MATLAB para o Curso de Mecanismos

Ricardo Cury Ibrahim

(versão 01/2005)

Conteúdo

1 Introdução

Esta apostila tem a intenção de ser um guia rápido para a utilização do MATLAB em atividades das disciplinas PMR2430 e PMR2331, Mecanismos, do curso de graduação da Escola Politécnica da USP. Ela não pretende simplesmente listar os diversos comandos do MATLAB e dar alguns exemplos. O objetivo principal é inserir alguns elementos básicos a partir dos quais se possa progredir de maneira autônoma, fazendo com que o aluno se torne elemento ativo no processo de aprendizagem. No caso do MATLAB, deve-se estudar praticando num micro com o software instalado. Esta apostila foi baseada na versão Matlab6.5 (de junho de 2002), pois era esta a versão mais recente licenciada pela USP no momento em que esta apostila foi escrita.

O MATLAB possui diversos pacotes (toolboxes) específicos para várias áreas, inclusive mecanismos, robótica, controle, e simulador. Desta forma, ele é um ótimo ambiente de trabalho, em que se pode integrar vários resultados.

Existem versões do MATLAB para os mais diversos tipos de sistemas operacionais. Praticamente todos os comandos descritos nesta apostila para serem executados internamente ao ambiente MATLAB devem valer para qualquer plataforma. Entretanto, comandos para ler e salvar arquivos devem ser diferentes. Esta apostila descreverá comandos válidos para o sistema operacional Microsoft Windows, que é o mais utilizado atualmente.

Por fim, gostaria de lembrar que sempre é muito importante consultar livros textos no assunto para referências mais detalhadas.

2 Conceitos Gerais

Esta seção apresenta conceitos básicos do MATLAB para aplicação geral. A assimilação desses conceitos é fundamental para a boa utilização do software, constituindo um ferramental básico de uso frequente.

Recomenda-se a instalação dos seguintes pacotes: MATLAB, Simulink, Control System Toolbox, Optimization Toolbox, SimMechanics, Symbolic Math. Entretanto, todo o conteúdo desta apostila foi baseado no pacote básico MATLAB.

É interessante observar que MATLAB é uma abreviação da junção das palavras inglesas Matrix e Laboratory. Note, então, que o MATLAB existe basicamente para manipular matrizes.

Observação importante: o tipo de fonte exemplificado abaixo será usado sempre que for necessário indicar um comando a ser usado no ambiente MATLAB. Entende-se por ambiente MATLAB a janela para se digitar comandos, que é aberta ao se iniciar o programa. Ao iniciar o programa MATLAB por meio de seu ícone de atalho ou clicando a sequência INICIAR → PROGRAMAS → MATLAB → MATLAB, abre-se uma janela para entrada de comandos. Todos os comandos para execução de alguma atividade devem ser digitados em frente ao prompt >>.

```
>> help help
```

```
>> path
```

```
>> exit
```

O primeiro comando a ser apresentado é também um dos mais importantes: é o comando `help`. Basta digitar `help` seguido do nome de algum comando que se deseja obter informações e apertar a tecla `ENTER`. Experimente com

```
>> help plot
```

Uma forma mais conveniente de usar o `help` do MATLAB é abrindo uma janela própria de `help`. Do menu, escolha **Help** e selecione **MATLAB Help**. Na nova janela aberta pode-se encontrar diversos tópicos de ajuda organizados por um determinado tema. Tente abrir a tabela de `help` para operações elementares com matrizes (`elmat`). Obviamente, uma outra forma de se obter ajuda nesse tema é dar o seguinte comando na janela de comandos:

```
>> help elmat
```

Utilizando o comando `help` sem argumentos fará com que sejam listados todos os tópicos gerais de ajuda dos pacotes instalados.

Outra forma bastante prática de se aprimorar no MATLAB é executar alguns programas demos já incluídos no pacote. Use o comando `help demos` para obter uma listagem dos demos com função geral. Ou use o comando `help simdemos` para uma listagem dos demos do Simulink. Experimente também `help mechdemos`, e descubra quais os demos do simulador SimMechanics Toolbox.

Para rodar um demo basta digitar o seu nome após o prompt, na janela de comando, e seguir as instruções próprias

3 Operações matemáticas simples

Operações matemáticas simples podem ser realizadas diretamente na janela de comandos com o uso dos seguintes caracteres: soma (+), subtração (-), multiplicação (*), divisão (/ ou \), potência (^). Exemplo:

```
>> 3^2
ans =
     9
```

É importante saber qual a ordem de precedência dos operadores mencionados acima. Para tanto, teste você mesmo com vários exemplos para descobrir a ordem. Um dos testes para saber qual a precedência entre `+` e `^` poderia ser calcular `4^2+1`. Note que o uso de parênteses pode alterar a precedência de acordo com a necessidade.

A listagem completa dos operadores pode ser obtida consultando o help de “operators and special characters”, ou dando o comando `help ops`.

A esta altura você já deve ter reparado que muitas vezes o help de algum tópico é muito longo e a informação rola pela tela sem parar. Existe uma maneira de se apresentar a informação de forma que ao se preencher uma tela o help pare e espere por um novo sinal para apresentar a tela seguinte. Basta usar o comando `more`. Tente:

```
>> more on
>> help ops
```

Para retornar ao modo default, de rolagem sem parar, é só usar o comando

```
>> more off
```

É claro que sempre se pode usar o mouse para rolar. . .

Por outro lado, a janela gráfica específica de help, ativada pelo menu, é ótima para quem está começando a trabalhar com Matlab.

4 Armazenando dados em variáveis

Armazenar dados em variáveis é uma maneira muito útil de tratar com operações mais complexas. A associação de um determinado valor a uma variável é bem simples:

```
>> a=2.75

a =

   2.75
```

que associa o valor 2.75 à variável a.

Podemos ter também:

```
>> b=a+4.5

b =

   7.25
```

Se você esquecer e quiser relembrar o valor armazenado numa determinada variável, basta digitar a variável em seguida ao prompt:

```
>> a

a =

   2.75
```

Para saber todas as variáveis já usadas, basta usar o comando `who`, ou abrir a janela "Workspace".

Algumas regras devem ser seguidas para definição de variáveis.

- o nome da variável deve sempre começar por uma letra
- o nome da variável pode conter letras e números, mas não se deve usar os símbolos especiais (`!@#'%^&, etc.`)
- o MATLAB é sensível ao uso de letras maiúsculas e minúsculas. `valor37`, `Valor37` e `VALOR37` são três variáveis diferentes.

Algumas variáveis já são predefinidas e não podem mais ser utilizadas. É o caso, por exemplo, de pi que é a razão do perímetro de uma circunferência pelo seu diâmetro. Note que, excepcionalmente, i ou j representam o número imaginário $\sqrt{-1}$, mas também poder ser usados em variáveis.

Para deixar de usar uma determinada variável numa seção do MATLAB, basta usar o comando `clear` seguido do nome da variável que se deseja excluir da memória.

No MATLAB é possível salvar em arquivo todas as variáveis definidas num determinado trabalho para uso posterior. Para tanto, basta clicar no menu **File** e selecionar **Save Workspace as...** para salvar sua seção de trabalho atual. As variáveis podem ser recuperadas usando a opção **Load** no menu **File**.

5 Formato dos números

O MATLAB pode utilizar números em diversos formatos. Uma listagem completa pode ser obtida usando o comando `help format`.

O formato default é o `format short`, em que números reais são mostrados com 4 algarismos após o ponto decimal.

Para mudar o formato em uso basta dar o comando `format` seguido do nome do formato desejado. Para mudar do formato default para o formato real longo, usa-se o comando:

```
>> format long
```

Como exercício, experimente com os diversos formatos disponíveis.

