UEL – UNIVERSIDADE ESTADUAL DE LONDRINA

OCTAVE - Uma Introdução

Primeiros contatos com o ambiente de programação numérica Octave

Prof. Sergio Roberto Teixeira
01/08/2010

Colaboradores:
Ulysses Sodre
Andrielber da Silva Oliveira
Sônia Ferreira Lopes Taffoli

Uma visão geral, desde os primeiros rudimentos, facilitando a introdução ao ambiente de cálculo e programação do pacote OCTAVE

AGRADECIMENTOS

A Deus em primeiro lugar por permitir a conclusão de mais essa etapa;

A minha esposa,
fiel companheira,
que mesmo na adversidade,
permanece firme confiando Nele;

Aos meus colegas indistintamente,
professores e funcionarios da UEL,
e em especial,
prof. Sodré com quem convivi lado a lado
a maior parte da vida profissional nesta instituição;

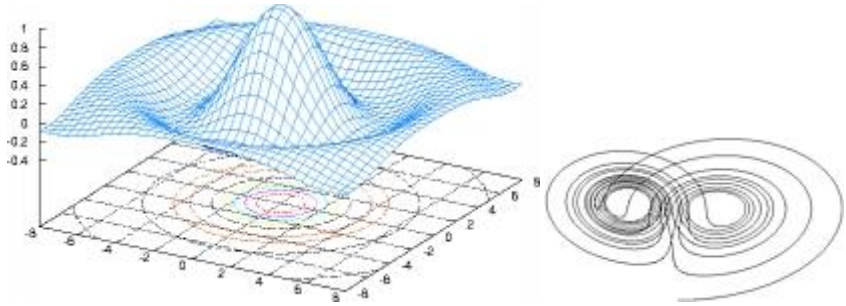
A cada aluno,
motivo maior, daquele que ensina;

"Nada há mais certo, justo e exato,
que passamos como a folha ao vento.
Construímos temporariamente, nesta vida passageira,
para a outra que perdura para a eternidade.
Nele – em Cristo Jesus."

Sumário

OCTAVE – UMA INTRODUÇÃO

1 O PACOTE OCTAVE



O programa GNU Octave é uma linguagem de alto nível, direcionada para cálculo numérico. Ele fornece uma interface de linha de comando conveniente para resolver problemas numericos lineares e não lineares, e para realizar outros experimentos numéricos usando uma linguagem que é bastante compatível com o Matlab. Também pode ser utilizada como linguagem de programação. O que se segue pode ser encontrado em
http://www.gnu.org/software/octave/

1.1 Novas versões:

- **01 de agosto de 2010 -** Uma nova atualização da versão atual em desenvolvimento do Octave está disponível para ftp (vr 3.3.52). Você só deve utilizar esta versão se estiver interessado em testar as novas funcionalidades e não se importar com bugs ocasionais. Até então não disponível para Windows.

- **24 de março, 2010 -** Uma nova atualização da versão atual em desenvolvimento do Octave está disponível para ftp (3.3.51). Você só deve utilizar esta versão se estiver interessado em testar as novas funcionalidades e não se importar com bugs ocasionais.

- **28 de janeiro de 2010 -** Versão 3.2.4 foi liberada e está agora disponível para ftp. Octave 3.2.4 é uma versão amadurecida, com uma lista de alterações expressivas em relação à versão anterior série 3.2.x.

1.2 Sobre o Octave

O programa Octave possui uma grande quantidade de ferramentas para a resolução de problemas de álgebra linear, encontra as raízes de equações não lineares, integra funções ordinárias, manipula polinômios, integra equações diferenciais ordinárias e equações diferenciais algébricas.

Pode ser extendido facilmente como personalizado, através das funções definidas pelo usuário escritas na própria linguagem do Octave, ou usando módulos lincados, escritos em C + +, C, Fortran ou outras linguagens.

GNU Octave é um software livre. Você pode redistribuí-lo e / ou modificá-lo sob os termos da GNU General Public License (GPL) conforme licença de publicação da Free Software Foundation.

O Octave foi escrito por John W. Eaton e muitos outros. Pelo fato do Octave ser um software livre, todos são convidados a torná-lo mais útil, escrevendo funções adicionais que contribuam para isso, e relatando quaisquer problemas que possam encontrar.

1.3 História

O Octave originalmente foi criado por volta de 1988, com o propósito de ser um software de apoio a um livro de graduação em projetos de reator químico, escrito por James B. Rawlings da Universidade de Wisconsin-Madison e John G. Ekerdt da Universidade do Texas. Originalmente foi idealizado como ferramenta muito especializada relacionado à criação de reatores químicos. Posteriormente, após constatar as limitações dessa abordagem, optou-se pela construção de uma ferramenta mais flexível.

Nessa época, muitas pessoas eram de opinião que se deveria usar apenas a linguagem Fortran, porque era a linguagem de computação da engenharia. Mas cada vez que os alunos tinham problema com o Fortran, gastavam muito tempo tentando descobrir o que deu errado no seu código Fortran, e não tinham tempo suficiente para aprender sobre a engenharia química. O ambiente interativo do Octave possibilita que o aluno domine rapidamente seus conceitos básicos e, começe a usá-lo com segurança em pouco tempo.

O desenvolvimento efetivo do Octave começou em 1992. A primeira versão alpha é de 04 de janeiro de 1993, e a versão 1.0 foi lançada em 17 de fevereiro, 1994. Desde então, o Octave tem passado por várias revisões importantes.

Atualmente é claro que o Octave é muito mais do que apenas um pacote para um curso com utilidades que vão além da sala de aula. Embora o objetivo inicial fosse um tanto vago, sabiam que queríam criar algo que permitisse aos alunos resolver problemas reais, e pudessem usá-lo para muitos outro propósitos além de projetos relacionados a reator químico. Hoje, milhares de pessoas no mundo usam Octave no ensino, na pesquisa e em aplicações comerciais.

O nome Octave pode dar a entender que tem algo haver com música, mas na verdade não. Está relacionado ao nome de um dos ex-professores, autor de um livro famoso sobre engenharia química.

Os autores tiveram como propósito neste software, possibilitar que muitas pessoas, realizem cálculos ambiciosos com mais facilidade usando um programa free.

1.4 Documentação

O Octave é amplamente documentado por um manual encontrado no seu site. A versão disponível está em HTML, juntamente com a cópia do código-fonte. É possível usar esses arquivos para criar uma cópia impressa do manual.

A versão para impressão do manual do Octave está disponível em

http://www.network-theory.co.uk/octave/manual/.

Toda a arrecadação com a venda deste livro apoia o desenvolvimento do Software Livre. Para cada exemplar vendido um dólar será doado para o Fundo de Desenvolvimento GNU Octave.

Se suas perguntas não forem respondidas pelo manual, ou pelo site, há também o FAQ, uma lista de questões freqüentes, com respostas. A cópia da FAQ disponíveis aqui, geralmente é mais atual do que a versão distribuída no pacote Octave.

Finalmente, se mesmo o FAQ não der uma resposta para o que você está procurando, existe uma lista help Octava para discussões de questões relacionadas à utilização e instalação do Octave.

1.5 Versão atual

Existem várias versões diferentes do Octave disponíveis para download:

- **Estável -** Esta versão é bem testada e é a versão recomendada para a maioria dos usuários do Octave.
- **Desenvolvimento -** A mais recente versão em desenvolvimento com a maioria das fontes em desenvolvimento (disponível). Use esta opção se quiser conhecer as últimas características do programa, não levando em conta, que algumas coisas podem apresentar problemas.

Octave versão:	**Versão:**	**Data de Lançamento:**
Estável	3.2.4 (. tar.gz) (. tar.bz2)	28 de janeiro de 2010
Ensaio	3.3.52 (. tar.gz) (. tar.bz2)	1 de agosto de 2010

1.6 Sistemas Operacionais disponíveis: Instalação

As distribuições binárias do Octave são feitas por voluntários. Não há como fazer promessas sobre estes pacotes.

- Linux: A versão Linux geralmente está disponível com as distribuições (Debian, SuSE, RedHat, e outros pacotes). É a versão própria do pacote.

- Mac OX X, Sun Solaris, OS/2 ou eComStation.

- Windows: O projeto Octave Forge tem uma distribuição binária do Octave construído com o compilador MinGW. O link **Windows installer (Octave and Octave-Forge)** vai fazer download do programa Octave-3.2.4_i686-pc-mingw32_gcc-4.4.0_setup.exe.
  Este programa é um instalador, basta executa-lo no ambiente Windows e você terá o Octave funcionando em seu computador.

2 INTRODUÇÃO:

2.1 PRIMEIROS CONTATOS

a) Variáveis de controle

Octave foi escrito em linguagem C e por isso, absorve o estilo do próprio C. Alguns comandos lembram o próprio C com, por exemplo, dir. O Octave dispõe de comando de gerenciamento tanto para gerenciamento de diretórios como, para gerenciamento do próprio ambiente Octave. Podemos, por exemplo, listar as variáveis por nome, tamanho, número de bits e classe. Podemos deletar uma variável ou todas elas. Tudo que segue será feio na linha de comando, seguido do comando <enter>.

Exemplo: Variáveis de Controle

```
#a) Deletando as variáveis de processo:
> clear all
>a=1
a =  1

 >b=2
b =  2

 >a+b
ans =  3

#b) Listando as variáveis de processo:
 >who
Variables in the current scope:
a    ans  b

#c) Deletando parcialmente as variáveis de processo:
>clear a, who
Variables in the current scope:

ans  b

 >clear b, who
Variables in the current scope:
ans


#d)  listando os arquivos de um diretório:
# ls, dir – listam diretorio
>  ls
>  dir

#e) Mostra o diretório em uso:
```

```
>pwd

#f) Seta o diretório de usuário do Windows:
>cd ~

 >ls
 Volume in drive C has no label.
 Volume Serial Number is 65FB-F8F8

 Directory of C:\Users\Pr. Sergio

[.]              .recently-used.xbel   [Music]
[..]             [.thumbnails]        Pictures.lnk
[.assistant]        .xmaximarc           [Searches]
.fonts.cache-1       [Contacts]           [Temp]

...
          13 File(s)    820,854,197 bytes
          22 Dir(s)  46,610,014,208 bytes free


 #g) Limpa tela:
 # clc, home, ctrl l -  limpa a tela (clear screan)
 # Somente a tela de saída, não deleta nada!
>clc;

{h) Nova linha de edição:
 Quando uma variável for muito longa,
 podemos quebrar a linha com os comandos "..." ou "\",
 e continuar na linha seguinte:
 }
> x=[1, 2, 3, 4, 5, ...
       6, 7, 8, 9, 10]
x =
   1   2   3   4   5   6   7   8   9  10


# i) prompt:
# PS1 - Variável que controla o prompt do Octave.

# Valor defaut de PS1:
>PS1

#O que usamos normalmente:
>PS1(" >")
```

b) Matrizes

As matrizes e suas operações de soma, subtração, multiplicação por escalar, multiplicação de matrizes e potenciação são definidas de forma natural.

Exemplo:

```
#a) Definindo as matrizes a,b:
> a=[1 2;3 4], b=[4 3;2 1]

 a =
   1   2
   3   4

b =
   4   3
   2   1

#b) Produto por escalar:
> 2*a
ans =
   2   4
   6   8

#c) Adição e subtração de matrizes:
> a+b
ans =
   5   5
   5   5

> a-b
ans =
  -3  -1
   1   3

#d) Produto de matrizes:
> a*b
ans =
    8    5
   20 13

#e) Criando uma matriz randomica 3x3:
#Cada vez tem uma resposta diferente
> rand(3)
ans =
   0.0044995   0.9793440   0.6066929
   0.3587679   0.6373938   0.5112695
   0.1748393   0.8444124   0.8188124
```

```
#f) Potenciação de matrizes:
>  a=[1 2;3 4], a*a, a^2, a*a-a^2

a =
  1   2
  3   4

ans =
   7   10
  15   22

ans =
   7   10
  15   22

ans =
  0   0
  0   0

#g) Definindo um vetor, somando seus elementos:
# sum – função interna
>v=[1 2 3 4], s=sum(v),p=prod(v)
v =
  1   2   3   4

s =  10
p =  24

#h) Encontrando o comprimento (módulo) do vetor:
> v=[1 2 3 4], length(v)
v =
  1   2   3   4
ans =4

#i) Encontrando o produto interno de x,y:
> x=[1 2 3 4], y=[2 4 6 8], dot(x,y)
x =
  1   2   3   4

y =
  2   4   6   8
ans =  60
```

```
#j)Determinante da matriz a:
> a=[1 2; 3 4], d=det(a)
a =
   1   2
   3   4
d = -2

#k) Inversa de a:
>a=[1 2;3 4];  b=inv(a)
b =
  -2.00000   1.00000
   1.50000  -0.50000

 >a*b
ans =
   1   0
   0   1

 >b*a
ans =
   1   0
   0   1

#l) Solução do sistema Ax=b
> A=[1 2;3 4];
A =
   1   2
   3   4

> b=[3;7]
b =
   3
   7

> x=A\b
x =
   1
   1
```

c) Gráficos:

Retorna um vetor partição de n elementos, com n-1 intervalos, uniformemente espaçados no intervalo [a,b].

Exemplo:

```
#a) Definindo uma partição em [0,10]
 #com: 5 intervalos, 6 pontos:
>linspace(0,10,6)
ans =
   0   2   4   6   8  10

##============================
## b)     2D – Gráficos no Plano
##============================
x=linspace(0,10,100);
#Plotando o gráfico do sin:
plot(x,sin(x))

title ("Funções trigonométricas:");
 xlabel ("x");
 ylabel ("y");

 ##text (x0,y0, "comentario no grafico");
 text (3,0.7, "Grafico do seno:");
 legend ("sin (x)");
 grid on;


##============================
## c)    3D – Gráficos no Espaço
##============================
## meshgrid, mesh:
 clc,clf;
 printf("Plot3d: meshgrid, mesh \n\n");
 #x = -2:0.1:2;
 x = y=-2:0.1:2;
 [u,v] = meshgrid(x,y);
 z = sin(u.^2-v.^2);

 mesh(x,y,z); #figura

title("mesh - superficie")
 text (1,2,3, "z = sin(x^2-y^2)");
 xlabel ("eixo - x");
 ylabel ("eixo - y");
zlabel ("eixo - z");
```

d) Números Complexos

Uma constante complexa pode ser um escalar, um vetor, uma matriz que podem conter valores complexos, sendo que i, é a unidade imaginário, i^2 = 1 ou i=sqrt(-1). Um número complexo tem a forma a+bi, onde a,b são constantes reais normalmente ponto-flutuante de precisão dupla. O Octave reorganiza os complexos na forma a+bi onde, *não pode ter espaço entre o b e o i, isto causará erro*. Podemos usar livremente a variável i, como outra qualquer, desde que não esteja sendo usada como complexa.

Exemplo: Números Complexos

```
#a) Unidade imaginária i usada como variável:
> i
ans =  0 + 1i

> i=2
i =  2

#a1) usada como variável numérica
> i^2
ans =  4

#a2) usada como vetor:
>i=1:10
i =
   1   2   3   4   5   6   7   8   9  10

#a3) usada como índice de contador:
 >for i=1:10 i^2, endfor
ans =  1
ans =  4
ans =  9
ans =  16
ans =  25
ans =  36
ans =  49
ans =  64
ans =  81
ans =  100

#a4) Deletando i, i volta para ser a unidade imaginária:
> clear i     %limpa a setagem
> i^2         % retorna o valor interno.
ans = -1
```

```
#b) Retorna um número complexo:
> complex(2)
ans =  2 + 0i

> complex(2,3)
ans =  2 + 3i

#c) Retorna verdadeiro (valor=1) se for complexo:
> iscomplex(ans)  % testa se a resposta é complexo
ans =  1

#d) As operações aritméticas com complexos:
z=1+i
w=1-i

2*z

z,w,z+w
z,w,z-w
z,w,z*w

#e) Parte real e imaginária de z, isto é, se z=a+bi, real(z)=a, imag(z)=b:
> z=3+4i
z =  3 + 4i

> imag(z),real(z)
ans =  4
ans =  3

#f)  Encontra o conjugado de z, isto é:
# se z=a+bi, conj(z)=a-bi:
> z=3+4i
z =  3 + 4i

 > conj(z)
ans =  3 - 4i

 > z*conj(z)
ans =  25

 > r=sqrt(z*conj(z))
r  =  5

#g) norma de z, isto é:
# se z=a+bi, abs(z)=sqrt(a^2+b^2)= sqrt(z*conj(z))
>z =  3 + 4i;  r=abs(z)
r  =  5
```

```
#h) argumento de z , em coordenadas polares,  isto é:
# se z=a+bi, arg(z)=atan2(b,a)
> z =  1 + 1i
z =  1 + 1i

>t= arg(z)
t   =  0.78540

> atan(1)
ans =  0.78540

> pi/4
ans =  0.78540
```

e) Operadores de identificação:

Usamos os operadores de idntificação para introduzir alguns objetos básicos do Octve como número, caracter, complexo, vetor e matriz.

isnumeric (x)	retorna não nulo se x é um objeto numérico
ischar(s)	retorna 1 se s é uma string
isreal (x)	retorna não nulo se x é um objeto numérico real
isfloat (x)	retorna não nulo se x é um objeto numérico pto. flutuante
iscomplex (x)	retorna não nulo se x é um objeto numérico de valor complexo
ismatrix (a)	retorna 1 se a é matriz, c/c zero
isvector (a)	retorna 1 se a é vetor, c/c zero

Exemplo: is...

```
#==================
# a) Numérico e string:
#==================
>isnumeric("abs")
ans = 0
>ischar("123")
ans =  1
 >isnumeric(123)
ans =  1

#================
#b) ponto flutuante:
#================
 >isfloat (123)
ans =  1
 >isfloat (123.45)
```

```
ans =  1

>x=12345, isinteger(x),isreal(x), isfloat(x),y=int32(x), isinteger(y)
x =  12345
ans = 0
ans =  1
ans =  1
y = 12345
ans =  1

 >whos
Variables in the current scope:
  Attr Name       Size                Bytes  Class
  ==== ====       ====                =====  =====
      ans        1x1                     1  logical
       x         1x1                     8  double
       y         1x1                     4  int32

Total is 3 elements using 13 bytes

#=============
#c) Complexos:
#=============
 >iscomplex (123)
ans = 0
 >isreal (123)
ans =  1
  >iscomplex (1+2i)
ans =  1

#===============
#d) matriz e vetor:
#===============
 >a=[1 2;3 4],ismatrix (a)
a =
   1   2
   3   4
ans =  1

 >v=[1 2  3 4],ismatrix (v)
v =
   1   2   3   4
ans =  1

 >v=[1 2  3 4],isvector (v)
v =
   1   2   3   4
```

```
ans =  1

 >v=1:4,isvector (v)
v =

  1   2   3   4
ans =  1

>a, isvector (a), ismatrix(a)
a =
  1   2
  3   4

ans = 0
ans =  1
```

2.2 FUNÇÕES:

a) Funções Trigonométricas

O Octave disponibiliza funções trigonométricas em radianos e graus e suas inversas. Também encontramos funções hiperbólicas e suas inversas. Segue alguns exemplos:

Exemplo: Trigonométricas

```
#a) sin(x) e asin(x) em radianos:
>x=pi/4
x =  0.78540
 >y=sin(x)
y =  0.70711
 >asin(y)
ans =  0.78540
 >ans*4
ans =  3.1416

#b) sin(x) e asin(x) em graus:
> x=45; y1=sind(x), x2=pi/4;y=sin(x)
y1 =  0.70711
y2 =  0.70711

> x=45; y1=cosd(x), x=pi/4;y2=cos(x)
y1 =  0.70711
y2 =  0.70711

#c) Hiperbólicas:
> x=1,   y=(exp(x)-exp(-x))/2,
```

```
y =  1.1752

> x =  1, z= sinh(x)
z =  1.1752

#d) Trigonometria em graus:
> x=45; y=sind(x)
y =  0.70711

> z=asind(y)
z =  45.000
```

b) Funções Exponenciais e Logarítmicas:

Algumas das funções definidas são;

- exp (x) – calcula a exponencial de x para um x real.
- log (x) – calcula o logaritmo natural `ln (x)', para um valor x.
- log10 (x) – calcula o logaritmo na base 10 de um valor x.
- log2 (x) - calcula o logaritmo na base 2 de um valor x.
- sqrt (x) – calcula a raiz quadrada de um valor x.

