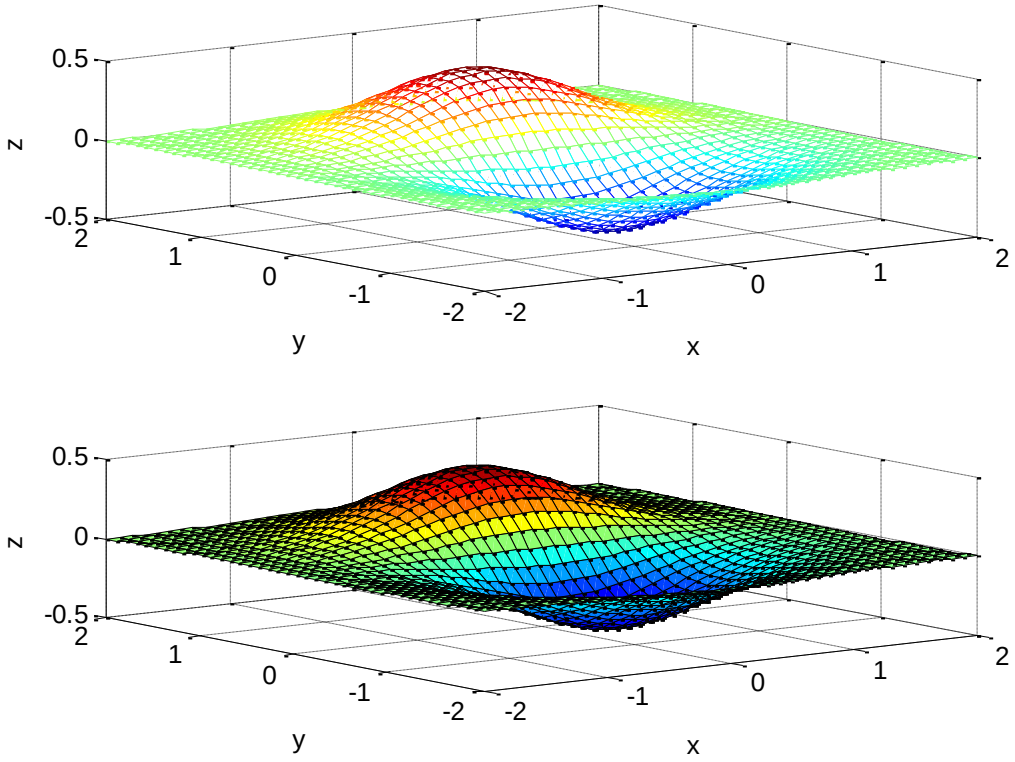

Figura 18 Exemplo de gráfico 3d obtido com o comando mesh (superior) e surf (inferior).

Outro exemplo de gráfico 3D:

```
>> [th,r] = meshgrid((0:5:360)*pi/180,0:.05:1);
>> [X,Y] = pol2cart(th,r);
>> Z = X+i*Y;
>> f = (Z.^4-1).^(1/4);
>> surf(X,Y,abs(f))
>> hold on
>> surf(X,Y,zeros(size(X)))
>> hold off
```

Cujo resultado é mostrado na figura 19.