6 Utilizando funções matemáticas elementares

O MATLAB contém diversas funções matemáticas elementares (trigonométricas, para números complexos, logarítmicas, e numéricas) que são muito úteis em vários tipos de aplicações. Consulte o help para funções matemáticas elementares (uma maneira é usar o comando `help elfun`) para uma listagem completa.

Para o nosso curso, algumas funções podem ser destacadas. Pratique com elas!

Experimente com as seguintes funções:

```
abs(x); conj(x); real(x); imag(x); angle(x); exp(x); log(x); log10(x);
```

```
sqrt(x); rem(x,y); round(x); ceil(x); floor(x); sin(x); asin(x); etc.
```

Entre um número complexo qualquer, como no exemplo abaixo, e experimente com as diversas funções pertinentes (`abs(x)`, `angle(x)`, `conj(x)`, `real(x)`, `imag(x)`)

```
>> nc1 = -4 + 3i

nc1 =

  -4.0000 + 3.0000i
```

Descubra se os argumentos das funções trigonométricas como `sin(x)`, `cos(x)`, `tan(x)` devem ser em radiano ou em grau.

Também é importante para o nosso curso a Expressão de Euler: $e^{i\theta} = \cos(\theta) + i\sin(\theta)$.

No Matlab a função `exp(x)` também pode receber como argumento um número complexo. Por exemplo, para inserir $R_{AO_2}e^{i\theta_2}$, onde $R_{AO_2} = 4$ e $\theta_2 = 30^\circ = pi/6$ basta fazer:

```
>> R=4;

>> T2=pi/6;

>> R*exp(i*T2)


ans =


   3.4641 + 2.0000i
```

7 Listas

Uma lista ou um vetor é uma coleção de dados de um mesmo tipo e unidimensional, ou seja, a cada elemento se pode associar um índice único. A lista é um dos elementos mais úteis no MATLAB.

A maneira mais simples de se criar uma lista no MATLAB é escrevendo seus elementos um a um separados por vírgula ou por espaço dentro de colchetes.

```
>> L1 = [12,exp(1),pi,i]

L1 =

  12.0000            2.7183            3.1416                 0 + 1.0000i
```

ou então,

```
>> L1 = [12 exp(1) pi i]

L1 =

  12.0000            2.7183            3.1416                 0 + 1.0000i
```

Note que esta é uma lista (ou vetor) horizontal. Para criar uma lista vertical (coluna), basta separar seus elementos por ponto e vírgula.

```
>> L1 = [12;exp(1);pi;i]

L1 =

  12.0000

   2.7183

   3.1416

        0 + 1.0000i
```

Ou então, use o sinal de apóstrofo para representar a transposta do vetor linha em seguida ao sinal de ponto:

```
>> L1 = [12 exp(1) pi i].'

L1 =

  12.0000

   2.7183

   3.1416

        0 + 1.0000i
```

É importante observar que se não fosse usado o sinal de ponto antes do apóstrofo o resultado seria o conjugado da transposta, no caso de listas com números complexos. Verifique você mesmo!

Utilizaremos, em seguida, um exemplo muito comum nos livros sobre MATLAB. Imagine que se queira plotar uma determinada função, co-seno por exemplo. Escolhe-se um determinado intervalo de interesse. Diversos elementos compreendidos no intervalo constituem uma lista no eixo horizontal do gráfico a ser plotado. Então, para cada elemento, calcula-se o valor correspondente da função desejada. Essa operação

formará uma outra lista (com mesmo número de elementos) de valores a serem plotados segundo o eixo vertical.

Digamos que se queira plotar a função co-seno para valores compreendidos no intervalo de $-\pi/2$ rad (aproximadamente -1.5708) a 20 rad. Uma maneira de se criar uma lista com, por exemplo, 50 elementos é usando a função `linspace(x1,x2,n)`:

```
>> x=linspace(-pi/2,20,50);
>>
```

Note o sinal de ponto e vírgula no final da linha de comando. Ele foi usado para evitar que os 50 elementos fossem mostrados na tela. Tente usar o mesmo comando sem o sinal de ponto e vírgula e veja o que acontece.

Em seguida, veja como é prático o uso de listas em MATLAB: para calcular o valor da função co-seno para cada elemento da lista basta usar o comando

```
>> y=cos(x);
```

Desta maneira, foi criada uma nova lista y também com 50 elementos.

A cada elemento de uma lista corresponde um índice. O índice do primeiro elemento sempre é 1 (e não zero). O primeiro elemento da lista x ($-\pi/2$) pode ser mostrado com o comando:

```
>> x(1)
ans =
   -1.5708
```

Analogamente, o quinto elemento da lista y é:

```
>> y(5)
ans =
    0.9820
```

Também é possível utilizar a notação de matrizes para se referir a uma lista. O quinto elemento de uma lista horizontal h seria `h(1,5)`; enquanto que o quinto elemento de uma lista vertical v seria `v(5,1)`.

Uma outra maneira de se criar uma lista, em que se tem o primeiro e o último elemento e o intervalo constante de variação entre elementos subsequentes, será mostrada em seguida. Vamos criar uma lista $L2$ de números reais entre 1 e 0 com espaçamento -0.1. Isto é feito da seguinte forma:

```
>> L2 = 1:-0.1:0
L2 =
  Columns 1 through 7
    1.0000    0.9000    0.8000    0.7000    0.6000    0.5000    0.4000
  Columns 8 through 11
    0.3000    0.2000    0.1000         0
>>
```

7.1 Operações matemáticas com listas

A grosso modo, as operações podem ser divididas em operações com escalares (um valor numérico simples) e operações entre listas.

Uma lista pode ser multiplicada, dividida, adicionada ou subtraída por/de um escalar de uma maneira bem simples. Tomemos a lista $L2$ definida anteriormente.

```
>> 10*L2-1

ans =

  Columns 1 through 7

    9.0000    8.0000    7.0000    6.0000    5.0000    4.0000    3.0000

  Columns 8 through 11

    2.0000    1.0000         0   -1.0000

>>
```

Por outro lado, as operações matemáticas entre listas são realizadas elemento a elemento. Por exemplo,

```
>> L3=0:0.1:1

L3 =

  Columns 1 through 7

         0    0.1000    0.2000    0.3000    0.4000    0.5000    0.6000

  Columns 8 through 11

    0.7000    0.8000    0.9000    1.0000

>> L2-L3

ans =

  Columns 1 through 7

    1.0000    0.8000    0.6000    0.4000    0.2000         0   -0.2000

  Columns 8 through 11

   -0.4000   -0.6000   -0.8000   -1.0000

>> L2./L3   %divisao de L2 por L3 elemento a elemento

Warning: Divide by zero.

ans =

  Columns 1 through 7

       Inf    9.0000    4.0000    2.3333    1.5000    1.0000    0.6667

  Columns 8 through 11

    0.4286    0.2500    0.1111         0

>> L2.\L3   %divisao de L3 por L2 elemento a elemento
```

```
Warning: Divide by zero.
ans =
  Columns 1 through 7
         0    0.1111    0.2500    0.4286    0.6667    1.0000    1.5000
  Columns 8 through 11
    2.3333    4.0000    9.0000       Inf
>> L2.*L3   %multiplicacao de L2 por L3 elemento a elemento
ans =
  Columns 1 through 7
         0    0.0900    0.1600    0.2100    0.2400    0.2500    0.2400
  Columns 8 through 11
    0.2100    0.1600    0.0900         0
```

Repare, entretanto, nas seguintes operações:

```
>> L2*L3   %tentativa de multiplicacao de uma matriz (1x11) por outra (1x11)
??? Error using ==> *
Inner matrix dimensions must agree.
>> L2*L3.'  %multiplicacao de uma matriz (1x11) por outra (11x1)
ans =
    1.6500
```

Note a maneira de se executar uma multiplicação ou uma divisão entre listas. Deve-se usar o sinal de ponto (.) antes do símbolo da operação de multiplicação ou divisão.