Exemplo: Funções logarítmicas e exponenciais

```
> exp(1)
ans =  2.7183

> x=exp(1),log(x)
x =  2.7183
ans =  1

> log10(1000)
ans =  3

> x=log2(16), sqrt(x)
x =  4
ans =  2
```

c) Função Fatoral e Gamma:

Exemplo: Algumas funções encontradas no Octave

```
#a) Fatorial:
> x=6; factorial(x)
 ans =  720
```

```
#b) Função gamma (x+1):
> gamma(x+1)
ans =  720

>#c)  Só pode ser calculado com a função gamma:
> x=6.001; gamma(x+1)
ans =  721.35
```

d) Editando uma função:

Podemos editar uma função basicamente de três maneiras:

Ex.01: Usando o comando inline()

```
# nome_func=inline("corpo_da_função")
#                    corpo_da_função deve ser uma variável string
# Apropriado para uso imediato de funções de edição mais simples.

#Exemplo do uso do comando inline para definir f=x^2+1:
>f = inline("x^2+1")
f =
f(x) = x^2+1

 >f(2)
ans =  5

 >f
f =
f(x) = x^2+1

#Para o gráfico de f, devemos vetorizar  f:
#Trocar:  "*", "/", " ^"  por ".*",  " ./",  ".^"
#          "+","- " permanecem sem alteração.
>x=linspace(-1,1,50);

>plot(x,f(x))
error: for A^b, A must be square
error: called from:
error:    at line -1, column -1
error: evaluating argument list element number 2
error: evaluating argument list element number 1

>f = inline("x.^2+1")
>plot(x,f(x));   ##ok...
```

Ex.02: Usando o comando function ... endfunction

```
# function  y=f(x)  comandos_da_função;  endfunction:
#              f – nome da função
#              y - retorno
# Apropriado para programação de funções mais complexas,
# e rotinas de várias variáveis como, por exemplo a integral de Simpson.

#Exemplo do uso para definir f=x^2+1:
> function y = f (x)
   y = x.^2+1;
  endfunction;

>f(1)
ans =  2

>x=linspace(-1,1,50);
>plot(x,f(x))
```

Ex.03: Usando o comando @()

```
#  nome_func=@(var) corpo_da_função;
#                        corpo_da_função - uma variável normal (não string).
# Semelhante ao primeiro caso,
# Apropriado para passagem de parâmetros que sejam funções.
# Isto é, quando uma função tem como parâmetro outra função.

#Exemplo do uso para definir f=x^2+1:
>f = @(x)x.^2+1;
f =
@(x) x .^ 2 + 1

>f(1)
ans =  2

 >f
f =
@(x) x. ^ 2 + 1

>x=linspace(-1,1,50);
>plot(x,f(x))
```

Exemplo: função de mais que uma variável

```
>function [soma, produto] = sp(x,y)
 soma=x+y;
 produto=x*y;
endfunction

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

 >sp(2,3)
ans =  5

 >[s,p]=sp(2,3)
s =  5
p =  6
```

e) Recursividade

Uma função é recursiva quando chama a si própria direta ou indiretamente. Um exemplo clássico é a função fatorial para um inteiro. Neste caso, para valores maiores, retorna ponto flutuante.

Exemplo:

```
>function retval = fatorial (n)
if (n > 0)
    retval = n * fatorial (n-1);
else
    retval = 1;
endif
endfunction

> fatorial(5)
ans =  120

> fatorial(170)
ans =  7.2574e+306        # aproximação em ponto flutuante

> fatorial(171)
ans = Inf                 # limite de  Overflow
```

2.3 OPERADORES

a) Operadores lógicos:

Operadores de comparação para determinar, por exemplo, se são iguais. Todos os operadores de comparação do Octave retornam 1 como verdade e 0 como falso.

Resumo:

- x < y, verdade se x é menor que y.
- x <= y, verdade se x é menor ou igual a y.
- x >= y, verdade se x é maior ou igual a y.
- x > y, verdade se x é maior que y.
-
- x == y, verdade se x é igual a y.
- x != y,
- x ~= y, verdade se x é diferente de y.

Exemplo:

```
#a) Testa igualdade termo a termo:
>a= [1, 2; 3, 4],  b= [1, 3; 2, 4],  a==b
a =
  1  2
  3  4

b =
  1  3
  2  4

ans =
  1  0
  0  1

%Observe a diverença da atribuição:
>a=b
a =
  1  3
  2  4

#b) Testa quem é igual a 1:
> 1 == [1, 3; 1, 4]
ans =
  1  0
  1  0

#c) Testa quem é maior que quem termo a termo:
> [1, 2; 3, 4]  >  [1, 3; 2, 4]
ans =
```

```
   0   0
   1   0

> [1, 2; 3, 4] !=  [1, 3; 2, 4]
ans =
   0   1
   1   0
```

b) *Operadores booleanos*

Uma expressão booleana elemento por elemento, é uma combinação de comparação de expressões usando os operadores:

- |, operador "ou"
- &, operador "e"
- ~ ou !, operador "não"

Uma expressão booleana é verdadeira se o resultado do cálculo das combinações lógicas das expressões booleanas for verdadeira. O valor será considerado falso se for zero, e verdadeiro de outra forma.

Exemplo:

```
 >  [1, 0; 0, 1] & [1, 0; 2, 3]
ans =
   1   0
   0   1

>~[1 0 3 4]  % 0 – falso, não 0 - verdadeiro
ans =
   0   1   0   0
```

c) *Operadores de incremento*

Operadores de incremento aumentam ou diminuem 1 numa variável, escrito com o símbolo de "++" ou "--". Temos duas variantes do operador o pré-incremento e o pós-incremento:

- x++, retorna o valor de x, depois incrementa.
- ++x, incrementa, depois retorna o valor de x.
- x--, retorna o valor de x, depois decrementa.
- --x, decrementa, depois retorna o valor de x.

Exemplo:

```
%a)  retorna o valor de x, depois incrementa.
> x=1, x++, x
x =  1
ans =  1
x =  2
```

```
%b)  incrementa, depois retorna o valor de x.
> x=1, ++x, x
x =  1
ans =  2
x =  2
```

2.4 LAÇOS

Laços ocupam um lugar relevante no controle e núcleos de repetição. Os comandos de controle de laço são: while, for, do, if, switch

Exemplo: Comandos de controle:

```
#==================================
# while CONDIÇÃO  COMANDOS;  endwhile:
#==================================

> function f (n)
     while (n-- > 0)          % operador n++, n—da linguagem C
         printf (“ n =  %d \n”,n);
     endwhile
  endfunction

> f(5)
 n =  4
 n =  3
 n =  2
 n =  1
 n =  0

#==================================
# for var = expressão  COMANDOS;  endfor
#==================================

> function S=f (n)
 #S=f(n): Soma de 1:n
S=0;
for i=1:n
     S=S+i;
endfor;
endfunction;

> f(5)
 ans = 15
```

```
#==================================
#    do COMANDOS; until (CONDIÇÃO)
#==================================
> function y=fib(n)
# y=fib(n): Calcula a seqüência de Fibonatti, de ordem n.
y=ones (1, n);
i = 2;
do
      i++;
     y (i) = y (i-1) + y (i-2);
 until (i == n)
endfunction


> fib(3)
ans =
  1   1   2

> fib(5)
ans =
  1   1   2   3   5

> fib(8)
ans =
   1   1   2   3   5   8   13   21
```

2.5 ENTRADAS E SAÍDAS

a) Entrada e saída numérica:

Controla a saída e o formato de saída das variáveis numéricas.

Exemplo: Controlando a saída de variáveis numéricas.

```
#a) Variável de resposta default:
> pi
ans = 3.1416

#a1) Display:
> disp("Valor de pi: "), disp(pi)
Valor de pi:
 3.1416

#b)  format(op) -  op: short – saída curta, long – saída longa.
#Se op é omitido, retorna ao formato default.
> format short, pi
ans =  3.1416
```

```
> format long, pi
ans =  3.14159265358979
> pi^64
ans = 6.57040064457169e+031  % formato exponencial, base 10
```

b) save/load

Salva / Carrega as variáveis em um arquivo nomeado.

Exemplo: Mostra o uso dos comandos save/load. Todas as linhas abaixo são de comandos, foram omitidas as saídas para maior clareza do texto.

```
who                #mostra as variáveis definidas
clear all   #deleta estas variaveis

#a) define duas matrizes a, b:
a=[1 2;3 4]
b=[4 3;2 1]

#b) salva a,b no arquivo arq_01.m
save arq_01.m a b
clear all        #deleta as variáveis a, b.
who        #lista as variáveis definidas...

#c) carrega arquivo arq_01.m
load arq_01.m
who         #a,b agora estão disponiveis
a,b
clear all     #limpa tudo...

#d) carrega a variável a, do arquivo arq_01.m
load arq_01.m  a      #carrega no arq_01.m, a variável a
who         #a-disponíveis, b-indisponível
a^2
```

c) Saída convencional: printf

- printf (TEMPLATE, ...) Imprime os argumentos sob o controle as string template, na saída `stdout', isto é, saída default, normalmente o vídeo.

Conversões de saída:

- '%d', '%i' - Imprime um inteiro com um simples número decimal.

- '%f' - Imprime um ponto-flutuante em ponto-flutuante.
- '%e' - Imprime um ponto-flutuante em notação exponencial.
- '%g' - Imprime um ponto-flutuante em ponto-flutuante ou exponencial, o que tiver tamanho mais conveniente.

- '%c' – Imprime um caráter.
- '%s' – Imprime uma string,
- '%%' – Imprime o caractere literal '%'.

O uso de conversões inválidas pode causar erros não perceptíveis (que pode facilmente causar engano!), o que deve ser evitado ao máximo.

Exemplo:

```
#a) Várias saídas numérico:
>  printf ("%4d, %4.2f %8.4e %8.4g\n", 1234, 34.123456, e^10, e^10);
1234, 34.12, 2.2026e+004, 2.203e+004

#b) Várias saídas de um valor numérico:
>  e^10, printf (" a=%5i,\n b=%4.2f,\n c=%8.4e,\n d=%8.4g\n", e^10, e^10,e^10, e^10)
ans =  22026.4657948067
 a=22026,
 b=22026.47,
 c=2.2026e+004,
 d=2.203e+004

#c) Saída string:
> printf ("Podemos escrever strings: %s \n",  "Octave");
Podemos escrever strings: Octave

#d) Saída de caracteres:
> printf ("Caracteres acentuados e especiais: %c, %c \n", 128, 123)
Caracteres acentuados e especiais: Ç, {
```

```
#e) Saída matricial:
> h=hilb(3), printf ("%4.2f %10.2e %8.4g\n", h);
h =
  1.00000   0.50000   0.33333
  0.50000   0.33333   0.25000
  0.33333   0.25000   0.20000

1.00  5.00e-001   0.3333
0.50  3.33e-001     0.25
0.33  2.50e-001      0.2

>  h=hilb(4), printf ("   %.5f   %.5f   %.5f   %.5f\n", h);
h =

  1.00000   0.50000   0.33333   0.25000
  0.50000   0.33333   0.25000   0.20000
  0.33333   0.25000   0.20000   0.16667
  0.25000   0.20000   0.16667   0.14286

  1.00000   0.50000   0.33333   0.25000
  0.50000   0.33333   0.25000   0.20000
  0.33333   0.25000   0.20000   0.16667
  0.25000   0.20000   0.16667   0.14286

f) Saída de caracteres especiais:
>printf("\n Isto %c um sufoco, mas %c poss%cvel: \n\n",130,130,161)

Isto é um sufoco, mas é possível:

> printf("%c\n%c f(x) dx = 1.2  \n",244,245)
⌠
⌡ f(x) dx = 1.2
```

3 MATRIZES E ÁLGEBRA LINEAR:

3.1 ELEMENTOS DE UMA MATRIZES – MATRIZES PRÉ DEFINIDAS

a) linspace (a,b, n)

Retorna um vetor partição de n elementos, com n-1 intervalos, uniformemente espaçados no intervalo [a,b].

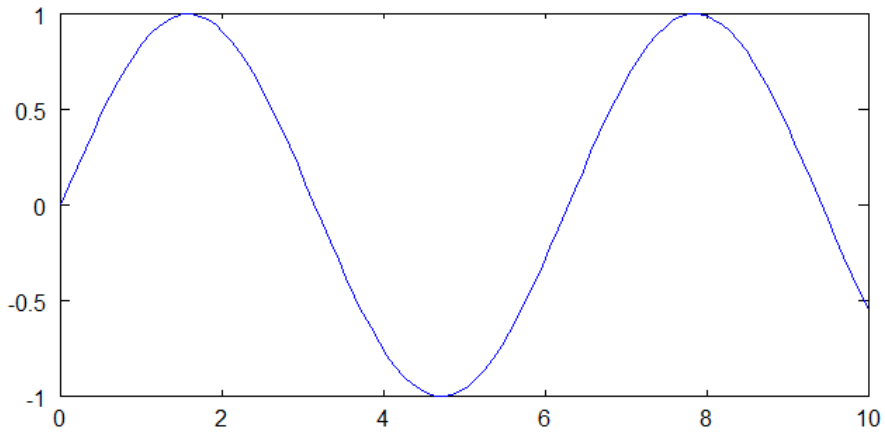
Exemplo:

```
> linspace(0,10,11)

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

Usado para definir uma partição de domínio de uma função. Como o ambiente do Octave é numérico, tudo que fazermos com funções, primeiramente precisamos definir o domínio.

Exemplo:

```
> x=linspace(0,10,100); plot(x,sin(x))
```



b) eye (n), eye (m,n)

Retorna a matriz diagonal generalizada d (i,i)=1, nxn ou mxn.

Exemplo:

```
> eye(3)

ans =
Diagonal Matrix
  1  0  0
  0  1  0
  0  0  1

> eye(3,4)
```

```
ans =
Diagonal Matrix
  1  0  0  0
  0  1  0  0
  0  0  1  0
```

c) zeros (n), zeros (m,n)

Retorna a matriz nula nxn ou mxn.

Exemplo:

```
> zeros(3), zeros(2,3)

ans =
  0  0  0
  0  0  0
  0  0  0

ans =
  0  0  0
  0  0  0
```

d) ones (n, m)

Retorna a matriz mxn com todos os elementos iguais a 1.

Exemplo:

```
> ones(3)
ans =
  1  1  1
  1  1  1
  1  1  1
```

e) repmat (A, m,n)

Faz cópia de A, numa matriz de bloco, tantos blocos quanto mxn.

Exemplo:

```
> a=[1 2 ;3 4]
a =
  1  2
  3  4
```

```
> repmat(a,2,2)
ans =
  1  2  1  2
  3  4  3  4
  1  2  1  2
  3  4  3  4
```

f) rand, randn,

rand - retorna um número randômico entre 0 e 1.
rand(n), rand(m,n) - retorna uma matriz mxn, com elementos randômicos entre 0 e 1.

randn(m,n) - retorna uma matriz mxn, com elementos randômicos com tendência de média = 0 e variância = 1, a medida que m,n crescem.

Exemplo: rand()

```
> rand
ans =  0.28384

> rand(2)
ans =
  0.333241   0.094428
  0.368583   0.950103

> rand(2,3)
ans =
  0.341284   0.313604   0.768712
  0.053145   0.846739   0.818190
```

Exemplo: randn()