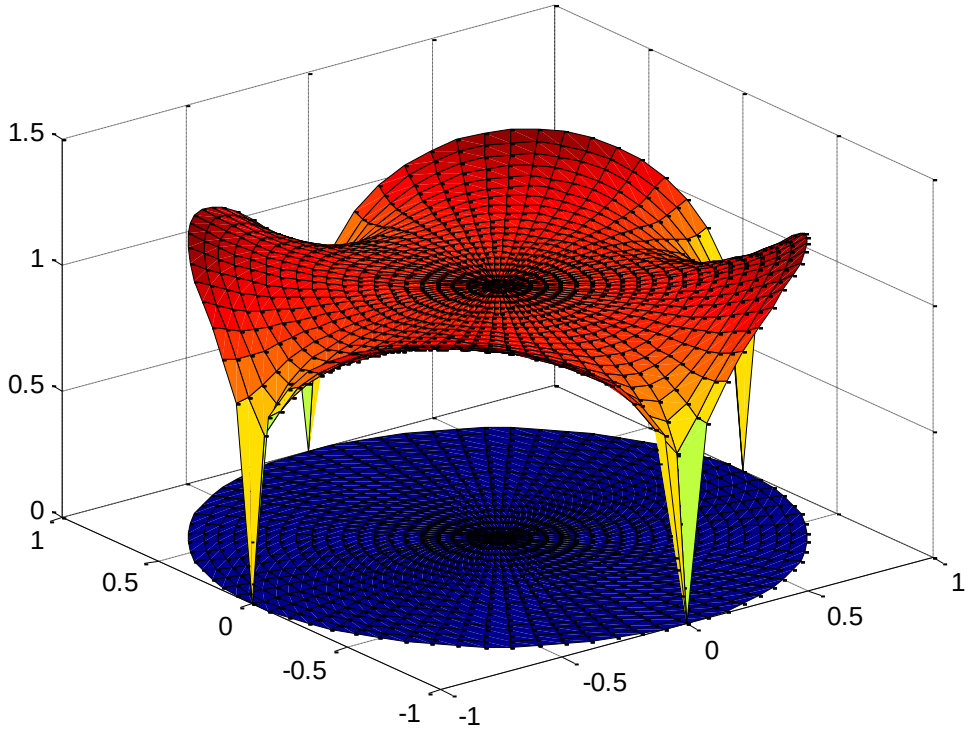

Figura 19 Gráfico 3d composto.

Animações

O Matlab permite que animações sejam usadas para verificar o comportamento matemático de um sistema, por exemplo. Os comandos seguintes usam a idéia de animação para simular o comportamento de uma mola:

```
th=0:pi/60:32*pi;
a=1;
A=0.25;
w=2*pi/15;
M=moviein(16);
for t=1:16;
```

```
x=a*cos(th);
y=a*sin(th);
z=(1+A*cos(w*(t-1)))*th;
plot3(x,y,z,'r');
axis([-2 2 -2 2 0 40*pi]);
M(:,t)=getframe;
end
movie(M,15)
```

O comando moviein apenas inicializa a memória necessária para um conjunto de frames, no caso 16 frames. O laço for (será discutido posteriormente) gera um conjunto de gráficos 3D e armazena nas respectivas posições do vetor M através do comando M(:,t)=getframe. Como resultado o comando movie executa um vetor de frames que representarão o movimento certo número de vezes, no caso, 15.

A figura 20 mostra um dos frames usados no movimento.

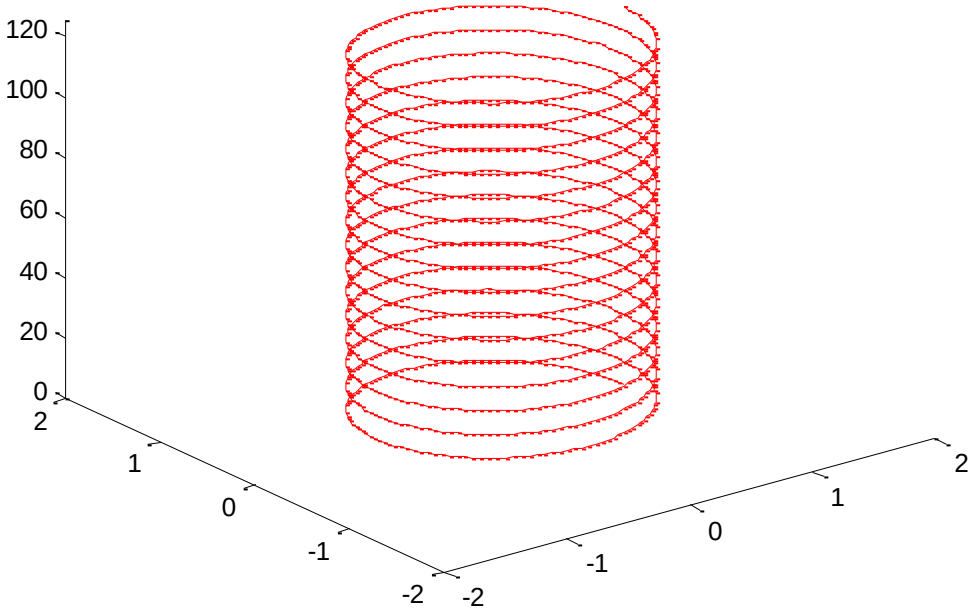

Figura 20 Um dos frames usados na animação.

Exercício: Para a função da figura 18 aplique os seguintes comandos:

```
>> surfl(x,y,z),xlabel('x'),ylabel('y'),zlabel('z')
>> shading interp
```

5 PROGRAMANDO NO MATLAB

Escrevendo Funções

Uma das aplicações dos arquivos .M é a possibilidade de escrever funções.