Operações entre listas só podem ser executadas se as listas forem de mesma dimensão (mesmo número de elementos).

Repare a diferença entre a divisão direita (/) e a divisão esquerda (\).

Note também os resultados e os avisos de divisão por zero.

8 Matrizes

A maneira mais simples de se criar uma matriz em MATLAB é inserindo seus elementos um a um, como uma lista, mas separando cada linha da matriz por ponto e vírgula:

```
>> format bank
>> M1=[3 49 pi; 2^5,0.87,0.2-10*i;100 200 200]
M1 =
```

```
    3.00        49.00         3.14
   32.00         0.87         0.20
  100.00       200.00       200.00
```

Repare que o formato dos números (ver seção 5) foi modificado para *bank*. O que aconteceria se a mesma matriz fosse definida com o formato default do MATLAB (*short*)?

Um elemento de uma matriz bidimensional pode ser referenciado através de seus dois índices:

```
>> M1(2,3)
ans =
    0.20
```

As listas (vetores) vistas anteriormente podem ser consideradas um caso particular de matriz unidimensional.

Em geral, as matrizes bidimensionais em MATLAB podem ser consideradas como sendo formadas por linhas (ou colunas) de listas. Por exemplo, sejam dadas duas listas definidas abaixo:

```
>> format
>> Lis1=linspace(1,10,10)
Lis1 =
     1     2     3     4     5     6     7     8     9    10
>> Lis2=linspace(0.1,1,10)
Lis2 =
  Columns 1 through 7
    0.1000    0.2000    0.3000    0.4000    0.5000    0.6000    0.7000
  Columns 8 through 10
    0.8000    0.9000    1.0000
```

Pode-se facilmente criar matrizes com estas listas:

```
>> Mat1=[Lis1;Lis2]
Mat1 =
  Columns 1 through 7
    1.0000    2.0000    3.0000    4.0000    5.0000    6.0000    7.0000
    0.1000    0.2000    0.3000    0.4000    0.5000    0.6000    0.7000
  Columns 8 through 10
    8.0000    9.0000   10.0000
    0.8000    0.9000    1.0000
>> Mat2=[Lis1.' Lis2.']
```

```
Mat2 =
    1.0000    0.1000
    2.0000    0.2000
    3.0000    0.3000
    4.0000    0.4000
    5.0000    0.5000
    6.0000    0.6000
    7.0000    0.7000
    8.0000    0.8000
    9.0000    0.9000
   10.0000    1.0000
```

É possível referenciar todos os elementos de uma linha ou coluna utilizando-se o sinal de dois pontos (:),

```
>> Mat1(2,:)
ans =
  Columns 1 through 7
    0.1000    0.2000    0.3000    0.4000    0.5000    0.6000    0.7000
  Columns 8 through 10
    0.8000    0.9000    1.0000
>> Mat2(5,:)
ans =
    5.0000    0.5000
>> Mat1(:,7)
ans =
    7.0000
    0.7000
>> Mat2(:,1)
ans =
     1
     2
     3
     4
     5
```

```
     6
     7
     8
     9
    10
```

Extendendo ainda mais o conceito de criação de matrizes, uma matriz também pode ser formada de outra matriz,

```
>> Mat3=[Mat1;Lis1]
Mat3 =
  Columns 1 through 7
    1.0000    2.0000    3.0000    4.0000    5.0000    6.0000    7.0000
    0.1000    0.2000    0.3000    0.4000    0.5000    0.6000    0.7000
    1.0000    2.0000    3.0000    4.0000    5.0000    6.0000    7.0000
  Columns 8 through 10
    8.0000    9.0000   10.0000
    0.8000    0.9000    1.0000
    8.0000    9.0000   10.0000
```

A dimensão de uma matriz pode ser obtida com a função `size(M)`.

```
>> size(Mat2)
ans =
    10     2
```

ou seja, a matriz *Mat2* tem dimensão $10x2$.

A matriz identidade pode ser criada com a função `eye`

```
>> Ident3=eye(3)
Ident3 =
     1     0     0
     0     1     0
     0     0     1
```

8.1 Operações matemáticas com matrizes

As operações com matrizes são bem convenientes com o MATLAB. Uma lista de todas as operações pode ser obtida com o comando `help matfun`. Veja alguns exemplos:

```
>> det(M1)
ans =
-2.9239e+005- 4.3000e+004i
>> inv(M1)
ans =
  -0.0014 - 0.0066i   0.0307 - 0.0045i   0.0002 + 0.0016i
   0.0219 + 0.0002i  -0.0010 + 0.0001i  -0.0003 - 0.0001i
  -0.0211 + 0.0031i  -0.0144 + 0.0021i   0.0052 - 0.0008i
>> Mat1*Mat2
ans =
  385.0000   38.5000
   38.5000    3.8500
>> Mat1.*Mat2
??? Error using ==> .*
Matrix dimensions must agree.
>> Mat1.*Mat2.'
ans =
  Columns 1 through 7
    1.0000    4.0000    9.0000   16.0000   25.0000   36.0000   49.0000
    0.0100    0.0400    0.0900    0.1600    0.2500    0.3600    0.4900
  Columns 8 through 10
   64.0000   81.0000  100.0000
    0.6400    0.8100    1.0000
```

9 Loops e tomadas de decisão

O que veremos nesta seção pode ser bastante útil em diversas situações. Loops podem ser usados para criar listas, executar diversas operações de forma repetitiva, etc. Tomadas de decisão podem ser utilizadas na criação de suas próprias funções, conforme será visto mais adiante. A utilização dessas ferramentas no MATLAB é bastante semelhante ao uso em linguagens de programação populares.

9.1 Loops usando for

O `for` pode ser usado quando se quer repetir certas operações um número bem definido de vezes, atribuindo os valores de uma lista a uma variável. Isto pode ser entendido mais facilmente através de um exemplo.

Para se criar uma lista que tenha ordem de formação mais complexa do que o visto na seção 7, podemos usar o comando `for`

```
>> x=2;
>> y=5;
>> for k=1:3
     for l=1:3
     M1(k,l)=sin(x);
     x=y+x;
     y=y/k;
     end
end
>> M1
M1 =
    0.9093    0.6570   -0.5366
   -0.9614   -0.0089   -0.5914
    0.5788    0.9465    0.9928
```

Note que as operações compreendidas entre um par *for end* são executadas em cada iteração. Descubra o por quê do uso do sinal de ponto e vírgula no final das expressões. O que aconteceria se não fosse usado o ponto e vírgula?

9.2 Loops usando while

O comando `while` é usado de forma análoga ao comando `for`. A diferença fundamental é que com o uso do `while` os comandos são repetidos até que se satisfaça uma determinada condição. Veja no exemplo a seguir.

```
>> v1=5;
>> n=1;
>> while v1>0.5
L(n)=n*v1;
n=n+1;
v1=v1-v1/2;
end
>> L
L =
    5.0000    5.0000    3.7500    2.5000
```

9.3 Tomadas de decisão usando if-else

É usado de forma análoga a outras linguagens de programação. Não há muito o que explicar já que deve ser do conhecimento de todos. Repare que deve terminar com o comando `end`. Exemplo:

```
>> for m=1:3
     for n=1:3
        if m==n
           M2(m,n)=1;
        elseif m<n
           M2(m,n)=2;
        else
           M2(m,n)=0;
        end
      end
end
>> M2
M2 =
     1     2     2
     0     1     2
     0     0     1
```

Note que se houver apenas duas possibilidades a verificar deve-se usar *if-else-end*, enquanto que se houver três ou mais possibilidades a verificar deve-se usar *if-elseif-...-elseif-else-end*.

10 Plotando gráficos

O MATLAB é bastante versátil para criação de gráficos em diversos estilos (bidimensional, tridimensional, barras, polar, etc.).