```
> a=-randn(1,10)
a =
 Columns 1 through 6:
  -0.5940489   0.9916678   1.4583828   1.1016080  -0.0073346  -1.3948840

 Columns 7 through 10:
   1.7967888   0.5589617  -1.0512288  -0.4912209
```

g) Testando se um elemento é nulo:

- any(v) - retorna 1 o vetor v contem algum elemento não nulo.

Exemplo:

```
> v=[0 0 0 0 1], w=[0 0 0 0 ], any(v), any(w)
v =
     0   0   0   0   1

w =
     0   0   0   0

ans =  1
ans = 0
```

- any(a) - verifica se cada coluna da matriz a é ou não nula.

Exemplo:

```
> a=[1 2 3 0; 5 6 7 0; 0 1 2 0]
a =
  1   2   3   0
  5   6   7   0
  0   1   2   0

> any(a)
ans =
  1   1   1   0
```

h) find (a)

Quando encontra um elemento não nulo na matriz a, retorna sua posição na matriz a, contada como um vetor, de cima para baixo, da esquerda para a direita.

Exemplo:

```
> a=eye(3), find(a)
a =
Diagonal Matrix
  1   0   0
  0   1   0
  0   0   1

ans =
  1
  5
  9
```

i) sort(a)

Ordena as colunas da matriz a, em ordem crescente de cima para baixo.

Exemplo:

```
> a=[5 3 2;2 2 6;7 5 3], sort(a)
a =
  5  3  2
  2  2  6
  7  5  3

ans =
  2  2  2
  5  3  3
  7  5  6
```

j) vec(a)

Retorna um vetor coluna, com os elementos da matriz a, coluna a coluna.

Exemplo:

```
> a=[1 2;3 4], vec(a)
a =
  1  2
  3  4

ans =
  1
  3
  2
  4
```

k) sum(v), sum(a)

Se v, um vetor, retorna a soma dos elementos de v.
Se a, uma matriz, retorna um vetor cujos elementos são a soma das colunas da matriz a.

Exemplo 01:

```
> v=[1 2 3 4], sum(v)
v =
  1  2  3  4

ans =10
```

Exemplo 02:

```
> a=[1 2; 3 4], sum(a), sum(ans)
a =
  1  2
  3  4

ans =
  4  6

ans = 10
```

l) length(v)

Retorna o comprimento do vetor v.

Exemplo:

```
> v=[1 2 3 4], length(v)
v =
  1  2  3  4

ans =4
```

m) size(a), [m,n]=size(a)

Retorna a dimensão mxn da matriz a.

Exemplo:

```
> a=[1 2 3; 4 5 6], [m,n]=size(a)
a =
  1  2  3
  4  5  6

m =  2
n  =  3
```

Exemplo: size(v) - a resposta é dada na forma de matriz.

```
> v=[1 2 3 4], size(v)
v =
  1  2  3  4

ans =
1  4
```

n) size_equal(a,b,...)

Retorna verdadeiro (valor=1) se as dimensões das matrizes a e b são iguais, caso contrário, retorna falso (valor= 0).

Exemplo:

```
> a=[1 2;3 4], b=[4 3;2 1],c=[3 2 1;6 5 4],...
  size_equal (a,b), size_equal(a,c)
a =
  1  2
  3  4

b =
  4  3
  2  1

c =
  3  2  1
  6  5  4

ans =  1
ans = 0
```

o) columns (a), rows (a)

Retorna respectivamente, o número de colunas e linhas da matriz a.

Exemplo:

```
> a=[1 2 3; 4 5 6], columns (a),rows (a)
a =
  1  2  3
  4  5  6

ans =  3
ans =  2
```

p) diag(a), diag(v,m)

Retorna o vetor diagonal generalizada a(i,i), da matriz a.

Exemplo:

```
> a=[1 2 3;3 4 5],  diag(a)
a =
  1  2  3
  3  4  5
```

```
ans =
  1
  4
```

diag (v,m)

Retorna uma matriz diagonal-m,
com os elementos diagonais do vetor v.

m= 0 diagonal principal,
m= 1 diagonal uma acima da principal,
m=-1 diagonal uma abaixo da principal.

Exemplo:

```
> v=[1 2 3 4]; diag(v), diag(v,1), diag(v,-1)
ans =
Diagonal Matrix
  1  0  0  0
  0  2  0  0
  0  0  3  0
  0  0  0  4

ans =
  0  1  0  0  0
  0  0  2  0  0
  0  0  0  3  0
  0  0  0  0  4
  0  0  0  0  0

ans =
  0  0  0  0  0
  1  0  0  0  0
  0  2  0  0  0
  0  0  3  0  0
  0  0  0  4  0
```

q) trace (a)

Calcula o traço da matriz a. A soma dos elementos da diagonal: sum(diag(a)).

Exemplo:

```
> a=[1 2; 3 4], diag(a), trace(a)

a =
  1  2
  3  4
```

```
ans =
  1
  4

ans =  5
```

r) cond (a)

Calcula a norma-2 da matriz, definida como:
norm(a) * norm (inv (a)).

Exemplo:

```
> a=[1 2; 3 4], cond(a)

a =
  1  2
  3  4
ans =  14.933
```

s) det (a)

Calcula o determinante da matriz a, usando a rotina LINPACK.

Exemplo:

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

ans = -2
```

t) r = rref (a), [r,k] = rref (a,tol)

Retorna a matriz linha reduzida da matriz a.
A tolerância tol por default vale eps * max (size (a)) * norm (a, inf).
O argumento k retorna um vetor com o número das colunas sobre as quais a eliminação tem sido efetuada.

Exemplo 01:

```
> a = [1 2 3; 4 5 6; 7 8 9]
a =
  1  2  3
  4  5  6
  7  8  9
```

```
> [r] = rref (a),  rank(r)
r =
  1.00000   0.00000  -1.00000
  0.00000   1.00000   2.00000
  0.00000   0.00000   0.00000

ans =  2
```

Exemplo 02: posto depende da tolerância zero

```
> a = [
  0.00000   0.00001  -0.01000   0.00004   0.00008;
  0.00010   0.00001   0.00001   0.00001   0.00010;
 -0.01000   0.00006   1.00000   0.00009   0.00009;
  0.00001   0.00006   0.00003   0.00007   0.00007;
  0.00003   0.00001   0.00008   0.00009  -1.00000];

> clc; [r] = rref (a,0.01),  rank(r)

r =
  0.00000   0.00000   1.00000   0.00009   0.00000
  0.00000   0.00000   0.00000   0.00000   1.00000
  0.00000   0.00000   0.00000   0.00000   0.00000
  0.00000   0.00000   0.00000   0.00000   0.00000
  0.00000   0.00000   0.00000   0.00000   0.00000

ans =  2

> clc; [r] = rref (a,0.00000000000001),  rank(r)

r =
  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

ans =  5
```

i. EXERCÍCIO:

Encontre o núcleo e a imagem da Transformação Linear (TL) dada pela matriz a.

Definindo a matriz a:

```
> a=[1 2 0 3 1;3 2 0 1 1;3 0 2 1 1]

a =
  1   2   0   3   1
  3   2   0   1   1
  3   0   2   1   1
```

Encontrando a Imagem e o Núcleo da matriz a:

```
> Ia=rref(a'), Na=null(a)

Ia =
  1   0   0
  0   1   0
  0   0   1
  0   0   0
  0   0   0

Na =
  0.320843  -0.182821
 -0.629907  -0.110943
 -0.629907  -0.110943
  0.320843  -0.182821
 -0.023557   0.953172
```

Como a:$R^5 \rightarrow R^3$, posto(Ia)+posto(Na)=2+3=5, dimensão do espaço de saída R^5.

```
>  posto_Ia=rank(Ia), posto_Na=rank(Na), [m,n]=size(a)

posto_Ia =  3
posto_Na =  2

m = 3
n = 5
```

u) dot (x, y)

Calcula o produto interno de dois vetores de mesma dimensão.

Exemplo:

```
> x=[1 2 3 4], y=[2 4 6 8], dot(x,y)
x =
  1   2   3   4

y =
  2   4   6   8

ans =  60
```

v) eig (a), [p, diag] = eig (a)

Encontram os autovalores, autovetores da matriz a.
p - matriz de mudança de base,
diag - matriz diagonal onde,
diag = p^(-1)*a*p.

Exemplo:

```
> a=[1 2;3 2], [p,diag]=eig(a), p^(-1)*a*p
a =
  1   2
  3   2

p =
 -0.70711  -0.55470
  0.70711  -0.83205

diag =
Diagonal Matrix
 -1   0
  0   4

ans =
 -1.00000   0.00000
  0.00000   4.00000
```

3.2 MANIPULAÇÃO DE ELEMENTOS DE UM VETOR/MATRIZ OBTIDOS POR CONTROLE DE ÍNDICE

a) Elementos de um vetor/matriz:

- v(k) é o k-ésimo elemento de um vetor linha ou coluna.
- v(:) lista os elementos de v.
- v(m:n) é a parte do vetor v, compreendida entre m e n.
- a(i,j) é o elemento da matriz a de posição i, j.
- a(j,:) é a j-ésima linha da matriz a.
- a(:,k) é a k-ésima coluna da matriz a.

Exemplo:

```
>v=[1 2 3 4 5], v(2), v(:)
v =
  1   2   3   4   5
ans =  2
ans =
  1
  2
  3
  4
  5

> a=[1  2 3 4; 5 6 7 8; 9 10 11 12]
a =
   1   2   3   4
   5   6   7   8
   9  10  11  12

> a(2,3),a(2,:),a(:,3)

ans =  7

ans =
  5   6   7   8

ans =
   3
   7
  11
```

b) Substituindo elementos de um vetor/matriz:

- a(k,:) = v_linha, substitue a k-ésima linha da matriz a pelo vetor v_linha.
- a(:,j) = v_coluna, substitue a j-ésima coluna da matriz a pelo vetor v_coluna.
- a(j,:) = [] deleta a j-ésima linha da matriz a.
- a(:,k) = [] deleta a k-ésima coluna da matriz a.

Exemplo:

```
>  a = [1 2 3; 4 5 6]; v = [7; 8];

> a(2,3) = v(2)
a =
    1 2 3
    4 5 8

> a(:,2) = v
a =
    1 7 3
    4 8 8

> a(1,1:2) = v'
a =
    7 8 3
    4 8 8
```

3.3 SISTEMAS E INVERSÃO DE MATRIZES:

a) inv(a)

Retorna a matriz inversa da matriz a.

Exemplo 01:

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

ans = -2

ans =
 -2.00000   1.00000
  1.50000  -0.50000
```

b) norm(a), norm(a,p)

Calcula a norma-p da matriz a.
Se o segundo argumento é omitido, p=2 é assumido.

Se a é uma matriz:

- p = 1, norma-1, a maior soma dentre as colunas da matriz a.
- p = 2, maior valor singular da matriz a.
- p = 'inf', norma-infinito,
  a maior soma dentre as linhas da matriz a.
- p = 'fro', norma de Frobenius da matriz a, definida como
  sqrt (sum (diag (a' * a))).

Se a é um vetor ou escalar:

- p = inf, max (abs (a))
- p = -inf, min (abs (a)).

Exemplo:

```
> a = [1 2;3 4]
a =
  1   2
  3   4

> norm(a,1)
ans =  6

> norm(a,2)
ans =  5.4650

> norm(a,inf)
ans =  7
```

c) rcond(a)

Uma estimativa baseada na norma-1. Se a matriz é bem condicionada, o valor é próximo de um, caso contrário, será próximo de zero. O comando não funciona para matrizes esparsas.

Exemplo 01:

```
> a=[1 2;3 4], rcond(a)
a =
  1  2
  3  4

ans =  0.047619
```

Exemplo 02:

```
> b=[1 2;1 2.00002], rcond(b),[x,rcond]=inv (b)
b =

  1.0000  2.0000
  1.0000  2.0000

ans = 1.6666e-006
x =

 1.0000e+005 -1.0000e+005
 -5.0000e+004  5.0000e+004

rcond = 1.6666e-006
ans =

 1.0000e+000  0.0000e+000
 1.0000e+005 -1.0000e+005

> b*x
ans =

  1  0
  0  1
```

d) Ax=b => x=A\b

Seja A uma matriz nxn com determinante não nulo, e b um vetor 1xn.
A solução do sistema matricial Ax=b é dada por:
x=inv(A)*b ou,
x=A\b

Exemplo 01:

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

> b=[3;7]
b =
  3
  7

> det(A)
ans = -2

> x=A\b
x =
  1
  1
```

Exemplo 02:

```
> A =[ 0.0321532   0.3077212   0.5646759;
     0.9663633   0.8617561   0.4065418;
     0.0055062   0.9915849   0.2429759];

> d=det(A)
d =  0.46062

> b=[1,2,3]'
b =
  1
  2
  3

> x = A\b
x =
 -0.66858
  2.98442
  0.18263
```

```
> A*x-b
ans =
 -1.1102e-016
  0.0000e+000
  0.0000e+000
```

e) null (a, tol)

Retorna uma base ortogonal do espaço nulo (Núcleo) da transformação dada por a. A dimensão do espaço nulo é obtida pelo número de valores singulares não maiores que a tolerância tol. Se o argumento tol é omitido, será calculado como:

max (size (a)) * max (svd (a)) * eps.

Maiores informações no Help do programa Octave.

Exemplo 01:

```
 > a=[1 2 3;4 5 6;7 8 9]
a =
   1   2   3
   4   5   6
   7   8   9

> det(a)
ans = -1.3326e-015

> null(a)
ans =
  -0.40825
   0.81650
  -0.40825
```

Exemplo 02:

```
> a=[1 2 3; 0 0 0;0 0 0 ]
a =
   1   2   3
   0   0   0
   0   0   0

> v=null(a)
v =
   0.96362   0.00000
  -0.14825  -0.83205
  -0.22237   0.55470
```

```
> v(:,1)
ans =
  0.96362
 -0.14825
 -0.22237

> p=v(:,1).*v(:,2)
ans =

  0.00000
  0.12335
 -0.12335

> sum(p)
ans = 2.7756e-017
```

f) rank (a, tol)

Encontra o posto da matriz a.
tol - estabelece o limite do quase-zero

Exemplo 01:

```
> a=[1 2 3; 4 5 6; 7 8 9], rank(a), det(a)
a =
  1  2  3
  4  5  6
  7  8  9

> rank(a)
ans =  2

> det(a)
ans = -1.3326e-015
```

Exemplo 02:

```
% rank(a,tol):  tol - tolerância
> a=[1 2 3; 0 4 6; 0 0 0.0001]
a =
  1.00000  2.00000  3.00000
  0.00000  4.00000  6.00000
  0.00000  0.00000  0.00010

> det(a)
ans = 4.0000e-004
```

```
> rank(a,0.00001)
ans =  3

% Neste caso, a matriz a é considerada singular
> rank(a,0.0001)
ans =  2
```

3.4 FATORIZAÇÃO:

a) r=chol (a)

Calcula o fator Cholesky r, de uma matriz simétrica, positiva definida a, onde r'*r=a.

Exemplo:

```
> a=[2 1; 1 2], c=chol(a)
a =
  2  1
  1  2

c =
  1.41421  0.70711
  0.00000  1.22474

> erro = erro=c'*c-ae
erro=
 4.4409e-016  0.0000e+000
 0.0000e+000  -4.4409e-016

> round(erro)
ans =
  0  0
  0  -0
```

b) h = hess (a), [p, h] = hess (a)

Encontra a decomposição de Hessenberg da matriz a. A decomposição de Hessenberg é normalmente usada como primeiro passo na obtenção de autovalores, mas já existem opções melhores.

Decomposição de Hessenberg: a = php',
onde p é uma matriz quadrada unitárias (p'p = I),
h é uma Hessenberg superior:
(h(i,j) =0; i>= j + 1).

Exemplo:

```
> a = [0.913586  0.766178  0.718336;...
   0.990734  0.683166  0.645245;...
   0.090929  0.656789  0.802129]

a =
  0.913586  0.766178  0.718336
  0.990734  0.683166  0.645245
  0.090929  0.656789  0.802129
```

```
>  [p,h]=hess(a)
p =
   1.00000   0.00000   0.00000
   0.00000  -0.99581  -0.09140
   0.00000  -0.09140   0.99581

h =
   0.91359  -0.82862   0.64530
  -0.99490   0.80266  -0.64520
   0.00000  -0.65674   0.68263
```

```
> Ip= p'*p, erro=p*h*p'-a
Ip =
   1.00000   0.00000   0.00000
   0.00000   1.00000   0.00000
   0.00000   0.00000   1.00000

erro =
  0.0000e+000  -1.1102e-016  0.0000e+000
  0.0000e+000  -1.1102e-016  0.0000e+000
  0.0000e+000  0.0000e+000  -1.1102e-016
```

c) [l, u, p] = lu (a)

Encontra a decomposição LU, p*a = l*u, onde
L - triangular inferior,
U - triangular superior,
p – matriz de permutação.
A matriz a não necessita ser quadrada. Para matrizes esparsas, é necessário uma generalização.

Exemplo:

```
> a=[1 2;3 4], [l,u,p]=lu(a),  disp("p*a-l*u:"), p*a-l*u
a =
   1   2
   3   4

> [l,u,p]=lu(a)
l =
   1.00000   0.00000
   0.33333   1.00000

u =
   3.00000   4.00000
   0.00000   0.66667
```

```
p =
Permutation Matrix
  0  1
  1  0

>  p*a-l*u
ans =
  0  0
  0  0
```

d) [q, r] = qr (a), [q, r, p] = qr (a)

Calcula a fatorização qr, onde para [q, r] = qr (a) temos q*r=a ou para [q, r, p] = qr (a) temos a*p=q*r, com

q - é uma matriz ortogonal,
r - uma triangular superior,
p - matriz de permutação.