Por exemplo, através dos passos: Menu File – New M-File é gerada uma nova janela. Nesta janela digite as seguintes linhas:

```
function y = teste1 (x)
a =10
b = 20
(a+b)*x
```

Salve esta nova janela como teste1.m.
Agora, na linha de comandos do Matlab digite:

```
>> teste1(3)
a =
   10

b =
   20

ans =
   90
```

Agora modifique a função teste1:

```
%esta função é um exercício
%retorna em y o valor de (a+b)*x
function y = teste1 (x)
a =10  %entre o valor de a
b = 20 %entre o valor de b
(a+b)*x % calcule
```

Agora digite:

```
>> help teste1
 esta funçao é um exercio
 retorna em y o valor de (a+b)*x
```

Ou seja, é possível documentar uma nova função criada. É sempre bom fazê-lo.

Implementado laços

É possível implementar um laço com a seguinte estrutura:

```
for índice = início: incremento: fim
linhas de comando
end
```

Por exemplo, o programa que soma os valores de um vetor:

```
function soma = teste2 (vetor)
soma = 0;
for i = 1:1:lenght(vetor)
soma = soma + vetor(i);
end
```

Executando:

```
>> v = [1 2 3 4];
>> teste2(v)

ans =
   10
```

Exercício: Crie uma função que calcule a média de valores de um vetor de inteiros.

Usando o while

O comando while usa a seguinte estrutura:

```
while condição
linhas de comando
end
```

Por exemplo, uma função que gere os n primeiros números de Fibonaci:

```
function fibo = teste3 (n)
fibo(1) =1; fibo(2) = 1;
k = 2;
while k<n
fibo(k+1) = fibo(k)+fibo(k-1);
k= k+1;
end
```

Executando:

```
teste3(10)
```

```
ans =
    1    1    2    3    5    8   13   21   34   55
```

Usando switch

O comando switch pode ser usado segundo a estrutura:

```
switch expressão
case 1
        execute este comando
case2
        execute este comando
case 3
        execute este comando
.
.
.
otherwise
        execute este comando
end
```

Uma função que associa um nome a um telefone:

```
function teste4(nome)
switch nome
    case 'carlos'
        12345678
    case 'cristine'
        22222222
    case 'luiz'
        33333333
    Otherwise
```

```
    00000000
End
```

Executando:

```
>> teste4('carlos')

ans =
   12345678

>> teste4('cristine')

ans =
   22222222

>> teste4('luiz')

ans =
   33333333

>> teste4('ana')

ans =
   10000000
```

Usando o if

O comando if pode ser usado segundo a estrutura:

```
if expressão1
   comandos 1
elseif expressão2
   comandos 2
else
   comandos 3
end
```

Assim, uma função que testa se um número é par ou impar:

```
function teste5(numero)
```

```
if rem(numero,2) == 1
        'este número é impar'
elseif rem(numero,2) == 0
        'este número é par'
end
```

Executando:

```
>> teste5(8)

ans =
este número é par

>> teste5(9)

ans =
este número é impar
```

Operadores

Os operadores usados pelo Matlab são:

Operadores relacionais:

<	menor
<=	menor ou igual
>	maior
>=	maior ou igual
==	igual
~=	diferente

Alguns operadores Lógicos

& operador lógico and

| operador lógico or

~ operador lógico not

Os operadores lógicos devem ser usados com cuidado, pois podem gerar resultados inesperados.

Comandos de entrada e saída de dados

Alguns comandos podem ser usados para melhorar a entrada e a saída de dados em uma função. Por exemplo o comando error pode ser usado para interromper a execução de uma função, gerando uma mensagem de erro:

```
% função de soma de vários números com comando error
function s = teste6(a,b,c)
if nargin < 2
   error('pelo menos dois argumento são necessários.')
end
if nargin == 2
s = a+b;
else
s = a+b+c;
end
```

Executando:

```
>> teste6(1,2,3)

ans =
    6

>> teste6(1,2)

ans =
    3

>> teste6(1)
??? Error using ==> teste6
pelo menos dois argumentos são necessários.
```

O comando disp pode ser usado para enviar mensagens enquanto a função está sendo executada:

```
% função de soma de vários números com comando error
function s = teste6(a,b,c)
if nargin < 2
    error('pelo menos dois argumento são necessários.')
end
if nargin == 2
s = a+b; disp(['estou rodando com 2 argumentos'])
else
s = a+b+c; disp(['estou rodando com 3 argumentos'])
end
```