Vamos plotar a função co-seno com listas x e y. Para gráficos simples bidimensionais deve-se usar a função `plot`.

```
>> x=linspace(-pi/2,20,50);
>> y=cos(x);
>> plot(x,y)
>>
```

Logo em seguida a esse comando, deve ser aberta uma nova janela gráfica com o gráfico da função desejada. Para se obter o gráfico com grades para facilitar a leitura, use a função `grid on`. Note que o MATLAB ajusta os eixos automaticamente.

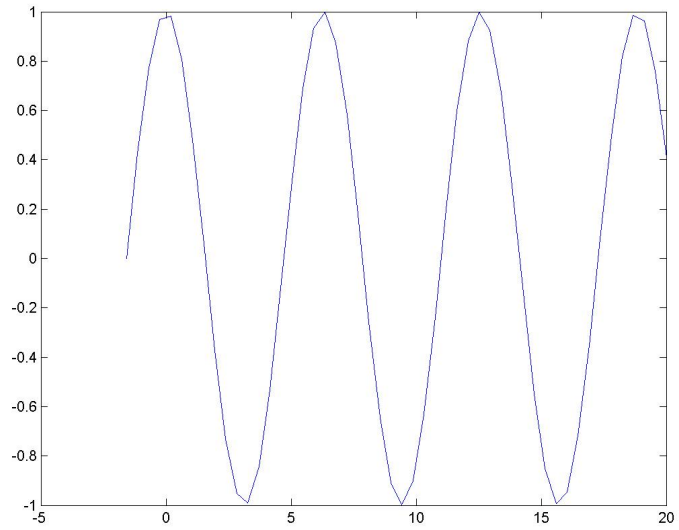


Figura 1: Exemplo de gráfico simples da função co-seno.

Como salvar o gráfico num formato que possa ser inserido por um editor de textos? Na janela do gráfico, é só clicar no menu *File* → *Export...* e escolher o formato desejado. Ou, se preferir trabalhar na janela de comandos, use a função `print` adequadamente. Dê um `help print`.

Para adicionar títulos aos eixos e ao gráfico, siga o modelo abaixo:

```
>> xlabel('Eixo x')
>> ylabel('Eixo y')
>> title('Gráfico cos(x)')
```

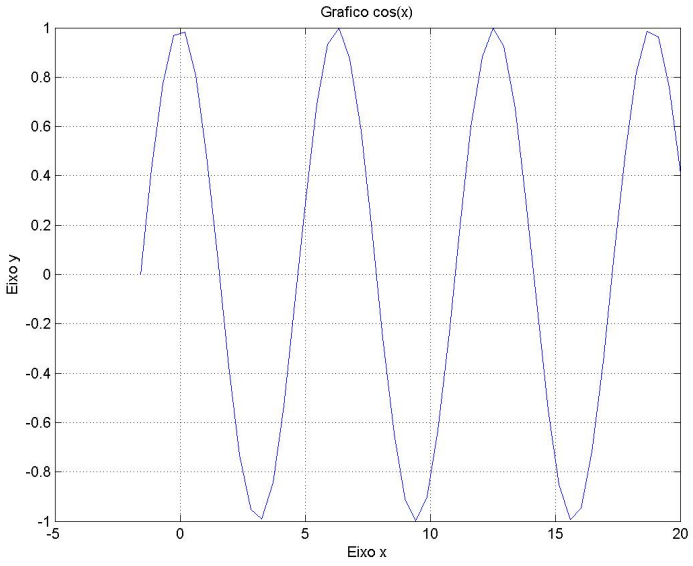


Figura 2: Exemplo de gráfico simples da função co-seno com títulos.

Essas e muitas outras edições do gráfico podem ser feitas mais facilmente na janela do próprio gráfico utilizando o menu Edit ou Insert.

Para plotar mais de uma curva ao mesmo tempo:

```
>> plot(x,y,x,sin(x))
```

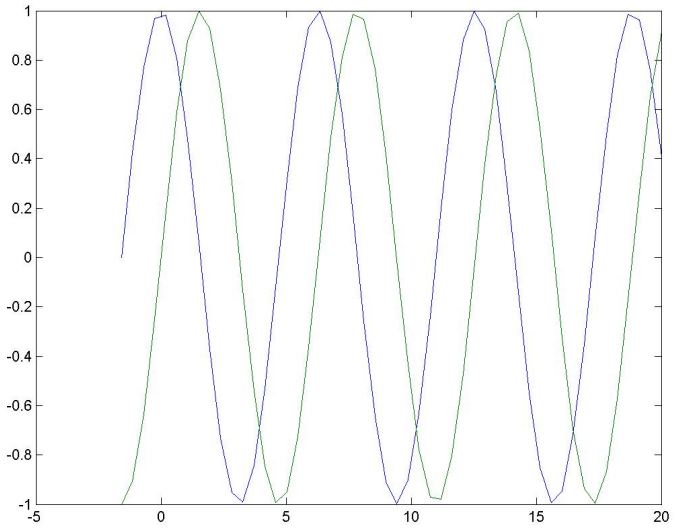


Figura 3: Exemplo de duas funções (co-seno e seno) plotadas no mesmo gráfico.

Outras opções da função `plot`, tais como cor ou tipo da linha do gráfico, símbolos dos pontos plotados, podem ser vistas no `help` correspondente. Procure também alguma informação sobre legendas (`help legend`).

11 Trabalhando com arquivos de dados

O Matlab possui diversas funções para importar ou exportar dados de/para arquivos. Apesar de o Matlab permitir o uso de diversos formatos de arquivos, as formas mais usuais são o formato txt (ascii) e o formato binário próprio do Matlab. Geralmente, o mais conveniente é salvar em formato txt pois poderá ser utilizado por outros programas.

Vamos utilizar um exemplo simples para gerar várias listas de dados e exporta-las na forma de colunas para um arquivo. As listas são as seguintes:

(Lembre que todo texto colocado após o símbolo % serve apenas como comentário/documentação).

```
>> clear  %destrói todas as variáveis que estejam em uso.
>> x=[0:20*pi/400:20*pi]'; %note que foi usado ' para gerar uma coluna (transposta)
>> A=cos(x);
>> B=exp(-x/20);
>> for i=1:length(x)
```

```
>>     C(i)=B(i)*A(i);
>> end
>> C=C'; %foi usado ' para transformar na forma de coluna
>> D=-B;
```

Podemos plotar as listas geradas em função de x com o comando plot(x,A,x,B,x,C,x,D):

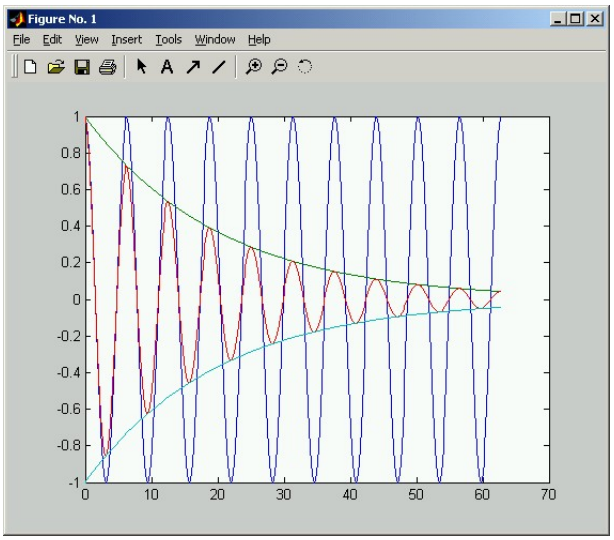


Figura 4: Gráfico com os dados criados.