Exemplo:

```
> a=[1 2;3 4]
a =
  1  2
  3  4

> [q,r]=qr(a)
q =
 -0.31623  -0.94868
 -0.94868   0.31623

r =
 -3.16228  -4.42719
  0.00000  -0.63246

> q*r-a
ans =
  2.2204e-016  -4.4409e-016
 0.0000e+000  0.0000e+000
```

4 GERENCIAMENTO DO AMBIENTE DO OCTAVE

Octave foi escrito em linguagem C e por isso, absorve o estilo do próprio C. Alguns comandos lembram o próprio C com, por exemplo, printf.
O Octave dispõe de comando de gerenciamento tanto para gerenciamento de diretórios como, para gerenciamento do próprio ambiente Octave. Podemos, por exemplo, listar as variáveis por nome, tamanho, número de bits e classe. Podemos deletar uma variável ou todas elas.

a) who, whos

- who lista as variáveis
- whos lista as variáveis com detalhe:nome, size, bytes, class

Exemplo:

```
> who
Variables in the current scope:
ans     dirlist  ii

> whos
Variables in the current scope:
  Attr Name        Size                     Bytes  Class
  ==== ====        ====                     =====  =====
       ans         1x30                        30  char
       dirlist     1x2                         11  cell
       ii          1x1                          8  double

Total is 33 elements using 49 bytes
```

b) dir, ls

- dir,ls lista os arquivos do diretório corrente

c) pwd

- pwd lista o diretório de trabalho corrente.

d) cd

- cd Troca o diretório corrente para o diretório de trabalho do usuário windows.
- cd dir Troca o diretório corrente para o diretório dir.
- dc ~ Troca o diretório corrente para o diretório user

e) cls

- clc, home, ctrl l limpa a tela (clear screan)

f) home

- clc, home, ctrl l limpa a tela (clear screan)

g) ctrl l

- clc, home, ctrl l limpa a tela (clear screan)

h) Comentário

- Iniciamos um comentário linha, com os símbolos # ou %.
- {...} Delimita um bolco de comentário.

Exemplo:

```
##================================
##Uso do comentario:
##ATENÇÃO: O uso do comando <TAB>
##              pode causar erro.
##              Use somente espaço!
##================================

a=1  %valor de a
b=2 %valor de b

{
Isto é
um comentário
de bloco
}
```

i) Lookfor

- lookfor c – mostra todas a incidências da seqüência de caracteres c no help.

Por exemplo, se quisermos saber sobre a função cosseno e suas extensões de sintaxe, procedemos da seguinte forma:

Exemplo: Todas as incidências do vocábulo “cos”

```
>  lookfor cos
acos
acosh
cos
cosh
acosd
cosd
    Computes the discrete cosine transfor
                m of x
   Computes the 2-D discrete cosine transfo
                rm of matrix x
    Computes the inverse discrete cosine
                transform of x
```

```
    Computes the inverse 2-D discrete cosin
                e transform of matrix
tukeywin        Return the filter coefficients of a Tukey window (also kn
 cosine-tap     own as the
Ci              CI compute the cosine integral function define by:
cosint          COSINT compute the cosine integral function define by:
```

j) help, doc

Agora que temos certeza que existe o vocábulo, "cos" podemos usar:
help cos, ou doc cos,
e assim, teremos a descrição da função cosseno em inglês, pelo help.

Exemplo: help cos

```
>  help cos
`cos' is a built-in function

 -- Mapping Function:  cos (X)
    Compute the cosine for each element of X in radians.

    See also: acos, cosd, cosh

Additional help for built-in functions and operators is
available in the on-line version of the manual.  Use the command
`doc <topic>' to search the manual index.

Help and information about Octave is also available on the WWW
at http://www.octave.org and via the help@octave.org
mailing list.
```

k) Diary e o editor Notepad ++

Usamos um editor de texto simples para dar suporte a edição no Octave. É bom que esse editor seja apropriado para edição de fonte. Um dos melhores editores de fonte para programação, com uma variedade de linguagem de programação, e ainda free é o NotePad++. Entre muitas vantagens como, atualização constante e automática com permição, está o fato de abrir como uma planilha, várias arquivos de uma vez disponíveis em abas diferentes: é só clicar e ele abre seu fonte. Constantemente trabalho com vários fontes aberto de uma vez. Ele também tem repõe quando aberto, a posição que estava quando fechado. Vamos usa-lo junto com o comandoo diary (on/off), que quando ligado, salva num arquivo com nome "diary" no diretório em execução, quase tudo que é mostrado na tela quando executamos o Octave.

Exemplo: Uso do diary com o editor NotePad++:

```
clc;
cd ~/octave      %supondo que já foi criado /octave no diretório de trabalho
pwd
diary on
pwd
a=rand(3),b=rand(3)
a+b
a*b
a^2

#=================
##Setando o Editor:
#=================
#A variávvel EDITOR deve ser o caminho do editor:
>EDITOR
ans = C:\Octave\3.2.4_gcc-4.4.0\tools\notepad++\notepad++.exe

#Acessando o notepad++:
#        Você vai precisar abrir no ediotr o arquivo “diary”
#        no diretório de trabalho \octave
>edit

##Você tem agora uma cópia da maioria das saídas do Octave.
```

Por exemplo, o arquivo que segue não faz sentido ser editado diretamente no Octave. Você pode como exercício, salvar esse arquivo no NotePad++ com o nome “poly01.m”.

```
#============================
#Ajustes polinomiais:
#Ex.02
#Graus: 1,2,3
#p=polyfit(x,y,n)
#29/03/2010
#============================

clc;
disp("AJUSTE POLINOMIAL:");
disp("DADOS: OBTIDOS POR y=x^3-x POR PEQUENA PERTURBACAO:");
disp("");
 x=-1.2:0.1:1.2;y=x.^3-x;    %ptos a serem ajustados

#a) Análise do grau: n=1
x=-1.2:0.1:1.2;y=x.^3-x;

disp("a) Poly grau=1:")
```

```
 p=polyfit(x,y,1)    %Ajuste polinomial grau=1
 plot(x,y,"o",   x, polyval(p,x));
 title("Ajuste: Dados e p1:")
 disp("CONCLUSAO: p1 - NAO AJUSTOU")
pause(3)

#b) Análise do grau: n=2
x=-1.2:0.1:1.2;y=x.^3-x;

disp("b) Poly grau=2:")
 p=polyfit(x,y,2)    %comando de ajuste polinomial
 plot(x,y,"o",   x, polyval(p,x));
 title("Ajuste: Dados e p2:")
 disp("CONCLUSAO: p2 NAO AJUSTOU")
pause(3)

#c) Análise do grau: n=3
x=-1.2:0.1:1.2;y=x.^3-x;

disp("c) Poly grau=3:")
 p=polyfit(x,y,3)    %comando de ajuste polinomial
 plot(x,y,"o",   x, polyval(p,x));
 title("Ajuste: Dados e p3:")
 disp("CONCLUSAO: p3 ajustou")
pause(3)

 #Erro abs:
  plot(x,abs(y -  polyval(p,x)), "o");
  title("Erro Absoluto: p3")
  pause(3)

 #Erro relativo:
  plot(x,abs(y- polyval(p,x))/abs(y),"o");
  title("Erro Relativo: p3")
  pause(3)

 #Análise de coeficientes:
 #Em p – grau 3, p(2), p(4) são significativos?
 plot(x, p(1).*x.^3+p(2).*x.^2 +p(3).*x+p(4),
       x,  p(1).*x.^3+p(3).*x, "o");
 grid on ;
 title("p – grau3, Análise de Coeficientes.:")
 pause(3)

#COEFICIENTES SIGNIFICATIVOS:
#ps - polinômio com coef significativos:
#Teste impírico sobre os coef. quasi-zero
```

```
#ps=inline("p(1).*x.^3+p(3).*x") ?

ps=[p(1),0,p(3),0]
#Erro relativo do polinômio ps:

#ps(x) aproxima bem!!!
 ys=polyval(ps,x);
 plot(x,abs( (y-ys)/y ),"o" );
 title("ERRO RELATIVO: ps3 - Coef. Significativos:")

# Conclusão final:
disp("CONCLUSAO: ps3- Ajusta Satisfatoriamente:")
x=-1.2:0.1:1.2;y=x.^3-x;
plot(x,y,"o",   x, polyval(ps,x));
 title("AJUSTE FINAL: Dados e ps3:")
```

FUNÇÕES

4.2 Algumas funções encontradas no Octave:

a) factorial (N)

Retorna o fatorial de N quando N é um inteiro positivo. Se N é um escalar, é equivalente a função produto "prod (1:N)". Quando o argumento for vetor ou matrizes, retorna o fatorial de cada elemento no vetor ou matriz. Para valores não inteiros use a função fatorial generalizada "gamma(x+1)".

Exemplo:

```
> x=6; factorial(x), gamma(x+1)
ans =  720
ans =  720

>#  Só pode ser calculado com a função gamma:
> x=6.001; gamma(x+1)
ans =  721.35
```

b) sum (v), sum (a,k)

sum(v) - soma os elementos do vetor v.
sum(a,[k]) – se k=1, soma as colunas da matriz a;
se k=2, somo as linhas da matriz a;
se k é omitido o defaut é 1, isto é, soma de coluna.

Exemplo: Soma das linhas, colunas, todos os elementos da matriz a.

```
> a=[1 2;3 4]
a =
  1   2
  3   4

>  sum(a)
ans =
  4   6

>  sum(a,1)
ans =
  4   6

>  sum(a,2)
ans =
  3
  7
>  sum(sum(a,2))
ans =  10
```

c) prod(v), prod (a,k)

prod(v) – retorna o produto dos elementos do vetor v.
prod(a,[k]) – se k=1, produto das colunas da matriz a;
se k=2, produto das linhas da matriz a;
se k é omitido o defaut é 1, isto é, produto das coluna.

Exemplo:

```
> a=[1 2;3 4], prod(a,1), prod(a,2), prod(prod(a,2))
a =
  1  2
  3  4

ans =
  3  8

ans =
   2
  12

ans =  24
```

d) rat, rats

s=rat(x) – racionaliza o valor x na variável string s.
s=rats(x, len) – racionaliza o valor x na variável string s, com comprimento no máximo len.
Podemos converter a string em número usando o comando str2num.

Exemplo: rat, rats

```
> rat(e)
ans =
3 + 1/(-4 + 1/(2 + 1/(5 + 1/(-2 + 1/(-7)))))

> rats(e)
ans = 2721/1001

> x=str2num(ans)
x =  2.7183
```

e) Operadores de identificação:

isnumeric (x)	retorna não nulo se x é um objeto numérico
ischar(s)	retorna 1 se s é uma string
isreal (x)	retorna não nulo se x é um objeto numérico real
isfloat (x)	retorna não nulo se x é um objeto numérico pto. flutuante
iscomplex (x)	retorna não nulo se x é um objeto numérico de valor complexo
ismatrix (a)	retorna 1 se a é matriz, c/c zero
isvector (a)	retorna 1 se a é vetor, c/c zero

Exemplo: is...

```
#===============
# Numérico e string:
#===============
>isnumeric("abs")
ans = 0
>ischar("123")
ans =  1
 >isnumeric(123)
ans =  1

#=============
#ponto flutuante:
#=============
 >isfloat (123)
ans =  1
 >isfloat (123.45)
ans =  1

>x=12345, isinteger(x),isreal(x), isfloat(x),y=int32(x), isinteger(y)
x =  12345
ans = 0
ans =  1
ans =  1
y = 12345
ans =  1

 >whos
Variables in the current scope:
  Attr Name      Size              Bytes  Class
  ==== ====      ====              =====  =====
      ans        1x1                   1  logical
       x         1x1                   8  double
       y         1x1                   4  int32

Total is 3 elements using 13 bytes
```

```
#==========
#Complexos:
#==========
 >iscomplex (123)
ans = 0
 >isreal (123)
ans =  1
  >iscomplex (1+2i)
ans =  1

#===========
#matriz e vetor:
#===========
 >a=[1 2;3 4],ismatrix (a)
a =
   1   2
   3   4
ans =  1

 >v=[1 2  3 4],ismatrix (v)
v =
   1   2   3   4
ans =  1

 >v=[1 2  3 4],isvector (v)
v =
   1   2   3   4
ans =  1

 >v=1:4,isvector (v)
v =

   1   2   3   4
ans =  1

>a, isvector (a), ismatrix(a)
a =
   1   2
   3   4

ans = 0
ans =  1
```

4.3 Funções Trigonométricas

O Octave disponibiliza funções trigonométricas em radianos e graus e suas inversas. Também encontramos funções hiperbólicas e suas inversas

i. Funções trigonométricas em radianos:

- sin (x) – seno de x
- cos (x) – co-seno de x
- tan (x) – tangente de x
- sec (x) – secante de x
- csc (x) – cosecante de x
- cot (x) – cotangente de x

ii. Funções trigonométricas inversas em radianos:

- asin (x) – arco seno de x
- acos (x) – arco co-seno de x
- atan (x) – arco tangente de x
- asec (x) – arco secante de x
- acsc (x) – arco co-secante de x
- acot (x) – arco cotangente de x

Exemplo:

```
> x=sin(pi/4)
x =  0.70711

> asin(x)
ans =  0.78540

> ans*4
ans =  3.1416  #pi=3.1416
```

iii. Funções hiperbólicas:

- sinh (x) – seno hiperbólico de x
- cosh (x) – co-seno hiperbólico de x
- tanh (x) – tangente hiperbólico de x
- sech (x) – secante hiperbólico de x
- csch (x) – co-secante hiperbólico de x
- coth (x) – cotangente hiperbólico de x

iv. Funções hiperbólicas inversas:

- asinh (x) – inversa seno hiperbólico de x
- acosh (x) – inversa co-seno hiperbólico de x

- atanh (x) – inversa tangente hiperbólico de x
- asech (x) – inversa secante hiperbólico de x
- acsch (x) – inversa co-secante hiperbólico de x
- acoth (x) – inversa cotangente hiperbólico de x

- atan2 (Y, X) – calcula atan (Y / X) para os elementos Y e X. Teremos erro se X e Y for um vetor de comprimento nulo no plano, ou sem orentação.

O Octave prove funções trigonométricas com argumento em graus.
Por exemplo:

```
cosd (90)
   => 0
cos (pi/2)
   => 6.1230e-17
```

Exemplo:

```
> x=1,   y=(exp(x)-exp(-x))/2,   z= sinh(x)
x =  1
y =  1.1752
z =  1.1752
```

v. Funções trigonométricas em graus:

- sind (x) – seno em graus
- cosd (x) – co-seno em graus
- tand (x) – tangente em graus
  retorna zero quando `x/180' é um inteiro,
  e `Inf' se `(x-90)/180' for inteiro.
- secd (x) – secante em graus
- cscd (x) – co-secnte em graus
- cotd (x) – cotangente em graus

vi. Funções trigonométricas inversa em graus:

- asind (x) – arco seno em graus
- acosd (x) – arco co-seno em graus
- atand (x) – arco tangente em graus
  retorna zero quando `x/180' é um inteiro,
  e `Inf' se `(x-90)/180' for inteiro.
- asecd (x) – arco secante em graus
- acscd (x) – arco co-secnte em graus
- acotd (x) – arco cotangente em graus

Exemplo: trigonometria em graus

```
> x=45, y=sind(x)
x =  45
y =  0.70711

> z=asind(y)
z =  45.000
```

4.4 Funções Exponenciais e Logaritmicas

- exp (x) – calcula a exponencial de x para um x real.
- expm1 (x) – calcula a `exp (x) - 1' numa vizinhança do ponto zero.
- log (x) – calcula o logaritmo natural `ln (x)', para um valor x.
- log1p (x) – calcula o logaritmo `log (1 + x)', expandida numa vizinhança do ponto zero.
- log10 (x) – calcula o logaritmo na base 10 de um valor x.
- log2 (x) - calcula o logaritmo na base 2 de um valor x.
- sqrt (x) – calcula a raiz quadrada de um valor x.

Exemplo: funções logarítmicas

```
> exp(1)
ans =  2.7183

> x=exp(1),log(x)
x =  2.7183
ans =  1

> log10(1000)
ans =  3

> x=log2(16), sqrt(x)
x =  4
ans =  2
```

4.5 Editando uma Função

Podemos editar uma função basicamente de três maneiras:

1. Usando o comando inline():

Ex.01:

```
# nome_func=inline("corpo_da_função")
#                    corpo_da_função deve ser uma variável string
# Apropriado para uso imediato de funções de edição mais simples.

#Exemplo do uso do comando inline para definir f=x^2+1:
>f = inline("x^2+1")
f =
f(x) = x^2+1

 >f(2)
ans =  5

 >f
f =
f(x) = x^2+1
```

2. Usando o comando function ... endfunction:

Ex.02:

```
# function  y=f(x)  comandos_da_função;  endfunction:
#              f – nome da função
#              y - retorno
# Apropriado para programação de funções mais complexas,
# e rotinas de várias variáveis como, por exemplo a integral de Simpson.

#Exemplo do uso para definir f=x^2+1:
> function y = f (x)
   y = x^2+1;
  endfunction;

>f(1)
ans =  2
```

3. Usando o comando @():

Ex.03:

```
#  nome_func=@(var) corpo_da_função;
#                    corpo_da_função - uma variável normal (não string).
# Semelhante ao primeiro caso,
# Apropriado para passagem de parâmetros que sejam funções.
# Isto é, quando uma função tem como parâmetro outra função.
```

```
#Exemplo do uso para definir f=x^2+1:
>h = @(x)x^2+1
h =
@(x) x ^ 2 + 1

>h(1)
ans =  2

 >h
h =
@(x) x ^ 2 + 1
```

a) Funções: functions ... endfunction

As funções são chamadas com seus devidos argumentos, as variáveis nelas usadas são locais, portanto, não interferindo nas variáveis previamente definidas como, por exemplo, nos índices de controle.