Executando:

```
>> teste6(1,2)
estou rodando com 2 argumentos
ans =
    3

>> teste6(1,2,3)
estou rodando com 3 argumentos
ans =
    6
```

O comando ginput pode ser usado para capturar uma coordenada em um gráfico, se o mesmo existir. A função seguinte demonstra esta aplicação:

```
function teste7
if isempty(get(0,'CurrentFigure'))
   error('Não existe qualquer figura')
end
flag = ~ishold;
if flag
hold on
end
disp('click no ponto em que você deseja colocar um x.')
[x, y] = ginput(1);
plot(x, y, 'x')
if flag
hold off
end
```

Executando:

```
>> x = linspace(0,2*pi);
>> y = sin(x);
>> plot(x,y);
>> teste7
```

O comando eval permite a execução de um comando do Matlab descrito por uma string. Por exemplo, a geração das n (onde n é a ordem da matriz) matrizes identidade podem ser feita:

```
function teste8 (numero)

for n = 1:numero

    eye_str = ['M',int2str(n),' = eye(n)'];
    eval(eye_str)

end
```

Executando:

```
>> teste8(3)

M1 =
    1


M2 =
    1    0
    0    1


M3 =
    1    0    0
    0    1    0
    0    0    1
```

Exercício: Verifique o funcionamento do código seguinte, encontrando as novas funções criadas. Adicione no código os seus comentários sobre o que fazem e como fazem as funções apresentadas.

```
function [t,X,m,c,k,f1,f2,w,x0,v0]= smdplot(example)
 %
 % [t,X,m,c,k,f1,f2,w,x0,v0]= smdplot(example)
 % ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 % This function plots the response and animates the
 % motion of a damped linear harmonic oscillator
 % characterized by the differential equation
 % m*x''+c*x'+k*x=f1*cos(w*t)+f2*sin(w*t)
 % with initial conditions x(0)=x0, x'(0)=v0.
 % The animation depicts forced motion of a block
 % attached to a wall by a spring. The block
 % slides on a horizontal plane which provides
 % viscous damping.

 % example - Omit this parameter for interactive input.
 % Use smdplot(1) to run a sample problem.
 % t,X - time vector and displacement response
 % m,c,k - mass, damping coefficient,
 % spring stiffness constant
 % f1,f2,w - force components and forcing frequency
 % x0,v0 - initial position and velocity
 %
 % User m functions called: spring smdsolve inputv
 % ------------------------------------------------
 pltsave=0; disp(' '), disp(' SOLUTION OF '), disp('M*X" + C*X'' + K*X =
F1*COS(W*T) + F2*SIN(W*T)')
```

```
 disp(' WITH ANIMATION OF THE RESPONSE')
 disp(' ')
 % Example data used when nargin > 0
 if nargin > 0
 m=1; c=.3; k=1; f1=1; f2=0; w=2; x0=0; v0=2;
 tmax=25; nt=250;
 else % Interactive data input
 [m,c,k]=inputv('Input m, c, k (try 1, .3, 1) >> ? ');
[f1,f2,w]=inputv('Input f1, f2, w (try 1, 0, 2) >> ? ');
[x0,v0]=inputv('Input x0, v0 (try 0, 2) >> ? ');
[tmax,nt]=inputv('Input tmax, nt (try 30, 250) >> ? ');
 end
t=linspace(0,tmax,nt);
X=smdsolve(m,c,k,f1,f2,w,x0,v0,t);
% Plot the displacement versus time
plot(t,X,'k'), xlabel('time')
ylabel('displacement'), title('FORCED RESPONSE OF A DAMPED HARMONIC
OSCILLATOR')
grid on, shg, disp(' ')
if pltsave, print -deps smdplotxvst; end
disp('Press return for response animation')
pause
% Add a block and a spring to the displacement
xmx=max(abs(X)); X=X/1.1/xmx;
xb=[0,0,1,1,0,0]/2; yb=[0,-1,-1,1,1,0]/2;
% Make an arrow tip
d=.08; h=.05;
xtip=[0,-d,-d,0]; ytip=[0,0,0,h,-h,0];
% Add a spring and a block to the response
[xs,ys]=spring; nm=length(X); ns=length(xs);