O comando mais simples para salvar os valores das variáveis armazenadas é o `save`. Entretanto, este comando salva no formato binário, num arquivo com extensão `mat`. Como exemplo vamos salvar as variáveis x, A e B num arquivo chamado `dados1.mat` (note que é importante sempre utilizar a extensão `.mat` para que os dados possam ser importados em outra ocasião). Após selecionar o diretório onde o arquivo será salvo (current directory), dê o comando:

```
>> save dados1 x A B
```

Para importar esses dados e as variáveis correspondentes basta usar o comando load:

```
>> load dados1.mat
```

Com o comando `length` é possível determinar que a quantidade de dados de cada lista é 401. Para salvar todas essas 5 listas num arquivo `dados2.dat` de formato ascii simples podemos usar o comando `dlmwrite`:

```
>> dlmwrite('dados2.dat',[x A B C D],'\t')
```

Note que as variáveis foram reunidas numa única matriz `[x A B C D]` pois o comando dlmwrite é para exportar matrizes. `'\t'` é usado para separar os dados com um tab. Se não for usado `'\t'` o default será a separação por uma vírgula.

Para importar dados de um arquivo ascii é só usar o comando `dlmread`. Por exemplo, para importar os dados do arquivo `dados2.dat` recém criado numa matriz M é só usar o comando:

```
>> M=dlmread('dados2.dat')
```

Para atribuir as variáveis originais x, A, B, C, D a cada coluna:

```
>> x=M(:,1)
>> A=M(:,2)
>> B=M(:,3)
>> C=M(:,4)
>> D=M(:,5)
```

12 Arquivos de Roteiro - Arquivos M

O Matlab permite o uso de arquivos com extensão `m` para elaboração de funções e roteiros (scripts). Desta forma, qualquer usuário pode criar sua própria função, ou arquivar uma seqüência de comandos. Com isso consegue-se maior praticidade e economia de tempo em operações longas e/ou repetitivas.

Os arquivos M devem ter formato txt (ascii). Qualquer editor de texto que salva em formato txt pode ser utilizado. Entretanto, o próprio Matlab possui um editor de textos integrado que é mais conveniente por distinguir diversas operações com cores diferentes e por fazer verificação automática de sintaxe de programação (por exemplo, abertura e fechamento de parênteses).

É importante colocar a pasta onde o roteiro foi salvo na lista de caminhos ativos do Matlab para que ele possa ser utilizado. Caso contrário o Matlab responderá com uma mensagem de erro dizendo não ter encontrado nenhum arquivo ou função com esse nome.

Como exemplo, vamos considerar a ação de se importar uma série de dados na forma de tabela que estão armazenados em arquivos, e que se deseje plotar os gráficos correspondentes. Se houver vários arquivos e/ou várias colunas de dados a plotar por arquivo, será mais conveniente elaborar um arquivo M do que digitar cada comando repetitivamente na janela principal do Matlab. Vamos usar o arquivo `dados2.dat` criado na seção anterior. O arquivo com nome `roteiro1.m` preparado no editor do Matlab é mostrado na figura 5.

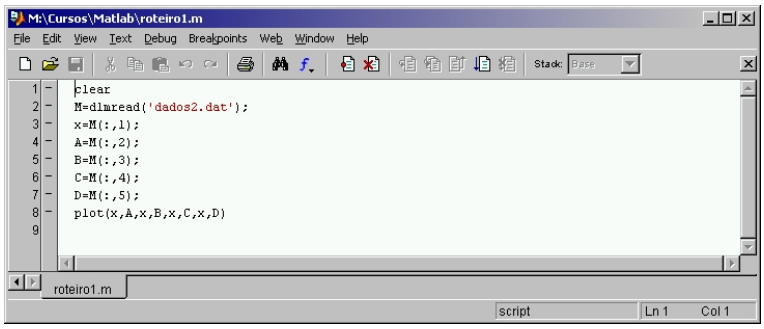


Figura 5: Roteiro para importação de dados do arquivo dados2.dat.

Para rodar este arquivo basta dar o comando `roteiro1` na janela de comandos do Matlab (não se esqueça de verificar se o diretório em que foi salvo o arquivo `roteiro1.m` está na lista de caminhos ativos).

13 Criando funções

O usuário pode criar suas próprias funções para facilitar seu trabalho. Basta seguir a sintaxe adequada. É importante notar que as variáveis internas de uma função são apenas locais, ou seja, não são utilizáveis na janela de comando; a não ser que a variável seja retornada como resultado pela função.

De uma forma geral, a primeira linha do arquivo deve começar com a declaração da função. A função deve ser criada como se fosse um arquivo roteiro (seção 12). Vários exemplos são dados a seguir:

```
function funcao1(p1,p2,p3) %declara a função de nome funcao1, com
%três argumentos de entrada (p1,p2,p3), e que não retorna nada.
```

```
function []=funcao1(p1,p2,p3) %declara a função de nome funcao1, com
%três argumentos de entrada (p1,p2,p3), e que não retorna nada.
```

```
function y=funcao2(p1,p2) %declara a função de nome funcao2, com duas
%entradas (p1 e p2) e uma saída y
```

```
function [y1,y2,y3]=funcao3(p1,p2) %declara a função de nome funcao3, com duas
%entradas (p1 e p2) e três saídas y1, y2 e y3
```

A função criada deve ser salva com o mesmo nome, acrescentando a extensão `m`. Por exemplo, a primeira função definida anteriormente deverá ter seu arquivo salvo com o nome `funcao1.m`. Deve-se tomar cuidado para não utilizar nomes de funções já existentes no Matlab.

Em seguida, podem ser inseridas informações de uso da função sob a forma de comentários (%). Toda essa informação pode ser acessada da janela principal de comandos do Matlab através do comando `help nome_função`. Para a primeira função definida podemos ter:

```
function funcao1(p1,p2,p3)
%FUNCAO1 executa uma determinada ação
%A ação utiliza os argumentos de entrada nas formas: p1 é um número
%inteiro positivo; p2 é um vetor; p3 é um número complexo.
```

O corpo da função, contendo todos os comandos a serem executados, vem após as linhas de informação. Comentários podem ser colocados em qualquer parte utilizando o símbolo %.

Exemplo 1

A função `exemplo1.m`, na figura 6, faz a conversão de valores em polegadas para metros, ou de metros para polegadas.

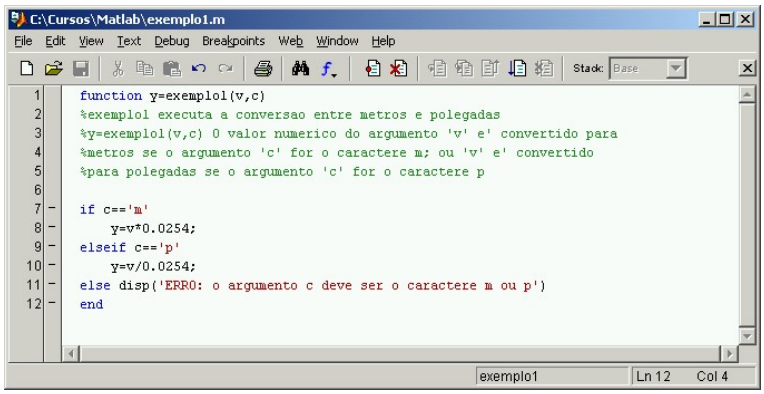


Figura 6: Exemplo de função.

Veja vários exemplos de utilização desta função na figura 7.