Ao editar uma função diretamente na linha de comando do Octave, o comando function inicia a edição da função que será encerrada com o comando endfunction devidamente associado, podendo pular linha sem problema.

A função f pode ser salva com o nome "f.m". No programa Octave várias funções podem ser salvas num único arquivo. Isto porem fará com que o arquivo seja incompatível com o Matlab que restringe cada função ao seu arquivo e nome distintos. Uma boa prática de compatibilidade é manter nome da função vinculado ao nome do arquivo.

Exemplo:

```
>function y = f (x)
 y = x^2+1;
endfunction;

# save filename.ext func
# load filename.ext func

#what – lista file.m no diretório corrente
#type f.m lista arquivo f.m
```

Por uma questão de simplicidade, a partir de agora não indicaremos mais linha de comando ao editar uma função.

A seguir, uma função de uma variável, com dois parâmetros de saída.

Exemplo:

```
>  function [dobro, triplo] = dobrotriplo(x)
       dobro =2*x;
       triplo=3*x;
   endfunction

   > [d,t]=dobrotriplo(2)
       d =  4
       t =  6
```

Uma variável definida no Octave é local por default. Uma variável se torna global ao ser designada como global, no ambiente geral do Octave.

Numa função, quando confirmamos uma variável global, setando como global no interior da função, ao alterarmos seu valor nesta função, em todo lugar onde a variável global apareçer, seu valor será mudado.

Exemplo: A função f armazenada como f.m

```
global N
    % N foi setada com variável global no Octave.
    % Ela agora pode ser alterada pela função f.

function out = f(arg1,arg2)
    global N              % Faz a variável local N se tornar global
    <Cálculos>
End
```

Se o valor de N for mudado na função f, ele será mudado onde quer que esteja.

b) help f

O comando help associado ao nome de uma função,

help nome_função

permite conhecer instruções sobre a função.

Exemplo:

```
> function y = f (x)
## Def.: y=x^2,  x – real ou matriz quadrada
  y = x^2;
endfunction;
```

```
> f(2)
ans = 4

> help f
`f' is a command-line function

Def.: y=x^2,  x – real ou matriz quadrada

Additional help for built-in functions and operators is
available in the on-line version of the manual.
Use the command
`doc <topic>' to search the manual index.

Help and information about Octave is also available on the internet
at http://www.octave.org and via the help@octave.org mailing list.
```

O Octave comenta sobre o help interno disponível no programa. É uma boa prática, sempre que possível recorrer a:

help comando
doc comando.

c) Controle de erro nos parâmetros

Um dos motivos mais comuns de erro no uso das funções esta ligado ao controle de parâmetros. O uso inadequado de parâmetros seja de entrada ou de saída, normalmente acarreta erro. Podemos exercer algum tipo de controle sobre esses parâmetros como veremos a seguir.
O exemplo que segue calcula a média dos elementos de um vetor.

Exemplo 01:

```
>  function retval = media (v)
     retval = sum (v) / length (v);
endfunction

>v=[1 2 3 4 5], media(v)
v =
  1   2   3   4   5
ans =  3

> a=[1 2;3 4], media(a)
a =
  1   2
  3   4

ans =
  2   3
```

Podemos escrever a função anterior da seguinte forma:

Exemplo 02:

```
> function retval = media (v)
if (isvector (v))   # retorna 1 se v é um vetor cc 0
     retval = sum (v) / length (v);
else
    error("Argumento de entra deve ser um vetor!!")
endif
endfunction

> v =[1   2   3   4   5],  x=media(v)
v =
  1   2   3   4   5
x =  3

> a =[1   2; 3   4], y=media(a),
a =
  1   2
  3   4

error: Argumento de entra deve ser um vetor!!
error: called from:
error:   media at line 5, column 6
```

O uso de um argumento que não seja um vetor fará com que a variável retval esteja indefinida e, portanto, se for usada numa atribuição, causara um erro indesejado.

Para prevenir esse tipo de erro uma boa idéia, é sempre retornar algum valor à variável de retorno retval e, um parâmetro de erro, quando o problema for encontrado. Agora, um simples tratamento do parâmetro erro, permite o controle da situação.

Exemplo 03:

```
function [retval,erro] = media (v)
retval = 0;
erro=0;
if (isvector (v))
     retval = sum (v) / length (v);
else
     erro=1;
endif
endfunction

> v=[1 2 3 4], [x,erro]=media(v)
```

```
v =
  1   2   3   4

x =  2.5000
erro = 0

> a=[1 2; 3 4], [y,erro]=media(a)
a =
  1   2
  3   4

y = 0
erro =  1      % erro=1, confirma erro na função.
```

Um problema adicional com as funções, esta associado ao número de parâmetros passado na chamada da função. O exemplo que segue, trata esse problema usando mensagens de erro.

Exemplo 04: media() com tratamento de parâmetros

```
function retval = media (v)
retval = 0;

if (nargin != 1)
  usage ("media (vetor) – entrada com somente um argumento!");
endif

if (isvector (v))
  retval = sum (v) / length (v);
else
  error ("media(vetor) – tem como argumento um vetor");
endif

endfunction;
```

Uso da media com tratamento de parâmetros:

```
> v=[1 2 3 4], m= media (v)

v =
  1   2   3   4
m =  2.5000
```

```
> a=[1 2; 3 4], m= media (a)
a =
  1  2
  3  4

error: media(vetor) - tem como argumento um vetor
error: called from:
error:   media at line 11, column 1

> media(v,a)

usage: media (vetor) - entrada com somente um argumento!
error:   media at line 5, column 1
```

d) Múltiplos parâmetros

Octave permite definir funções com lista de valores de entrada/saída. A sintaxe usada é

```
function [lista-ret] = name (lista-arg)
        corpo
endfunction
```

onde:

- lista-ret, é a lista de parâmetros de saída, separados por vírgula.
- lista-arg, é a lista de parâmetros de entrada, separados por vírgula.

A lista de valores de retorno não necessita ter valor. Quando retorna um parâmetro, recaímos no caso clássico de funções já estudado.

Segue um exemplo de uma função que retorna dois valores, o elemento máximo de um vetor e o índice de sua primeira ocorrência no vetor.

Exemplo: vmax()

```
> function [max, idx] = vmax (v)
idx = 1;
max = v (idx);

for i = 2:length (v)
if (v (i) > max)
    max = v (i);
    idx = i;
endif
endfor
endfunction;

> v=[3 2 9 3 1 9 2 1 ];
```

```
#Uso sem parâmetro de saída, retorna o primeiro:
>  vmax(v)
ans =  9

#Usando os dois parâmetros Max, id:
> [max,id]=vmax(v)
max =  9
id =  3

#Uso indevido de mais que dois parâmetros de saída:
>[max,id,x]=vmax(v)
max =  9
id =  3
error: element number 3 undefined in return list
```

Neste caso particular, os dois valores retornados poderiam ter sido definidos como elementos de uma única matriz, mas isto nem sempre é possível ou conveniente. Os valores a ser retornados podem ser de dimensões diferentes ou mesmo, ser conveniente dar nomes diferentes ao retorno da função. Um exemplo simples disso, fica por conta da clareza, quando cada parâmetro retornado expressa seu verdadeiro significado, como no exemplo anterior.

e) Lista de variáveis de comprimento variado

O número de argumentos da entrada de uma função nem sempre é conhecido previamente. Um exemplo disso pode ser a função que retorne o menor elemento de seus argumentos. Observe que neste caso, não podemos usar um vetor, pois se trata de vários argumentos, não necessariamente numéricos. O exemplo pode induzir a um rraciocínio errado.

```
a=menor(1,2,3),
b=menor(1,2,3,4),
```

que neste caso retorna a=b=1.

Uma possibilidade para escrevermos a função menor é

```
function val = menor (arg1, arg2, arg3, arg4, arg5)
        corpo
endfunction
```

Isso não vai funcionar, pois não temos como prever o tamanho da lista de argumentos.
No corpo da função, os argumentos de entrada, podem ser acessados pela variável varargin. Esta variável é um apontador de pilha que contem todos os parâmetros da entrada.

Vamos usar a função min() do Octave, para definir a função menor(). A função min(v) retorna o menor elemento do vetor v, e sua primeira ocorrência:

Exemplo: min()

```
> [x, id] = min ([1, 3, 0, 2, 0])
  x = 0
  id = 3
```

Não confundir com a função menor() que tem como propósito retornar o menor índice de uma entrada para um número arbitrário de parâmetros.

A função menor() pode assim ser então assim escrita:

```
> function [val,id] = menor (varargin)
     [val,id] = min ([varargin{:}]);
endfunction;

> menor(4,6,7,3,6,5,4,3,7,8)
ans =  3

> [res,id]=menor(10, 12, 2, 4, 2, 3, 4, 5)
res =  2
id =  3
```

Um exemplo ligeiramente mais complexo do uso de varargin é print_arg(), uma função que imprime todos os argumentos da entrada. Aqui supomos uma lista contendo ID, NOME, RG. Tal função pode ser definida como segue:

Exemplo: print_arg()

```
function print_arg (varargin)
clc;
printf("\n RELATORIO: ====================\n")

for i = 1:3:length (varargin)
     printf("\n ID= %d, NOME= %s, RG= %d \n\n",varargin{i}, varargin{i+1},varargin{i+2} );
endfor

printf("\n FIM DO RELATORIO: ==============\n")
endfunction
```

```
#Imprime um arq.:
 print_arg(10, "Pedro", 123456,11, "Jose", 3233456, 12, "Maria", 345456)

 RELATORIO: ====================
 ID= 10, NOME= Pedro, RG= 123456
 ID= 11, NOME= Jose, RG= 3233456
 ID= 12, NOME= Maria, RG= 345456
 FIM DO RELATORIO: ==============
```

f) Retorno numa função

O corpo de uma função definida pelo usuário pode conter uma indicação do retorno ou escape. Esta indicação deixa a função e retorna o controle do próximo comando, logo após sua chamada no programa Octave.

return – Ao contrário da indicação do retorno em C, o uso do comando return no Oitave não pode ser usado para retornar valor de uma função. Em lugar disso, temos que atribuir valores à lista de variáveis de retorno. A indicação do return é restrita e apropriada na saída de um laço ou de uma indicação condicional.

Um exemplo do uso do return, encontra-se na função elem_nao_nulo, que retorna 1 se encontra algum elemento não nulo num vetor.

Exemplo:

```
> function saida = elem_nao_nulo (v)
saida = 0;

for i = 1:length (v)
  if (v (i) != 0)
  saida = 1;
  return;
endif;
endfor;

printf ("Não foram encontrados elementos não nulos!\n");
endfunction;

> v=[0,0,0,0,1,0,20,0,1]; elem_nao_nulo (v)
ans =  1
```

g) Argumento default

Facilidade extremamente desejada quando se trata de banco de dados. Suponhamos que haja necessidade de confirmação do ano no preenchimento de um boleto. Sem dúvida que quem preenche espera que não haja necessidade de digitar o ano todas às vezes novamente.

```
> function c_ano (ANO = 2010)
    printf ("Confirmado ano: %d\n", ANO);
endfunction

> c_ano(1999)
Confirmado ano: 1999

> c_ano()
Confirmado ano: 2010

> c_ano(2011)
Confirmado ano: 2011
```

h) Arquivo script

O arquivo script é uma macro, como no Excel, que contem uma seqüência de comandos do Octave. Pode também ser salvo como arq.m, e quando chamado, executará cada comando como se estivesse numa linha de comando do Octave.

Uma particular utilidade deste recurso é setar o diretório de trabalho do Octave logo que é aberto. Este arquivo deve ser salvo no diretório default do Octave e não no diretório de trabalho, exceto se for o mesmo, o que é nada conveniente. Quando entrar no Octave, e digitar o comando mdir <enter>, sou levado ao meu diretório de trabalho, uma pasta na minha área de trabalho do Windows com o nome octave. Para que o programa funcione sem alteração no seu ambiente Windows, precisa criar o diretório octave na sua área de trabalho.

Exemplo: srt_dir

```
#cd  ~/octave - seta o diretorio de trabalho do user do Windows
#PS1(“ >”)  - seta cursor
# clc -  limpa tela
#ls – lista diretório de trabalho

cd ~/octave
PS1(">")
clc
ls
```

OBS.: tem problema de acesso

i) Função inline

A função inline seta uma função a partir da string contendo o corpo da função. O código a seguir define a função f (x) = x^2

f = inline (“x ^ 2”);

Após isso, é possível avaliar f num ponto x, chamando por f (x).

- inline (str)
- inline (str, arg1, . . .)
- inline (str, n)

Crie uma função interna str a partir de uma seqüência de caracteres. Se for definida com argumentos simples, os argumentos da função gerada são extraídos pelo Octave da própria função, em ordem alfabética. Deve-se notar que i e j não serão considerados como argumentos devido à ambigüidade na sua utilização como uma variável ou como uma constante do ambiente.

Todos os argumentos seguidos por um parêntese são considerados funções. Se os argumentos a partir do segundo, arg1,.., são cadeias de caracteres, devem representar os nomes dos argumentos da função. Se o segundo argumento é um inteiro n, os argumentos definidos imperativamente, serão "x", "P1",. . . , "NP.", independente de aparecerem ou não na função.

vii. argnames (fun)

- Retorna uma matriz strings de caracteres contendo nomes dos argumentos da função inline fun.

viii. formula (fun)

- Retorna a string de caracteres que representa a função inline fun. Isso é equivalente a fôrmula da fun.

Exemplo 01: O seguinte código define a função f (x) = x^2 + 1.

```
> f = inline("x*x+1")        #define f
f =
f(x) = x*x+1

> f(1)                       # exemplo de validação
ans = 2

> f                          #o nome da função lista ela própria
f =
f(x) = x*x+1

> argnames(f)                #lista os argumentos de f
ans =
{
  [1,1] = x
}
```

```
> formula(f)                #retorna a formula de f
ans = x*x+1
```

O próximo exemplo trabalha com uma função de mais que uma variável.

Exemplo 02: Função f(x, y) = x^2+x*y+y^2.

```
> f = inline("x^2+x*y+y^2",'x','y')
f =
f(x, y) = x^2+x*y+y^2

> f(1,2)
ans =  7

> f
f =
f(x, y) = x^2+x*y+y^2
```

ix. vectorize (f)

Cria uma versão vetorizada da função f, trocando todas as ocorrências de *, /, etc., por .*, ./, etc.
Não vetoriza a função já criada. Se caso isso for desejado, é necessário redefinir a função.

Exemplo 02: Função f(x, y) = x^2+x*y+y^2.

```
> f = inline('x^2+x*y+y^2','x','y');

> vectorize(f)                  # versão vetorizada
ans =
f(x, y) = x.^2+x.*y+y.^2

> f                             #f continua a mesma
f =
f(x, y) = x^2+x*y+y^2

> g=vectorize(f)                #g a vetorizada de f
g =                             #f  continua a mesma
f(x, y) = x.^2+x.*y+y.^2
```

j) Funções anônimas

As funções anônimas usam a seguinte sintaxe:

- @(lista_arg) expressão

As variáveis que não se encontram na lista de argumentos não farão parte do conjunto de variáveis passado pela função. As funções anônimas são úteis para criar uma simples função ou para chamadas em outras funções.

O exemplo a seguir mostra as duas formas básicas de uso: Ou defino inicialmente como função anônima ou, na chamada de função, referencio a função chamada como anônima.

Neste exemplo usamos a função quad("f",a,b) – integral definida da função f, no intervalo [a,b] pelo método da quadratura gaussiana.

Exemplo: Função f(x) = x.^2

```
f = inline("x.^2");          #f(x) uma função definida inline.
quad("f",0,1)                #passagem direta usando f

f = @(x) x.^2;#define f como anônima

quad(f,0,1)   #passagem direta usando f
quad(@(x)f(x),0,1)          #passagem por referência anônima.
quad(@(x)x.^2,0,1)          #passagem por referência anônima.
```

k) Particularidades na construção de gráficos:

Exemplo: Função

```
## plot01.m
## Por: Teixeira

#a) Plotando a função seno:
 clf;close;
 x =-3*pi:0.1:3*pi;
 plot (x, sin (x));

 title ("Funções trigonométricas:");
 xlabel ("x");
 ylabel ("y");

 ##text (x0,y0, "comentario no grafico");
 text (3,0.7, "Grafico do seno:");
 legend ("sin (x)");
 grid on;

 pause(10);
 clf;close;

#b) Plota sin, cos, discretamente
 plot (x, sin (x),"-",x,cos(x),"*");
 legend ("sin (x)","cos(x)");
 pause(2);
```

```
clf;close;

#c) hold on/off - liga desliga plotagem sobreposta
hold on; # sobrepoem os graficos
title ("Uso do hold: sobrepoe os graficos ");
plot (x, sin(x),"1");
plot (x, cos(x),"3");
plot (x, sin(x) + cos(x), "5");
hold off

pause(2);
legend ("sin (x)","cos(x)","sin(x)+cos(x)");
pause(5);
clf;close; clc;

#d) fplot e suas facilidades:
fplot ("cos", [0, 2*pi])
fplot ("sin", [0, 2*pi], 201)
fplot ("cos", [-2*pi, 2*pi,  -0.9,1.1], 201)

fplot ("[cos(x), sin(x)]", [0, 3*pi])

 #d1) figure - multiplos gráficos/paginação
fprintf("Grafico usando o comando: figure");
figure(1);
    fplot (@sin, [-10, 10]); pause(5)
title ("fig.: 01");
figure(2);
    fplot (@cos, [-10, 10]); pause(5)
title ("fig.: 01");
#clf;
close(1); pause(2);
close(2); pause(2);
clc;


pause(5);

#e) figure - multiplos gráficos/paginação
clf;
 fplot (@cos, [-2*pi,2*pi]);
 fig02 = gcf ();

%visivel
 set (fig02, "visible", "on");
 title ("fig01");
 pause(3);
```

```
%invisivel
printf ("gcf() – invizivel\n");
 set (fig01, "visible", "off");
 pause(3);

%visivel
 title ("gcf() - vizivel");
 set (fig02, "visible", "on");
 pause(3);

clf; pause(3);
close; pause(3);
disp(“comando -  clc:”) ;pause(3);
clc;
```

l) Organização das funções encontradas no Octave

Muitas das funções padrão do Oitava são distribuídas como funções f.m. Estão razoavelmente organizadas por tópico, nos sub-diretórios

octave-home/lib/octave/version/m,

tornando assim mais fácil encontrá-las. A seguir uma lista dos sub-diretórios, e do que se refere as funções que são encontradas lá.