```

```
nb=length(xb); x=zeros(nm,ns+nb);y=[ys,yb];
for j=1:nm, x(j,:)=[-1+(1+X(j))*xs,X(j)+xb];end
xmin=min(x(:)); xmax=max(x(:)); d=xmax-xmin;
xmax=xmin+1.1*d; r=[xmin,xmax,-2,2];
rx=r([1 1 2]); ry=[.5,-.5,-.5]; close;
% Plot the motion
for j=1:nm
% Compute and scale the applied force
f=f1*cos(w*t(j))+f2*sin(w*t(j));
f=.5*f; fa=abs(f); sf=sign(f);
xj=x(j,:); xmaxj=max(xj);
if sf>0
xforc=xmaxj+[0,fa,fa+xtip];
else
xforc=xmaxj+[fa,0,-xtip];
end
% Plot the spring, block, and force
% plot(xj,y,rx,ry,'k',xforc,ytip,'r')
%plot(xj,y,'k-',rx,ry,'k-',xforc,ytip,'k-')
plot(xj,y,'k-',xforc,ytip,'k-',...
rx,ry,'k-','linewidth',1)
title('FORCED MOTION WITH DAMPING')
xlabel('FORCED MOTION WITH DAMPING')
axis(r), axis('off'), drawnow
figure(gcf), pause(.05)
end
if pltsave, print -deps smdplotanim; end
disp(' '), disp('All Done')
%=====================================
function [x,y] = spring(len,ht)
```

```
% This function generates a set of points
% defining a spring
if nargin==0, len=1; ht=.125; end
x=[0,.5,linspace(1,11,10),11.5,12];
 y=[ones(1,5);-ones(1,5)];
y=[0;0;y(:);0;0]'; y=ht/2/max(y)*y;
 x=len/max(x)*x;
 %=====================================
 function [x,v]=smdsolve(m,c,k,f1,f2,w,x0,v0,t)
% [x,v]=smdsolve(m,c,k,f1,f2,w,x0,v0,t)
% ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
% This function solves the differential equation
% m*x''(t)+c*x'(t)+k*x(t)=f1*cos(w*t)+f2*sin(w*t)
% with x(0)=x0 and x'(0)=v0
% m,c,k - mass, damping and stiffness coefficients
% f1,f2 - magnitudes of cosine and sine terms in
 % the forcing function
 % w - frequency of the forcing function
 % t - vector of times to evaluate the solution
 % x,v - computed position and velocity vectors
 ccrit=2*sqrt(m*k); wn=sqrt(k/m);
 % If the system is undamped and resonance will
 % occur, add a little damping
 if c==0 & w==wn; c=ccrit/1e6; end;
 % If damping is critical, modify the damping
 % very slightly to avoid repeated roots
 if c==ccrit; c=c*(1+1e-6); end
 % Forced response solution
 a=(f1-i*f2)/(k-m*w^2+i*c*w);
 X0=real(a); V0=real(i*w*a);
```

```
X=real(a*exp(i*w*t)); V=real(i*w*a*exp(i*w*t));
% Homogeneous solution
r=sqrt(c^2-4*m*k);
s1=(-c+r)/(2*m); s2=(-c-r)/(2*m);
p=[1,1;s1,s2]\[x0-X0;v0-V0];
% Total solution satisfying the initial conditions
x=X+real(p(1)*exp(s1*t)+p(2)*exp(s2*t));
v=V+real(p(1)*s1*exp(s1*t)+p(2)*s2*exp(s2*t));

%=====================================
% function [a1,a2,...,a_nargout]=inputv(prompt)

function varargout=inputv(prompt)
%
% [a1,a2,...,a_nargout]=inputv(prompt)
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
%
% This function reads several values on one line.
% The items should be separated by commas or
% blanks.
%
% prompt - A string preceding the
% data entry. It is set
% to ' ? ' if no value of
% prompt is given.
% a1,a2,...,a_nargout - The output variables
% that are created. If
% not enough data values
% are given following the
% prompt, the remaining
```

```
 % undefined values are
 % set equal to NaN
 %
 % A typical function call is:
 % [A,B,C,D]=inputv('Enter values of A,B,C,D: ')
 %
 %-------------------------------------------
if nargin==0, prompt=' ? '; end
u=input(prompt,'s'); v=eval(['[',u,']']);
ni=length(v); no=nargout;
varargout=cell(1,no); k=min(ni,no);
for j=1:k, varargout{j}=v(j); end
if no>ni
for j=ni+1:no, varargout{j}=nan; end
end
```