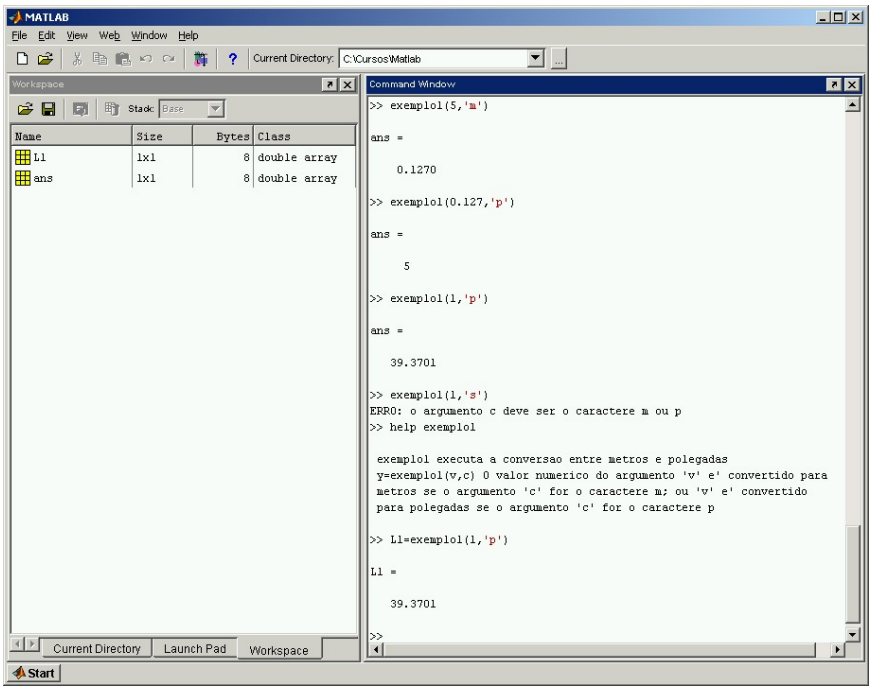


Figura 7: Execução da função exemplo1 na janela de comando do Matlab.

Exemplo 2

Neste segundo exemplo será visto como passar uma expressão literal (string) para dentro de uma função e como converter essa string em comandos executáveis pelo Matlab.

Suponha que seja necessário realizar o cálculo de uma função complexa para um número grande de valores, por exemplo, de 0 a 20 em variações de 0.50. Seja a função $y = 2x^{3.15} + \cos(x) + e^{-4x}$ e sua derivada $\frac{dy}{dx} = 6.30x^{2.15} - \sin(x) - 4e^{-4x}$.

Neste caso, será criada uma função que receba as duas expressões literais (duas variáveis tipo `string` como argumento) e o valor da variável `x`.

O Matlab possui o comando `eval` que converte uma string em comandos executáveis. A função poderá ser definida com os comandos seguintes:

```
function [y,Dy]=funcao2(Y,D,v)
%Funcao2 para retornar valores de uma funcao e sua derivada
%[y,Dy]=funcao2(Y,D,x) retorna o valor numerico da funcao y e de sua
%derivada Dy, recebendo como argumentos a funcao y na sua forma literal Y,
%a derivada de y na forma literal D, e o valor x da variavel.
m=size(v);
for k=1:m(2)
    x=v(k);
    y(k)=eval(Y);
    Dy(k)=eval(D);
end
```

Neste caso, é mais conveniente criar um arquivo roteiro (script) para entrada dos dados. Foi criado o arquivo `entrada.m` listado a seguir:

```
%arquivo roteiro para exemplificar o uso da funcao2.m
Y='2*x^(3.15)+cos(x)+exp(-4*x)';
D='6.30*x^(2.15)-sin(x)-4*exp(-4*x)';
x=[0:0.5:20];
[yy,dy]=funcao2(Y,D,x);
plot(x,yy,'o-',x,dy,'+-')
legend('y(x)','dy/dx');
xlabel('x');
```

Na janela de comando do Matlab deve-se colocar o diretório desses dois arquivos na lista de caminhos ativos. Então, basta dar o comando `entrada`. Será plotado o gráfico da figura 8.

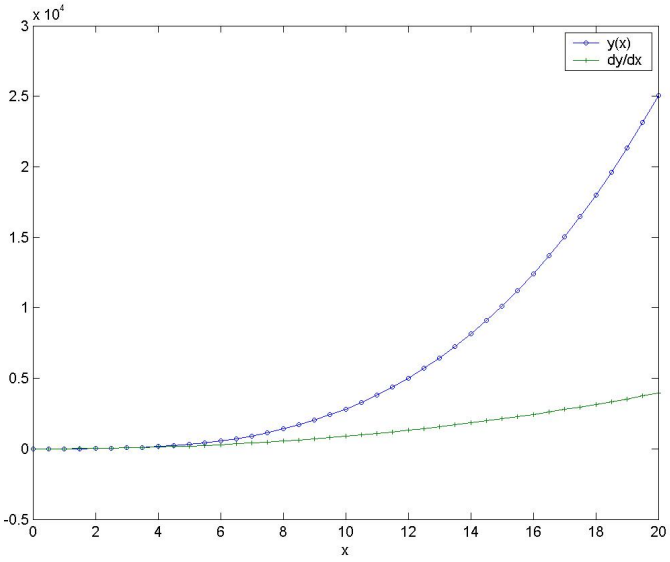


Figura 8: Execução do arquivo roteiro entrada.m gerando a plotagem das expressões calculadas na funcao2.m.

14 Gráficos avançados

Nesta seção serão vistos alguns recursos adicionais para plotagem de gráficos. É importante notar que a janela gráfica criada com o gráfico através do comando `plot`, visto na seção 10, possui opções de menu que podem editar diversas propriedades de plotagem de maneira bem prática. Entretanto, quando se necessita criar vários gráficos complexo é mais conveniente editar os gráficos por comandos em arquivos de roteiro.

Aqui serão vistos apenas alguns comandos considerados mais importantes. Para maiores detalhes, deve-se consultar o `help` de `Graphics`.

É possível abrir uma janela gráfica com o comando `figure(n)`, onde `n` é o número da janela.

O Matlab permite nomear gráficos e imagens para manipulação. A sequência seguinte mostra um caso prático para controlar várias propriedades dos gráficos. É mais conveniente ir digitando direto na janela de comando do Matlab e ir observando o resultado na janela gráfica.

```
%sequencia de comandos para graficos avancados

clear %limpa todas as variaveis armazenadas

figure(2) %abre uma janela grafica vazia de numero 2

close(2)%fecha a janela numero 2 recem-aberta (e' so' para aplicar esses comandos)

figure(3) %abre a janela vazia numero 3

x=[-10:0.5:10]; %define uma lista para o eixo x

h=plot(x,sin(x))  %nomeia a plotagem como h e plota na janela atual (3)

axis([-15 12 -2 3]);  %define os limites dos eixos: axis([xmin xmax ymin ymax])

%ver help do comando axis

a=axis  %para visualizar os limites atuais dos eixos
```

```
axis off %para desabilitar a visualizacao dos eixos e suas propriedades
axis on  %para habilitar a visualizacao dos eixos e suas propriedades
%definidas anteriormente
axis equal  %iguala os limites dos eixos
axis(a)  %retorna aos limites de eixos anteriores guardados na variavel a. Mas,
%note que o tamanho do grafico foi alterado
axis normal  %ocupa o maximo espaco possivel da janela
axis auto  %ajusta automaticamente os limites do grafico
axis(a)
axis ij%inverte os limites do eixo y (inicia com o ymin no canto esquerdo superior)
axis xy  %retorna o eixo y
axis square  %da' um formato quadrado ao grafico, mantendo os limites da variavel a.
axis normal  %retorna ao formato que ocupa o maximo espaco da janela
axis 'auto x'  %ajusta automaticamente apenas o limite do eixo x
axis manual  %controle manual de eixos
hold on  %mantem o grafico plotado atual
plot(x,cos(x),'Color','k')  %acrescenta o grafico cos(x) com a linha em cor codigo
%[R G B]=[0.1 0.9 0.8] ao grafico anterior. Os tres parametros RGB podem
%assumir valores entre 0 e 1.
c=plot(x,cos(x),'k:') %desenha novamente a funcao cos(x), mas em linhas pontilhadas
%e com a cor preta. Note que a linha cos(x) anterior continua presente.
%A variavel c esta nomeando esta nova linha.
set(c,'LineWidth',3)  %modifica a largura da linha c. Veja o help do comando set
set(c,'LineStyle','none','Marker','d',...  %... indica que o comando continua
    'MarkerFaceColor','r','LineWidth',1,'MarkerSize',8)
%modifica para a linha c: LineStyle=none nao plota linha; Marker=d marcas
%em formato de diamante; MarkerFaceColor=r preenche as marcas com a cor
%vermelha; LineWidth=1 reduz a linha para largura 1 (no caso a unica linha
%ativa c e' o marcador); MarkerSize=8 tamanho do marcador 8.
```

15 Animações

Nesta seção serão vistos alguns recursos importantes para gerar animações de gráficos. Estes recursos poderão ser usados em simulações animadas de mecanismos. É importante ter lido a seção 14.