- ‘audio’ Funções de som de jogos e gravação de som.
- ‘control’ Funções de projeto, simulação e controle de sistema.
- ‘elfun’ Funções elementares.
- ‘finance’ Funções para cálculo de pagamentos, valores de investimentos, e taxas de retorno.
- ‘general’ Manipulação de matrizes como flipud, rot90, e triu, e outras funções básicas como ismatrix, nargchk, etc.
- ‘image’ Ferramentas de processamento de imagem. Estas funções necessitam do X Window System.
- ‘io’ Funções de entrada-saida.
- ‘linear-algebra’ Funções de álgebra linear.
- ‘miscellaneous’ As funções que sobraram.
- ‘optimization’ Funções de otimização.
- ‘path’ Funções para gerenciamento de diretórios, usados no Octave para encontrar as funções, por exemplo.
- ‘pkg’ Instala um pacote externo de funções no Octave.
- ‘plot’ Funções para plotar e imprimir gráficos 2D e 3D.
- ‘polynomial’ Funções de manipulação polinomiais.
- ‘set’ Funções para criar e manipular conjuntos de valores únicos.
- ‘signal’ Funções para aplicações em processamento de sinal.

- ‘sparse’ Funções que disponibilizam o uso de matrizes esparsas.
- ‘specfun’ Funções especiais.
- ‘special-matrix’ Funções que criam formas para especiais.
- ‘startup’ Arquivos para iniciar o sistema Octave mais geralmente.
- ‘statistics’ Funções estatísticas.
- ‘strings’ Funções que disponibilizam o uso de strings.
- ‘testfun’ Unidade de teste de performance sobre outras funções.
- ‘time’ Funções associadas ao controle do tempo.

Por exemplo, no diretório linear-algebra encontramos a função trace.m, que é listada abaixo

Exemplo: type trace.m

```
## Copyright (C) 1993, 1994, 1995, 1996, 1997, 1999, 2005, 2006, 2007, 2008
##           John W. Eaton
##
## This file is part of Octave.
##
## Octave is free software; you can redistribute it and/or modify it
## under the terms of the GNU General Public License as published by
## the Free Software Foundation; either version 3 of the License, or (at
## your option) any later version.
##
## Octave is distributed in the hope that it will be useful, but
## WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
## General Public License for more details.
##
## You should have received a copy of the GNU General Public License
## along with Octave; see the file COPYING.  If not, see
## <http://www.gnu.org/licenses/>.

## -*- texinfo -*-
## @deftypefn {Function File} {} trace (@var{a})
## Compute the trace of @var{a}, @code{sum (diag (@var{a}))}.
## @end deftypefn

## Author: jwe

function y = trace (x)

 if (nargin != 1)
   print_usage ();
 endif

 if (ndims (x) > 2)
   error ("trace: only valid on 2-D objects");
 elseif (isempty (x))
```

```
  y = 0;
 elseif (any (size (x) == 1))
  y = x(1);
 else
  y = sum (diag (x));
 endif

endfunction

%!assert(trace ([1, 2; 3, 4]) == 5);
%!assert(trace ([1, 2; 3, 4; 5, 6]) == 5);
%!assert(trace ([1, 3, 5; 2, 4, 6]) == 5);
%!assert(trace ([]), 0);
%!assert(trace (randn(1,0)), 0);
%!
%!error trace ();
%!error trace (1, 2);
%!error <only valid on 2-D objects> trace(reshape(1:9,[1,3,3]));
```

Esse simples exemplo nos mostra, a riqueza de operadores disponíveis no Octave e a elegância e clareza na construção do código nas funções. A função trace() é construída com base nos argumentos:

i) O número de parâmetros na entrada deve ser um;
ii) A dimensão do parâmetro um ou dois.
iii) Encontra o traço como a soma dos elementos da diagonal.

4.6 Recursividade

Com algumas restrições, a recursividade de funções é permitida. A restrição se deve ao caso de algumas funções chamarem funções que não podem ser recursivas, como por exemplo, funções para resolver EDOs.

Uma função é recursiva quando chama a si própria direta ou indiretamente. Um exemplo clássico é a função fatorial para um inteiro. Neste caso, para valores maiores, retorna ponto flutuante.

Exemplo:

```
>function retval = fatorial (n)
if (n > 0)
    retval = n * fatorial (n-1);
else
    retval = 1;
endif
endfunction

> fatorial(5)
```

```
ans =  120

> fatorial(170)
ans =  7.2574e+306        # aproximação em ponto flutuante

> fatorial(171)
ans = Inf                 # limite de  Overflow
```

Esta função é recursiva porque chama a si próprio diretamente. Eventualmente termina, pois esta sempre decrescendo, limitada pelo zero. Sua limitação fica por conta da variável interna restritiva max_recursion_depth, que especifica o limite de recursividade do Octave, prevenindo recursividades infinitas ou, como em qualquer outra função quando tivermos overflow. Verifique o valor desta variável em seu computador, normalmente vale 256.

5 ESTRUTURAS DE CONTROLE

5.1 Operadores de comparação

Operadores de comparação para determinar por exemplo, se são iguais. Todos os operadores de comparação do Octave retornam 1 como verdade e 0 como falso.

A igualdade de matrizes é feita termo a termo e retorna a matriz lógica como resultado. Se um operador é escalar e o outro é matriz, o escalar é comparado com todos os elementos da matriz.

Exemplo:

```
> [1, 2; 3, 4] == [1, 3; 2, 4]
ans=
1   0
0   1

> 1 == [1, 3; 1, 4]
ans =
  1   0
  1   0
```

a) Resumo dos operadores lógicos:

x < y, verdade se x é menor que y.
x <= y, verdade se x é menor ou igual a y.
x >= y, verdade se x é maior ou igual a y.
x > y, verdade se x é maior que y.

x == y, verdade se x é igual a y.

x != y,
x ~= y, verdade se x é diferente de y.

Casos particulares como as strings, podem ser comparadas com o operador strcmp (s1,s2). O operador isequal(x1,x2, ..., xn) retorna verdadeiro se x1, x2, ..., xn são iguais.

Exemplo:

```
> [1, 2; 3, 4]  >  [1, 3; 2, 4]
ans =
  0  0
  1  0

> [1, 2; 3, 4] !=  [1, 3; 2, 4]
ans =
  0  1
  1  0
```

5.2 Expressões booleanas

a) Operadores booleanos

Uma expressão booleana elemento por elemento, é uma combinação de comparação de expressões usando os operadores:

- |, operador "ou"
- &, operador "e"
- !, operador "não"

Uma expressão booleana é verdadeira se o resultado do cálculo das combinações lógicas das expressões booleanas for verdadeira. O valor será considerado falso se for zero, e verdadeiro de outra forma.

Uma expressão booleana elemento por elemento, pode ser usada de forma ampla. Podemos usá-la num tratamento de controle com o if ou while. Assim, se o valor de uma matriz é usada como condicional, num controle if ou while, somente será verdadeiro se todos os elementos da matriz forem não nulos.

Exemplo:

```
 >  [1, 0; 0, 1] & [1, 0; 2, 3]
ans =
  1  0
  0  1
```

b) Operadores de incremento

Operadores de incremento aumentam ou diminuem 1 numa variável, escrito com o símbolo de "++" ou "--". Temos duas variantes do operador "++", o pré-incremento e o pós-incremento:

- x++, retorna o valor de x, depois incrementa.
- ++x, incrementa, depois retorna o valor de x.

- x--, retorna o valor de x, depois decrementa.
- --x, decrementa, depois retorna o valor de x.

Exemplo:

```
> x=1, x++, x       % retorna o valor de x, depois incrementa.
x =  1
ans =  1
x =  2

> x=1, ++x, x       % incrementa, depois retorna o valor de x.
x =  1
ans =  2
x =  2
```

c) Precedência de Operadores

Precedência de operadores determina, quando diferentes operadores aparecem numa expressão, quem deve ser operado primeiro. Um exemplo clássico é a precedência da multiplicação "*"sobre a adição "+".

Exemplo:

```
> 2+3*4             % primeiro a multiplicação depois a adição.
ans =  14

> (2+3)*4           % primeiro a soma, depois a multiplicação.
ans =  20           %O parênteses força a precedência

> -2^3, -(2^3)      % Neste caso indiferente pois,
ans = -8            % "^" tem precedência por "-".
ans = -8
```

Tabela de operadores do Octave, em ordem decrescente de precedência.

- Separadores:
  ';', ','.
- Atribuição:
  '=', '+=', '-=', '*=','/='. % Agrupados da direta para a esquerda
- Lógico "ou" e "e":
  '||', '&&'.
- "ou" e "e":
  '|', '&'.
- Relacional:
  '<', '<=', '==', '>=', '>', '!=', '~='.

- Dois pontos:
  ‘:’.
- Adição e subtração:
  ‘+’, ‘-’.
- Multiplicação e divisão:
  ‘*’, ‘/’, ‘\’,
  ‘.\’, ‘.*’, ‘./’.
- Transposição:
  ‘’’,
  ‘.’’
- Adição e subtração unária, incremento e decremento, e o “não”:
  ‘+’, ‘-’, ‘++’, ‘--’,
  ‘!’, ‘~’.
- Exponenciação:
  ‘^’, ‘**’,
  ‘.^’, ‘.**’.

5.3 Avaliação

Normalmente, avaliamos expressões simplesmente digitando na linha de comando do Octave ou pedindo que o Octave interprete comandos que foram salvos em um programa.

Algumas vezes, podemos encontrar a necessidade de avaliar uma expressão que tenha sido escrita e armazenada numa string. É isto exatamente que o comando eval, uma função interna do Octave fará por você.

a) eval (str, str_erro)

Interpreta a string str, como se fosse uma linha de programa do Octave. Se houver falha, avalia a string str_erro.

O seguinte exemplo faz a variável a assumir o valor aproximado de pi.

Exemplo:

```
> eval("a = acos(-1);"); a
a = 3.1416

> sqrt('a')
error: octave_base_value::sqrt (): wrong type argument `sq_string'

> eval("sqrt('a')","b=2") # A primeira avaliação causa erro, logo a segunda passa a ser avaliada.
b =  2
```

b) Chamada de funções pelo nome

A função feval permite que se chame uma função a partir de uma string que contenha seu nome. Isto é útil ao escrever uma função que precisa de chamada por usuários. A função feval toma o nome da função chamada como seus argumentos, que serão passados a função.

O seguinte exemplo faz uso de feval para encontrar a integral definida de função pelo método de Simpson.

Exemplo: Integral definida pelo método de Simpson

```
> function result = simpson(f, a,b,n)
# uso: simpson(f, a,b,n)
# f : a string nome da funcao a ser integrada.
# a, b: extremos da integral
# n: numero da particao
# Algoritmo:
#   I= delta/3*(f(a)+4*SI+2*SP-f(b))
#

delta = (b-a)/n;
SI=SP=0;
x=a;

for i = 1:n/2
    x=x+delta;
    SI=SI+feval(f,x);
    x=x+delta;
    SP=SP+ feval(f,x);
endfor

result = delta/3*( feval(f,a)+4*SI+2*SP- feval(f,b));
endfunction


#Definido uma função a ser integrada:
> function y = f (x)
    y = x^2;
endfunction


# Chamada da rotina simpson():
# Calcula a integral definida da função y=x^2,
# no intervalo [0,1],  com partição  n=100.
> simpson("f", 0,1, 100)

ans =  0.33333
```

5.4 Laços

Laços ocupam um lugar relevante no controle e núcleos de repetição.

a) while

```
while CONDIÇÃO
    COMANDOS
endwhile
```

Enquanto CONDIÇÃO é satisfeita, executa COMANDO.

Exemplo:

```
> function f (n)
    while (n-- > 0)          % operador n++, n—da linguagem C
        printf (“ n =  %d \n”,n);
    endwhile
  endfunction

> f(5)
 n =  4
 n =  3
 n =  2
 n =  1
 n =  0
```

Observe que n começa ser impresso com valor 4 e decresce até 0.

b) for

```
for var = expressão
 COMANDOS
endfor
```

Enquanto expressão varia, executa COMANDO.

Exemplo:

```
> function S=f (n)
S=0;
for i=1:n
    S=S+i;
endfor;
endfunction;

> f(5)
 ans = 15
```

c) do

```
do
 COMANDOS
until (CONDIÇÃO)
```

Executa COMANDO até que CONDIÇÃO seja satisfeita.

Exemplo:

```
> function y=fib(n)
y=ones (1, n);
i = 2;
do
      i++;
     y (i) = y (i-1) + y (i-2);
 until (i == n)
endfunction


> fib(3)
ans =
  1   1   2

> fib(5)
ans =
  1   1   2   3   5

> fib(8)
ans =
   1   1   2   3   5   8   13   21
```

5.5 Caminhos Condicionais

a) if

```
if (COND) ... endif
if (COND) ... else ... endif
if (COND1) ... elseif (CONDn) ... endif
if (COND1) ... elseif (CONDn) ... else ... endif
```

Exemplo: Iniciando um bloco: if, elseif, else

```
x = 1;
if (x == 1)
```

```
        disp ("O valor de x e': ums");
    elseif (x == 2)
        disp ("O valor de x e': dois");
    elseif (x ==3)
        disp ("O valor de x e': tres");

     else
        disp ("Não e' 1, 2, nem 3.");
 endif

um
```

b) switch

```
switch  TRATAMENTO
    Inicio do bloco do switch:
    case 1
    case 2
    ...
    otherwise
    ...
endswitch
```

O exemplo a seguir testa uma resposta: "yes"ou "no". Seta a variável

1. rer=1 se YES, e suas variações;
2. ret=0 se NO, e suas vriações;
3. caso contrário, dá erro com valor inválido.

A variável "ret" não é impressa, caso queira saber o seu valor tem que solicitar.

Exemplo:

```
>  yesno = "yes"              % inicio do bloco

 > switch yesno
        case {"Yes" "yes" "YES" "y" "Y"}    % entre chaves!!!
          ret = 1;
        case {"No" "no" "NO" "n" "N"}
          ret = 0;
        otherwise
          error ("Valor invalido!");  %usa a função error:
   endswitch

>  ret        %pede o conteúdo da valor

ret =1
```

6 GRAFICOS

a) clf

Limpa as figuras instanciadas no Windows. Coloca as propriedades gráficas na forma standard.

6.2 Gráficos 2D

a) plot(x,y[,fmt])

Plota uma curva sobre os pontos (xi ;yi). Com a string fmt, podemos setar tipo de linha, estilo, cor,... . Para maiores detalhes veja help plot.

Exemplo 01: Gráfico da função cos(x)

```
>  x = -10:0.2:10;
> plot (x, cos(x)); %plota a func. cos(x)
```

Exemplo 02: Título, eixos e legenda

```
 x = -10:0.2:10;
y1= cos(x);y2=sin(x);

plot (x, y1, x, y2); %plota a func. cos(x), sin(x)
 title ("Gráfico do sin(x), cos(x),  x = -10:0.1:10");
 xlabel ("x");
 ylabel ("y");
#text (x0,y0, "comentario no grafico");
 text (3,0.7, "Comentário localizado P(x0,y0):");
 legend (“cos(x)”,"sin(x)");
grid on;  # on/off – o grid
```

Exemplo 03:

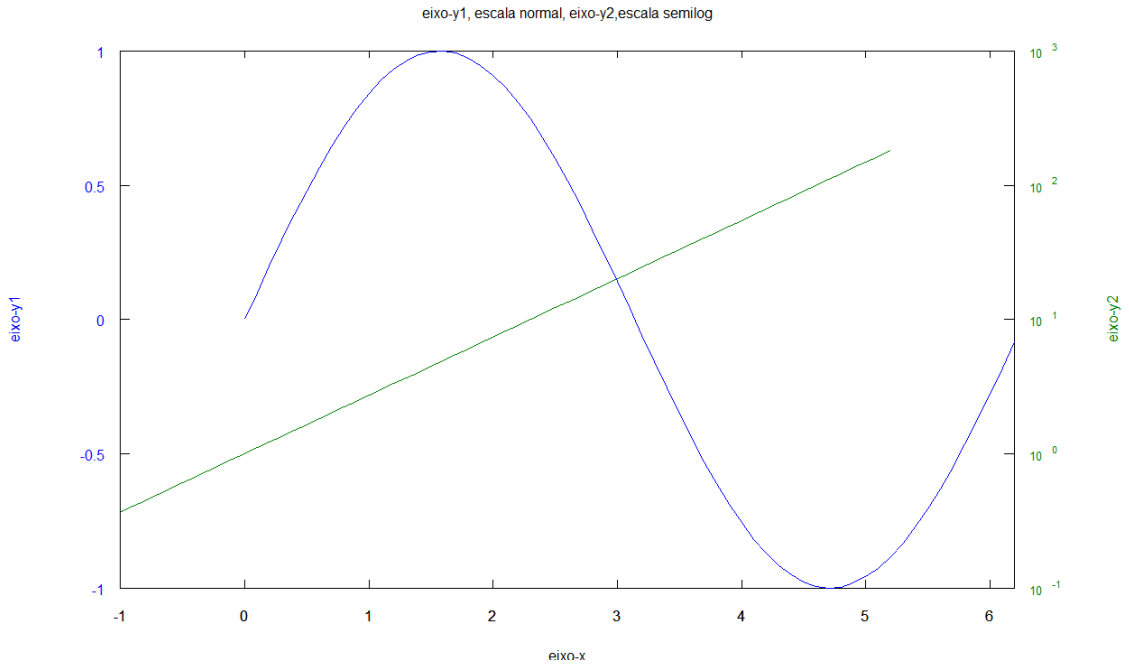
```
 x = -10:0.2:10;

plot (x, cos(x), "o;cos;1","markersize", 5);
% plota cos(x), usando a figura “ o”, cor 1, e tamanho 5

plot (x, cos(x),  "o; cos; 1", "markersize", 5,   x ,2*sin(x), "*; sin; 2", "markersize", 3);
%plota cos(x), 2*sin(x) juntas
```

b) plotyy(x1,y1,x2,y2[,fmt])

Plota duas curvas sobre os pontos (x1,y1) e (x2,y2). A string fmt refere-se às propriedades básicas da função plot. O detalhe fica pro conta que as escalas de y1, y2 podem ser diferentes, como mostra o exemplo a seguir:

Exemplo:

```
clf;
x = 0:0.1:2*pi;
y1 = sin (x);
y2 = exp (x - 1);

ax = plotyy (x, y1, x - 1, y2, @plot, @semilogy);
#ax = plotyy (x, y1, @plot,  x - 1, y2,@semilogy);

%y1 – escala plot (standard)
%y2 – escala semilogy

xlabel ("eixo-x");
ylabel (ax(1), "y1 - sin(x)");
ylabel (ax(2), "y2 - exp(x-1)");
title ("eixo-y1, escala normal, eixo-y2,escala semilog");
#legend ("sin (x)","exp(x-1)"); ##não funciona bem!!!
```



Ver também: mesh, contour, stairs, xlabel, ylabel, title, print.

c) fplot(fn,lim,n,fmt)

- Plota uma função fn, com os limites definidos.
- fn – pode ser uma string, uma função do Octave ou, uma função inline.
- Os limites do gráfico pode ser dado pelo domínio [x1,x2] ou, pelo quadro limítrofe [x1,x2,y1,y2].
- n – um inteiro que representa o número da partição na qual o gráfico será plotado.
- fmt – refere-se a formatações possíveis.

Exemplo:

```
% plota cos(x), para x no intervalo [2,11],
% y no intervalo [-0.9, 0.9]

##Ex.01 ==========================
 clf;
 fplot ("cos",  [2,11,   -0.9, 0.9], 100, "1")

##Ex. 02 ==========================
 clf;
fplot ("cos",  [0, 3*pi, -0.8, 0.9], 100, "1")
 title ("Gráfico com controle de moldura:");
 xlabel ("x");
 ylabel ("y");
#text (x0,y0, "comentario no grafico");
 text (3,0.7, "Comentário localizado P(x0,y0):");
 legend (“cos(x)”);
grid on;  # on/off – o grid


##Ex. 03 ==========================
 fplot ("[sin(x), 3*sin(2*x)]",  [0,3*pi, -1,3], 100)

##Ex. 03 ==========================
 fplot ("[x, 1/x]",  [0,1, 0,10], 100)
```

d) semilogx(x,y[,fmt]), semilogy(x,y[,fmt]), loglog(x,y[,fmt])

Plota usando escalas log respectivamente no eixo-x, eixo-y, e em ambos os eixos.

Exemplo:

```
% plota usando escalas semilog e loglog:

##Ex.01 – semilogx, semilog no eixo x
x=100:100:1000000;y=1./log(x); semilogx(x,y)
```

```
##Ex.01 – semilog, semilog nos eixos x e y
 x=10:10:10000;y=x.^2; loglog(x,y);
```

As funções hist, bar, barh, stairs, e stem que seguem, são funções de domínio discreto:

e) hist(v,m,n)

Plota um histograma representando a distribuição do vetor v.

- v – vetor linha ou coluna.
- m – número de colunas do histograma.
- n - parâmetro de controle da altura da coluna.

Exemplo:

```
% plota um histograma de um vetor randômico normalizado

##Ex.01 =================
d=randn(1000,1); hist(d,6,10);        %histograma com 6 colunas, e altura próximo de 1.

##Ex.02 =================
 d=randn(1000,1); hist(d,20,10);        %histograma com 20 colunas, e altura próximo de 1.

##Ex.02 =================
d=randn(1000,1); hist(d,30,100);      %histograma com 30 colunas, %e altura próximo de 10.
```

f) bar(x,y), bar(y), bar(a)

Plota um gráfico de barra a partir dos vetores x,y.

- Se somente um vetor é apresentado, assume que é o vetor y, e para x assume o índice dos elementos.
- Se x, y são parâmetros de entrada, x necessita estar em ordem crescente.
- Se a(m,n) é uma matriz mxn, assume um conjunto de histogramas de m-colunas, repetidas n-vezes.

Exemplo:

```
% plota um gráfico de barra:

## Ex.01 - Plota um gráfico de barra com dez
#            colunas randômicas, indexadas de 1 a 10.
bar (rand (10, 1));

## Ex.02 - Plota um gráfico de barra com dez
#            colunas randômicas, indexadas de 3 a 13.
 bar ([3 4 5 6 7 8 9 11 12 13], rand (10, 1));
```

```
##Ex.01 Plota as linhas da matriz a em sequência
 a=[1 2 3; 2 3 4], bar(a)

##Ex. 03 -  Plota um gráfico de barra com dez
##              grupos de cinco colunas randômicas.
##             A variável h indexa os grupos.
 h = bar (rand (5, 10));

#Plota o gráfico de barra associado
#ao primeiro grupos, com valor base em y de 0.5.
set (h(1), "basevalue", 0.5);

# Análogo ao anterior para grupo 2, valor base em y de 0.3.
#APLICAÇÃO: Nível de insumo aceitável no mínimo 0.3
set (h(2), "basevalue", 0.3);
```

g) stairs(x,y), stairs(y)

Plota um gráfico escada a partir dos vetores x(n+1),y(n).

- Dados os vetores x,y de mesma dimensão, a escada é obtida da seguinte maneira:
  para[x1,x2], a escada assumem valor y1,
  para[x2,x3], a escada assumem valor y2,
  ...
- O valor yn fica perdido, mas não pode ser omitido, pois a escada perderia o último degrau.

Exemplo:

```
% plota uma função escada:
> stairs([1 2 3 4 5] ,[1 3 5 2])
%plota uma função escada de quatro degraus 1, 3, 5, 2.
%O primeiro degrau liga o intervalo [1,2],
 %o segundo

> stairs([1 2 4 7] ,[1 3 5])
%plota uma função escada de três degraus 1, 3, 5.
```

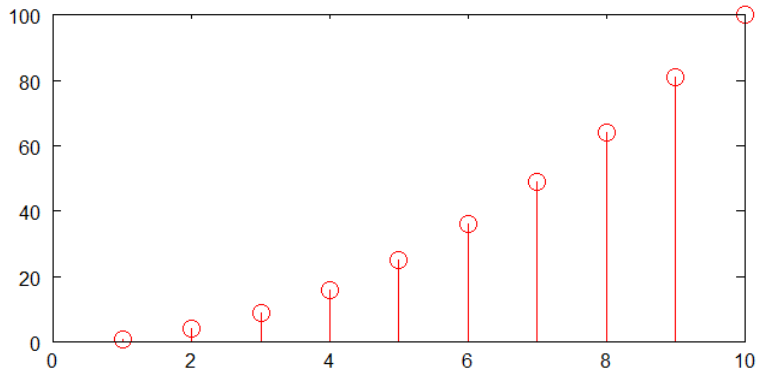
h) stem(x,y, s)

Plota um gráfico haste a partir dos vetores x,y.

- O string s determina a cor
  "r" – cor vermelha é default,
  " b" – cor azul.
- Ver help stem3 e 3D.

Exemplo:

```
% plota uma função haste:
 x=0:10;
y=x.^2;
stem(x,y,"r")
```



Exemplo:

```
% Muda a cor da segunda serei para verde,
% move a linha base da primeira para -0.5:
 x = [0 : 10].';  y = [sin(x), cos(x)]
 h = stem (x, y);
 set (h(2), "color", "g");

 set (h(1), "basevalue", -0.5)
```

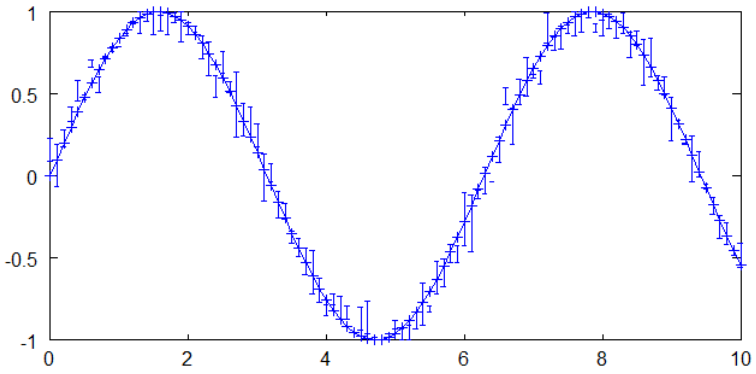
i) errorbar(x,y, y_erro_mim, y_erro_max)

Função plota x,y, e uma barra de erro vertical ligando y_erro_min até y_erro_max.
A função de erro é muito geral. Para maiores informações consulte "> doc errorbar"

Exemplo:

```
% plota função e barras de erro associado:
 x = 0:0.1:10; y = sin (x);

%cria um erro
```

```
yM = 0.1 .* randn (size (x));
ym = -0.1 .* randn (size (x));

%função erro
errorbar (x, sin (x), yM, ym);
```



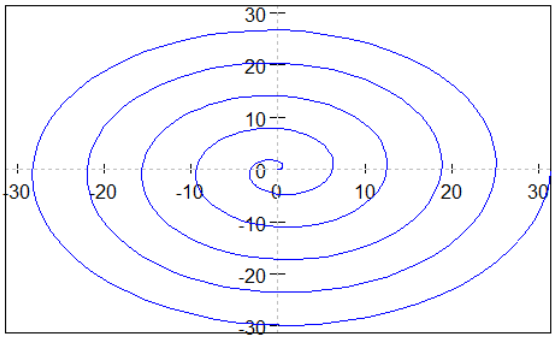
j) polar (theta, rho, fmt)

Plota uma curva em coordenadas polares (theta,rho).

- theta – ângulo do vetor posição,
- rho – comprimento do vetor posição.
- fmt - parâmetro de formatação convencional similar ao comando plot.

Exemplo:

```
% plota função polar r = tetha, [0,10pi], na cor verde:
 polar (0:0.1:10*pi, 0:0.1:10*pi,”g”);
```



k) pie(y,labels)

Plota um gráfico de piza com porcentuais dos valores de y.

- y – vetor da variável principal da piza.
- labels - conjunto de strings nomes relativo a cada fatia da piza. Se labels for omitido, aparecerão porcentagens para as fatias.

Exemplo:

```
% plota um gráfico de piza:
##Ex.01 – porcentuais diretos:
pie([1 2 3])

##Ex.02 – relaciona os índices aos seus conteúdos:
pie([1000 2000 3000],{"soja" "milho" "arros"})
```

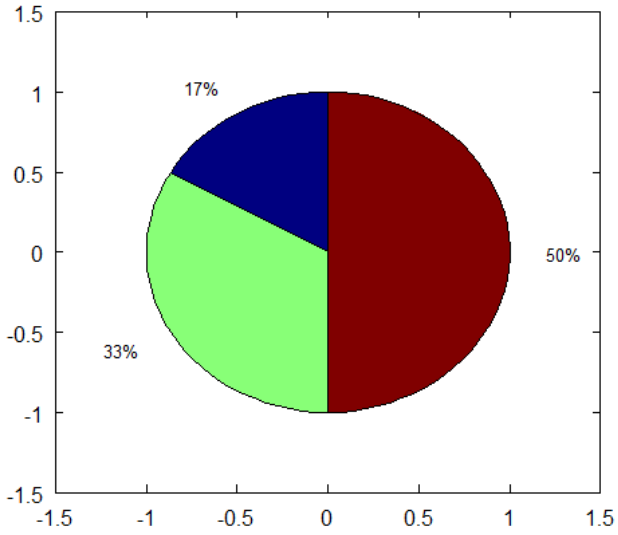


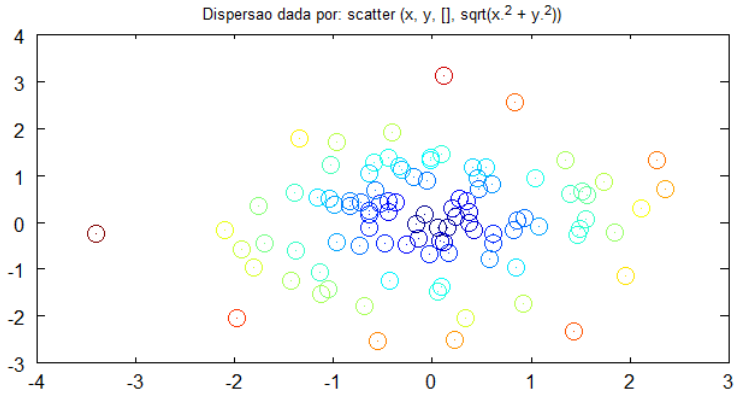
Gráfico: pie([1 2 3])

l) scatter (x, y, s, c)

Traçar um gráfico de dispersão dos dados cartesianos (x,y) . Um marcador é representado em cada ponto definido pelo vetores x,y. O tamanho dos marcadores utilizados é determinado por s, que pode ser um escalar, um vetor de mesmo comprimento de x e y. Se s não é dado ou é uma matriz vazia, então o valor padrão de 8 pontos é usado. O parâmetro c, uma variável real, define a cor.

Exemplo: plota uma dispersão usando scatter()

```
## Uso de scatter(x,y, s,c)
# Dispersão de tamanho s=5,
# cor c=sqrt(x.^2 + y.^2)
 x = randn (100, 1);  y = randn (100, 1);
 scatter (x, y, [], sqrt(x.^2 + y.^2));
 title ("Dispersão dada por scatter  s=5, c=sqrt(x^2 + y^2) ");
```



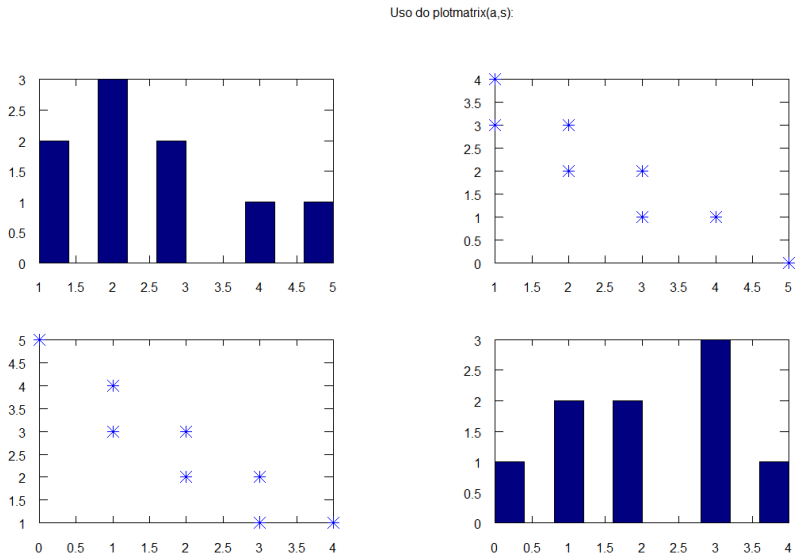
m) plotmatrix (a,s)

Plota a dispersão das colunas da matriz a, uma contra a outra, para uma matriz de bloco quadrada de dispersão, cuja dimensão é o numero de colunas de a. Na diagonal, ao cruzar uma coluna com ela mesma, plota o histograma desta coluna.

- No eixo x, aparecem os valores das colunas da matriz
- No eixo y a freqüência em que eles aparecem.
- O marcador é determinado pela string s.

Exemplo: plota a dispersão de uma matrize,

```
##Ex.01 Uso de plotmatrix(a,s)
a=[1 2 3 4 5 4 3 2 1;4 3 2 1 0 1 2 3 4]'
 plotmatrix (a,'*')
title ("Uso do plotmatrix(a,s): ");
```



6.3 Gráficos 3D

Segue alguns comandos usados para gráficos 2D e 3D:

- title(string) - escreve a string como título do gráfico.
- xlabel(string) – seta a string referência do eixo-x.
- y label(string) – seta a string referência do eixo-y.
- zlabel(string) – seta a string referência do eixo-z.
- xis(v) – seta os limites dos eixos de plotagem.
- v is a vector of form
  v = (xmin, xmax, ymin, ymax[, zmin zmax]).
- hold [on|off] – controla a saída gráfica, quando ligado permite a sobreposição gráfica.
- clf- clears the graphics window.
- close – fecha a janela gráfica.

OBS.: Todas as linhas nos exemplos a seguir, são comando ou comentários. Executar todas elas.