Considere um elo (corpo rígido) preso por uma junta de revolução e que realiza rotação pura, figura 9. Serão definidos apenas 3 pontos deste elo: origem (O) e dois pontos particulares de interesse (A e P).

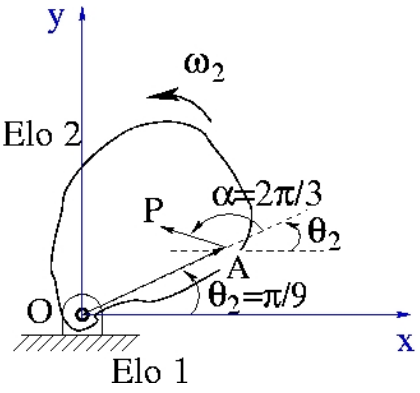


Figura 9: Mecanismo plano em rotação pura em torno do ponto O.

O roteiro `rotp1.m` mostra como gerar uma animação desse mecanismo sem manter os quadros anteriores, ou seja, criando e apagando quadros em sequência.

De outra forma, o roteiro `rotp2.m` mostra como gerar uma animação do mecanismo mantendo as posições das sequências anteriores. Há poucas, mas importantes, modificações entre os dois roteiros.

```
%rotp1.m
%roteiro para criar animacao de um corpo rigido em rotacao pura no plano XY
%ESTA ANIMACAO NAO MANTEM OS QUADROS ANTERIORES
clear  %limpa da memoria todas as variaveis
close  %fecha todas as janelas graficas abertas
figure(3);  %abre a janela grafica numero 3
R_AO=5;  %distancia entre os pontos A e O do elo 2
R_PA=2;  %distancia entre os pontos P e A do elo 2
T2_0=pi/9; %angulo theta 2 inicial da linha entre os pontos O e A
Alpha=2*pi/3;  %angulo fixo entre as linhas PA e AO
Ax_0=R_AO*cos(T2_0);  %componente x do ponto A inicial
Ay_0=R_AO*sin(T2_0);  %componente y do ponto A inicial
Px_0=Ax_0+R_PA*cos(T2_0+Alpha); %componente x do ponto P inicial
Py_0=Ay_0+R_PA*sin(T2_0+Alpha); %componente y do ponto P inicial
Lx=[0, Ax_0, Px_0, 0]; %coordenadas em x da sequencia de pontos a plotar
Ly=[0, Ay_0, Py_0, 0]; %coordenadas em y da sequencia de pontos a plotar
```

```
h=plot(Lx,Ly,'-ob'); %plotagem em linha continua, marcador circular e cor azul
axis([-10 10 -10 10]) %limites dos eixos x e y
axis square  %transforma a area de plotagem em quadrado
grid on  %ativa as grades
I=18; %I=numero de iteracoes desejadas
for k=1:I  %variacoes
    T2=(k-1)*(2*pi/I)+T2_0;  %calculo do angulo theta2 da sequencia de frames
    Ax=R_AO*cos(T2);  %calculo da posicao atual do ponto Ax
    Ay=R_AO*sin(T2);  %calculo da posicao atual do ponto Ay
    Px=Ax+R_PA*cos(T2+Alpha); %calculo da posicao atual do ponto Px
    Py=Ay+R_PA*sin(T2+Alpha); %calculo da posicao atual do ponto Py
    Lx=[0, Ax, Px, 0]; %coordenadas em x atual da sequencia de pontos a plotar
    Ly=[0, Ay, Py, 0]; %coordenadas em y atual da sequencia de pontos a plotar
    set(h,'XData',Lx,'YData',Ly,'EraseMode','xor')  %plota os pontos O, A e P
    %calculados , e define EraseMode como xor para nao manter a figura no
    %proximo quadro. Para manter a figura, use EraseMode=normal ou none
    axis([-10 10 -10 10])  %limites dos eixos x e y
    axis square %transforma a area de plotagem em quadrado
    grid on  %ativa as grades
    t1=text(Ax+0.5,Ay+0.5,'A','Visible','on','EraseMode','xor'); %cria o texto
    %'A' na coordenada [Ax+0.5,Ay+0.5], com EraseMode=xor para nao manter
    %no proximo quadro. Tambem, habilita Visible=on.
    t2=text(Px+0.5,Py+0.5,'P','Visible','on','EraseMode','xor'); %cria o texto
    %'P' na coordenada [Px+0.5,Py+0.5], com EraseMode=xor para nao manter
    %no proximo quadro. Tambem, habilita Visible=on.
    xlabel('Eixo X');
    ylabel('Eixo Y');
    T2g=T2*180/pi; %converte o angulo T2 em graus
    T2gs=num2str(T2g); %converte o numero T2g em string
    t3=text(-3,-8,['Theta2= ',T2gs],'BackgroundColor','w','EdgeColor','b',...
        'FontSize',14);
    %cria texto com background branco, borda azul e tamanho de fonte 14.
    M(k) = getframe(gcf,[0 0 550 400]); %comando getframe para transformar a
```

```
    %figura atual num frame (quadro) para animacao posterior.
    set(t1,'Visible','off'); %o texto t2 deve ser apagado para o proximo frame
    set(t2,'Visible','off'); %o texto t2 deve ser apagado para o proximo frame
    set(t3,'Visible','off'); %o texto t3 deve variar de valor a cada iteracao
end
movie2avi(M,'rotacao1.avi','quality',95,'fps',2) %o comando movie2avi gera um video
%formato avi com os frames M gerados.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%rotp2.m
%roteiro para criar animacao de um corpo rigido em rotacao pura no plano XY
%ESTA ANIMACAO MANTEM OS QUADROS ANTERIORES
clear  %limpa da memoria todas as variaveis
close  %fecha todas as janelas graficas abertas
figure(3);  %abre a janela grafica numero 3
R_AO=5;  %distancia entre os pontos A e O do elo 2
R_PA=2;  %distancia entre os pontos P e A do elo 2
T2_0=pi/9; %angulo theta 2 inicial da linha entre os pontos O e A
Alpha=2*pi/3;  %angulo fixo entre as linhas PA e AO
Ax_0=R_AO*cos(T2_0);  %componente x do ponto A inicial
Ay_0=R_AO*sin(T2_0);  %componente y do ponto A inicial
Px_0=Ax_0+R_PA*cos(T2_0+Alpha); %componente x do ponto P inicial
Py_0=Ay_0+R_PA*sin(T2_0+Alpha); %componente y do ponto P inicial
Lx=[0, Ax_0, Px_0, 0]; %coordenadas em x da sequencia de pontos a plotar
Ly=[0, Ay_0, Py_0, 0]; %coordenadas em y da sequencia de pontos a plotar
h=plot(Lx,Ly,'-ob'); %plotagem em linha continua, marcador circular e cor azul
axis([-10 10 -10 10]) %limites dos eixos x e y
axis square  %transforma a area de plotagem em quadrado
grid on  %ativa as grades
I=11; %I=numero de iteracoes desejadas
```

```
for k=1:I  %variacoes
    T2=(k-1)*(2*pi/I)+T2_0;  %calculo do angulo theta2 da sequencia de frames
    Ax=R_AO*cos(T2);  %calculo da componente x atual do ponto A
    Ay=R_AO*sin(T2);  %calculo da componente y atual do ponto A
    Px=Ax+R_PA*cos(T2+Alpha); %calculo da componente x atual do ponto P
    Py=Ay+R_PA*sin(T2+Alpha); %calculo da componente y atual do ponto P
    Lx=[Lx,0, Ax, Px, 0]; %Lx anterior acrescentado das coordenadas em x atuais
    %da sequencia de pontos a plotar
    Ly=[Ly,0, Ay, Py, 0]; %Ly anterior acrescentado das coordenadas em y atuais
    %da sequencia de pontos a plotar
    set(h,'XData',Lx,'YData',Ly,'EraseMode','normal')  %plota os pontos O, A e P
    %calculados , e define EraseMode como none para manter a figura no
    %proximo quadro. Tambem poderia ser usado EraseMode=normal.
    axis([-10 10 -10 10])  %limites dos eixos x e y
    axis square %transforma a area de plotagem em quadrado
    grid on  %ativa as grades
    t1=text(Ax+0.5,Ay+0.5,'A'); %cria o texto'A' na coordenada [Ax+0.5,Ay+0.5]
    t2=text(Px+0.5,Py+0.5,'P'); %cria o texto'P' na coordenada [Px+0.5,Py+0.5]
    xlabel('Eixo X');
    ylabel('Eixo Y');
    T2g=T2*180/pi; %converte o angulo T2 em graus
    T2gs=num2str(T2g); %converte o numero T2g em string
    t3=text(-3,-8,['Theta2= ',T2gs],'BackgroundColor','w','EdgeColor','b',...
        'FontSize',14);
    %cria texto com background branco, borda azul e tamanho de fonte 14.
    M(k) = getframe(gcf,[0 0 550 400]); %comando getframe para transformar a
    %figura atual num frame (quadro) para animacao posterior.
    %set(t1,'Visible','off');
    %set(t2,'Visible','off');
    set(t3,'Visible','off'); %o texto t3 deve variar de valor a cada iteracao
end
movie2avi(M,'rotacao2.avi','quality',95,'fps',2) %o comando movie2avi gera um video
%formato avi com os frames M gerados.
```

Como exercício, faça as modificações necessárias para salvar esses dados gerados nos roteiros das animações em um arquivo para análise posterior.

16 Utilizando polinômios

Nesta seção veremos como trabalhar com funções polinomiais. A manipulação de polinômios é feita, basicamente, com seus coeficientes formando uma lista (vetor) horizontal. Por outro lado, as raízes de um polinômio formam uma lista vertical.

Vejamos um exemplo simples, o polinômio $x^2 - 5x + 6$, o qual tem como raízes os valores 2 e 3. Sua representação pode ser feita com a lista `p1`. Repare a ordem de disposição dos coeficientes:

```
>> p1=[1 -5 6]
p1 =
     1    -5     6
```

Suas raízes podem ser encontradas com a função `roots`:

```
>> r1=roots(p1)
r1 =
     3
     2
```

Descubra o que faz a função `poly`. Qual seria o resultado de `poly([2;3])`?

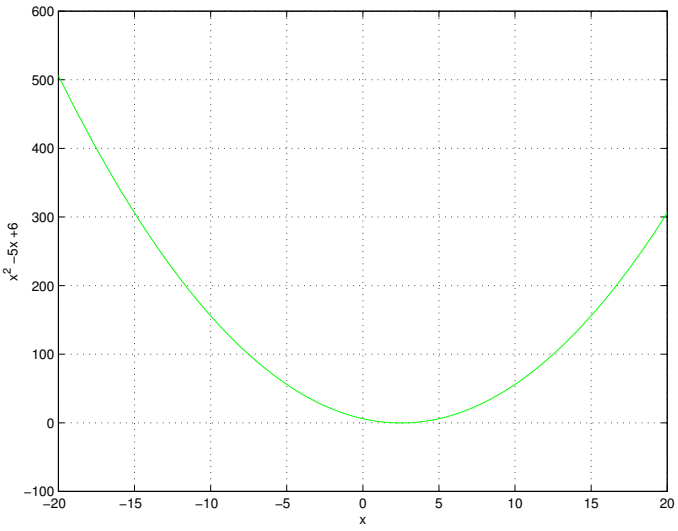
Em seguida, veremos como determinar o valor de uma função polinomial para um determinado valor da variável de entrada. Para o mesmo polinômio $p1$ definido acima, qual seria o resultado se $x = 59.2$? Basta usar a função `polyval`.

```
>> f1=polyval(p1,59.2)
f1 =
  3.2146e+003
```

Mais interessante do que isso, a variável x pode ser uma lista ou matriz. Veja como plotar a função $p1$ para valores entre -20 e 20:

```
>> x1=linspace(-20,20,300);
>> y1=polyval(p1,x1);
>> plot(x1,y1,'g'); grid on
>> xlabel('x')
>> ylabel('x^2 -5x +6')
```

Diversas operações podem ser efetuadas entre dois polinômios: soma, subtração, multiplicação, divisão. Para soma e subtração deve-se considerar as listas dos coeficientes dos polinômios e efetuar a operação



conforme visto na seção 7.1. Para multiplicação deve-se usar a função `conv`, enquanto que para divisão a função `deconv`.

Em todas as operações mencionadas acima, as listas devem ter número de elementos compatível com a ordem do polinômio. Deve-se completar com zeros até que essa condição seja satisfeita. Além disso, no caso de soma e subtração, as listas devem ser de mesma ordem.

Por exemplo, para efetuar a multiplicação do polinômio $p1 = x^2 - 5x + 6$ usado acima com o polinômio $p2 = x^5 + 7x^3 + 1$:

```
>> p2=[1 0 7 0 0 1]
p2 =
     1     0     7     0     0     1
>> p3=conv(p1,p2)
p3 =
     1    -5    13   -35    42     1    -5     6
```

Mas, para executar a soma dos dois polinômios devem ser utilizadas listas de mesma dimensão:

```
>> p1_1=[0 0 0 1 -5 6]   %lista p1 modificada
p1_1 =
     0     0     0     1    -5     6
>> p4=p1_1 + p2
p4 =
     1     0     7     1    -5     7
```

Por fim, veremos como determinar os resíduos, polos e termos diretos de uma fração de dois polinômios no método da expansão em frações parciais.

Dada uma fração de dois polinômios $B(s)$ e $A(s)$, podemos decompô-la numa soma de frações parciais,

$$\frac{B(s)}{A(s)} = k(s) + \frac{r(1)}{s - p(1)} + \frac{r(2)}{s - p(2)} + \ldots + \frac{r(n)}{s - p(n)}$$

Para isso, usa-se a função `residue`, que retorna como resposta três listas: resíduos, polos e termos livres. Por exemplo, para a fração de polinômios

$$\frac{B(s)}{A(s)} = \frac{s^3 + 5s^2 + 9s + 7}{(s+1)(s+2)}$$

teríamos a seguinte operação no MATLAB:

```
>> num=[1 5 9 7]
num =
     1     5     9     7
>> den=poly([-1;-2])
den =
     1     3     2
>> [r,p,k]=residue(num,den)
r =
    -1
     2
p =
    -2
    -1
k =
     1     2
```

Ou seja,

$$\frac{B(s)}{A(s)} = s + 2 + \frac{-1}{s+2} + \frac{2}{s+1}$$

Note que neste caso o numerador é um polinômio de ordem maior que o denominador, resultando, portanto, num termo livre k não nulo.

A função `residue` do MATLAB também serve para executar a operação inversa, ou seja, dadas as listas verticais dos resíduos e dos polos e a lista horizontal dos termos livres, ela determina as listas horizontais dos polinômios do numerador e do denominador:

```
>> [n,d]=residue(r,p,k)
n =
     1     5     9     7
d =
     1     3     2
```