Ex.01 – superfície em malha

```
##
## meshgrid, mesh:
##
 clc,clf;
 printf("Plot3d: meshgrid, mesh \n\n");
 #x = -2:0.1:2;
 x = y=-2:0.1:2;
 [u,v] = meshgrid(x,y);
 z = sin(u.^2-v.^2);

 mesh(x,y,z); #figura

title("mesh - superficie")
 text (1,2,3, "z = sin(x^2-y^2)");
 xlabel ("eixo - x");
 ylabel ("eixo - y");
zlabel ("eixo - z");

# pause(8);
#clf;close; # limpa e fecha a janela grafica
```

Ex.02 – curva de nível em superfície de malha

```
##
## meshgrid, mesh, mechc
##
```

```
# Ex.: Sombrero
tx = ty = linspace (-8, 8, 41)';
[xx, yy] = meshgrid (tx, ty);
r = sqrt (xx .^ 2 + yy .^ 2) + eps;
tz = sin (r) ./ r;
mesh (tx, ty, tz); #plota o sombrero mesh
pause(3)
meshc (tx, ty, tz);  #plota as curvas de nivel do sombrero

title("Sombrero com curva de nível:")
 xlabel ("eixo - x");
 ylabel ("eixo - y");
 zlabel ("eixo - z");
```

Ex. 03 - hélice

```
##
## plot3 - caminho em 3D
##

t = 0:0.1:10*pi;
r = linspace (0, 1, numel (t));
z = linspace (0, 1, numel (t));

plot3 (r.*sin(t), r.*cos(t), z,   “; helice; “);

title("plot3 - caminho em 3D:")
 xlabel ("eixo - x");
 ylabel ("eixo - y");
 zlabel ("eixo - z");
```

Ex. 04 – senoide complexa

```
##
## plot3 - caminho em 3D
##

 t = linspace(0,pi);
 plot3 (t, exp(2i*pi*t), ";senoide complexa;");

 xlabel ("eixo - t");
 ylabel ("eixo - x");
 zlabel ("eixo - y");
```

Ex.05 – hiperbolóide – superfície e curva de nível

```
##
## surf, e surfc
##

clf;
n=10;
np=30;
x = y = linspace (-n, n, np)';
[X, Y] = meshgrid (x,y);
Z = X.^ 2 - Y.^ 2;
surf (X,Y,Z); #plota o hiperboloide

pause(2);
surfc (X,Y,Z); #plota curvas de nível do hiperboloide

title("surf – superficies em 3D:")
 xlabel ("eixo - x");
 ylabel ("eixo - y");
 zlabel ("eixo - z");
```

Ex.06 – sombrero com curva de nível

```
##
## surf, e surfc - curva de nivel
##
# Ex.: Sombrero
clf;
x = y = linspace (-8, 8, 41)';
[X,Y] = meshgrid (x, y);
r = sqrt (X .^ 2 + Y .^ 2) + eps;
Z = sin (r) ./ r;
surf (X,Y,Z); #plota o sombrero surf
surfc (X,Y,Z); #plota o contorno sombrero surf

title("surf, surfc – sombrero com curva de nível:")
 xlabel ("eixo - x");
 ylabel ("eixo - y");
 zlabel ("eixo - z");
```

Ex.07 – parabolóide com campo de vetores☹?

```
##
## meshgrid, mesh
##
# Ex.: Sombrero
clf;
x=y = linspace (-5, 5, 20)';
[X,Y] = meshgrid (x,y);
Z = X .^ 2 +Y .^ 2;

surf (X,Y,-Z);  #plota o paraboloide
surfnorm (X,Y,-Z); # campo normal da superfície
```

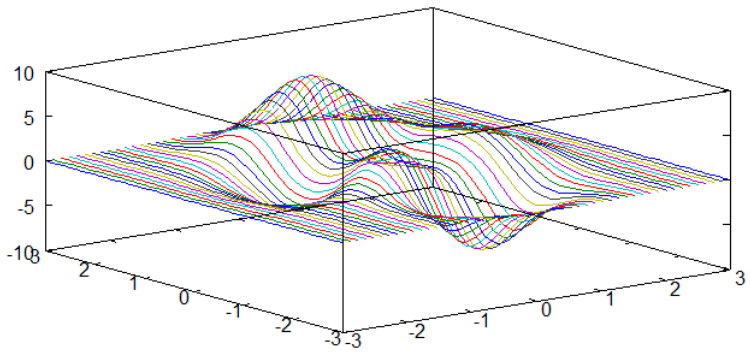
Ex.08 – Corte com curva de nível por cor:

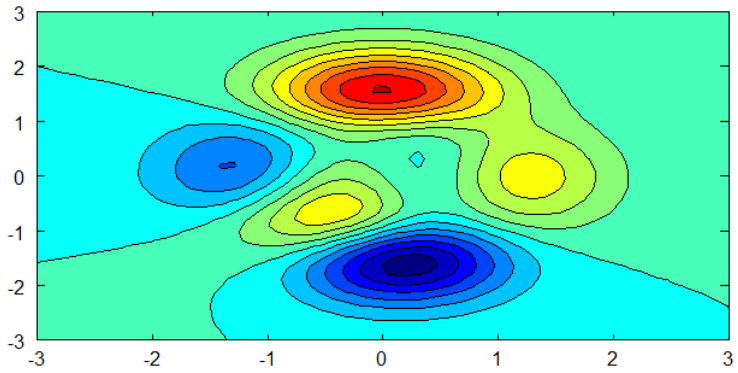
```
#a) Curva de nível com cor, corte y=0:
[x, y, z] = meshgrid (linspace (-8, 8, 32));
w = sin (sqrt (x.^2 + y.^2 + z.^2)) ./ (sqrt (x.^2 + y.^2 + z.^2));
slice (x, y, z,   w,   [], 0, []);
title("slice – fatia, corte y=0, curva de nível por cor:")
 xlabel ("eixo - x");
 ylabel ("eixo - y");
 zlabel ("eixo - z");

#b) Curva de nivel com cor, corte z=0:
[x, y, z] = meshgrid (linspace (-8, 8, 32));
w = sin (sqrt (x.^2 + y.^2 + z.^2)) ./ (sqrt (x.^2 + y.^2 + z.^2));
slice (x, y, z,   w,   [], [], 0);
title("slice – fatia, corte z=0, curva de nível por cor:")
 xlabel ("eixo - x");
 ylabel ("eixo - y");
 zlabel ("eixo - z");

#c) Curva de nível com cor, corte (xi,yi,zi):
[x, y, z] = meshgrid (linspace (-8, 8, 32));
w = sin (sqrt (x.^2 + y.^2 + z.^2)) ./ (sqrt (x.^2 + y.^2 + z.^2));

[xi, yi] = meshgrid (linspace (-7, 7));
zi = xi + yi;
slice (x, y, z,  w,  xi, yi, zi);

title("slice – corte pelo plano (xi,yi,zi):")
 xlabel ("eixo - x");
 ylabel ("eixo - y");
 zlabel ("eixo - z");
```

OBS.: temos uma semelhança nos cortes anteriores.

Ex.09 – contorno num intervalo

```
## contourf (x, y, z,  z1:z2)
# Plota curvas de nível de uma função 3D:
# entre para valores de z=[z1,z2]
clf;
[x, y, z] = peaks (50);
 plot3(x,y,z);
contourf (x, y, z, -7:9);
```





7 ENTRADAS E SAÍDAS

7.1 Saída do terminal

O Octave tem uma linha de comando, normalmente precedida por >. Para conhecermos o valor de pi, basta digitar na linha de comando e teclar <ENTER>.

a) ans

Toda vez que uma operação for realizada sem atribuição o Octave atribui o resultado à variável ans.

Exemplo:

```
> pi
ans = 3.1416

>  2+3
Ans = 5
```

b) disp(x) - display a variável x.

Display a variável x. Termina com nova-linha.

Exemplo:

```
> disp("O valor de pi: "), disp(pi)
O valor de pi:
 3.1416
```

c) fomat(op) – controla formatação de saída.

Valores que o parâmetro op pode assumir:

- short – saída curta.
- long – saída longa.
- hex – saída hexadecimal.
- bit – imprime o bit mais significativo primeiro.
- rat – transforma decimal em racional.

Se op é omitido, retorna ao formato default. O Octave tentara imprimir um número com pelo menos cinco algarismos significativos, com no máximo dez caracteres de comprimento. Se o Octave não consegue, passará a representar no formato exponencial “e”, de base 10..

Exemplo:

```
> format short, pi
ans =  3.1416
```

```
> format long, pi
ans =  3.14159265358979
> pi^64
ans = 6.57040064457169e+031   % formato exponencial, base 10

> format hex, 8
ans = 4020000000000000
> format short, ans
ans =  8

> format bit, 8
ans = 01000000001000000000000000000000000000000000000000000000000000000000000000000000
> format short, ans
ans =  8

> format rat, pi
ans = 355/113
> format long, ans
ans =  3.14159265358979

> format
> ans
ans =  3.1416
```

7.2 Arquivos básicos I/O (entrada e saída)

a) save data var1 [var2 , ...]

Salva os valores das variáveis var1, var2, ... no arquivo de nome data. O arquivo data tem propriedades para manter a estrutura das variáveis quando carregado.

b) load data var1 [var2 , ...]

Restaura os valores das variáveis previamente salvas no arquivo data. Podemos recuperar parcialmente algumas variáveis, nomeando cada uma delas ou, caso não sejam nomeadas, será recuperado todas as variáveis do arquivo de dados.

Os comandos save/load alocam dados para serem escritos e lidos no disco de armazenamento. Como exemplo disso, podemos salvar e carregar uma matriz de dados como segue:

Exemplo: Mostra o uso dos comandos save/load. Todas as linhas abaixo são de comandos, foram omitidas as saídas para maior clareza do texto.

```
who                #mostra as variaveis definidas
clear all   #deleta estas variaveis

#define duas matrizes a, b:
a=[1 2;3 4],  b=[4 3;2 1]

save arq_01.m a b  #salva a,b no arquivo arq_01.m
clear all         #deleta as variaveis a, b.
who         #lista as variaveis definidas...

load arq_01.m      #carrega arq_01.m
who          #a,b agora estão disponiveis
a,b
clear all     #limpa tudo...

load arq_01.m a      #carrega no arq_01.m, a varia'vel a
who          #a-disponiveis, b-indisponivel
a^2
```

7.3 Saídas convencionais

a) print

- print Serve para imprime gráficos. Usado para salvar arquivos com extensões específicas. Ver doc print.

b) fprint

- printf (TEMPLATE, ...) Imprime os argumentos sob o controle as string template, na saída `stdout', isto é, saída default, normalmente o vídeo.

c) fprintf

- fprintf (FID, TEMPLATE, ...) Esta função é a mesma que printf exceto que endereça a saída específica, FID e não à saída `stdout' .

Conversões de saída:

- '%d', '%i' - Imprime um inteiro com um simples número decimal.
- '%o' – Imprime um inteiro com um número octal
- '%x' - Imprime um inteiro com um número hexa-decinal.
- '%f' - Imprime um ponto-flutuante em ponto-flutuante.
- '%e' - Imprime um ponto-flutuante em notação exponencial.
- '%g' - Imprime um ponto-flutuante em ponto-flutuante ou exponencial, o que tiver tamanho mais conveniente.
- '%c' – Imprime um caráter.
- '%s' – Imprime uma string,
- '%%' – Imprime o caractere literal '%'.

O uso de conversões inválidas pode causar erros invisíveis (que pode facilmente causar engano!), o que deve ser evitado ao máximo.

Exemplo:

```
>  printf ("%4d, %4.2f %8.4e %8.4g\n", 1234, 34.123456, e^10, e^10);
1234, 34.12, 2.2026e+004, 2.203e+004

>  e^10, printf (" a=%5i,\n b=%4.2f,\n c=%8.4e,\n d=%8.4g\n", e^10, e^10,e^10, e^10)
ans =  22026.4657948067
 a=22026,
 b=22026.47,
 c=2.2026e+004,
 d=    1024

> printf ("Podemos escrever strings: %s \n",  "Octave");
Podemos escrever strings: Octave

> printf ("Caracteres acentuados e especiais: %c, %c \n", 128, 123)
Caracteres acentuados e especiais: Ç, {

> h=hilb(3), printf ("%4.2f %10.2e %8.4g\n", h);
h =
  1.00000   0.50000   0.33333
  0.50000   0.33333   0.25000
  0.33333   0.25000   0.20000

1.00  5.00e-001   0.3333
0.50  3.33e-001     0.25
0.33  2.50e-001      0.2

>  h=hilb(4), printf ("   %.5f   %.5f   %.5f   %.5f\n", h);
h =
  1.00000   0.50000   0.33333   0.25000
  0.50000   0.33333   0.25000   0.20000
  0.33333   0.25000   0.20000   0.16667
  0.25000   0.20000   0.16667   0.14286

  1.00000   0.50000   0.33333   0.25000
  0.50000   0.33333   0.25000   0.20000
  0.33333   0.25000   0.20000   0.16667
  0.25000   0.20000   0.16667   0.14286

> printf ("%2s%6s  \n", "no", "where");
no where
 > printf ("%2s%-6s  \n", "no", "where");  #junta
nowhere
```

Exemplo: listando caracteres acentuados

```
> for i=128:165 printf("%d ->  %c  \n",i,i) endfor

128 -> Ç
129 -> ü
130 -> é
131 -> â
132 -> ä
133 -> à
134 -> å
135 -> ç
136 -> ê
137 -> ë
138 -> è
139 -> ï
140 -> î
141 -> ì
142 -> Ä
143 -> Å
144 -> É
145 -> æ
146 -> Æ
147 -> ô
148 -> ö
149 -> ò
150 -> û
151 -> ù
152 -> ÿ
153 -> Ö
154 -> Ü
155 -> ¢
156 -> £
157 -> ¥
158 -> Pts
159 -> ƒ
160 -> á
161 -> í
162 -> ó
163 -> ú
164 -> ñ
165 -> Ñ

>printf("\n Isto %c um sufoco, mas %c poss%cvel: \n\n",130,130,161)
Isto é um sufoco, mas é possível:
> printf("%c\n%c f(x) dx = 1.2  \n",244,245)
⌠
⌡ f(x) dx = 1.2
```

8 EXEMPLOS E APLICAÇÕES

8.1 Polinômios o Octave

O ambiente Octave disponibiliza rotinas para manipulação de polinômios. Passamos a introduzir essas facilidades através do exemplo que segue.

Exemplo do uso de polinômios no Octave:

```
clc;
## a)Polinomio p - representado pelos seus coeficientes:
#==================================
# Se p = x^2 + 2x - 3
# sera representado por p=[1, 2, -3]
#==================================
p=[1, 2, -3]
# Polinomio na variavel x:
px=polyout(p,"x")

clc;
## b) polyval(p,x0) - valor numérico p(x0):
#=========================
# Vamos encontrar p(1):
#=========================
p=[1, 2, -3]
polyval(p,1)

clc;
## c) roots(p) - encontrando as raízes complexas de p:
#=============================================
# Vamos encontrar as raizes complexas de q=x^2+1:
#=============================================
q = [1, 0, 1]; #define q
qx=polyout(q,"x") #saida de q
roots(q)      #raizes de q


#Obs.:  z=0+1i , z=-0-1i são as raízes conjugadas de q.
# A notação z=-0 + 1i, significa valores quazi_zero,
# isto é, valores a partir dos quais, será considerado como zero.

clc;
## d) conv(p,q) - Produto de p por q:
# convolucao de p e q.
#============================
#Se p=x+1, q=x-1,
#vamos encontrar pq = x^2 -1,
#============================
```

```
p=[1  1]; q=[1 -1];
r=conv(p,q),polyout(r,"x")

clc;
## e) [q,r] = deconv(D,d)
# D - dividendo, d - divisor,
# q - quocinete, r - resto.
#=========================================
#Se D=x^2+x+1, d=x+1,
# vamos encontrar o quociente q=x, resto r=1:
#=========================================
D=[1 1 1];d=[1 1];
[q,r]=deconv(D,d)
D
conv(d,q)+r

clc;
## f) polyderiv(p) -  encontra a derivada de p:
p=[3 2 1]; polyout(p,"x")
polyderiv(p)

clc;
## g) polyint(p) - encontra a integral indefinida de p,
#   com a constante nula:
p=[3 2 1];
q=polyint(p)

clc;
## h) Máximo divisor comum: (gcd)
#=========================
# Se p=x^3-3x^2+2x-1
#    q=x^3-2x^2+x
# MDC(p,q) = x^2-2x+1
#=========================
 p=[3 2 1];  q=[1 -2 1 0];
 polygcd(p,q)

## i) A questão da aproximação nas raízes:
format long
 p=[1 -3 3 -1.001];  roots(p)
```

8.2 Ajuste Polinomial

Octave vem com um bom suporte para vários tipos de interpolação, a maioria dos quais estão descritos no tópico de Interpolação.

Uma alternativa para ajustar dados através de polinômios, evitando que estes sejam de graus excessivamente altos, normalmente é feito no sentido dos mínimos quadrados e, para isto usamos o comando polyfit:

[P, S, MU] = polyfit (X, Y, N)

- Retorna os coeficientes do polinômio P(X) de grau N que minimiza os erros de ajuste, no sentido dos mínimos quadrados. Os coeficientes do polinômio são retornados na forma de um vetor.

A segunda saída contem as seguintes informações:

- R – fator triangular da decomposição QR usada na solução interna.
- X – a matriz de Vandermonde usada no calculo dos coeficientes do polinômio.
- df - grau de liberdade.
- Normr – norma dos resíduos.
- yf - valor do polinômio para todo X.
- dyf – desvio associado a yf

```
#============================
#Ajustes polinomiais:
#Ex.02
#Graus: 1,2,3
#p=polyfit(x,y,n)
#29/03/2010
#============================
 x=-1.2:0.1:1.2;y=x.^3-x;    %ptos a serem ajustados

#a) Análise do grau: n=1
x=-1.2:0.1:1.2;y=x.^3-x;
 p=polyfit(x,y,1)    %Ajuste polinomial grau=1
 plot(x,y,"o",   x, polyval(p,x));
 title("Dados e p – grau 1:")

 #Conclusão:  não ajustou

#b) Análise do grau: n=2
x=-1.2:0.1:1.2;y=x.^3-x;
 p=polyfit(x,y,2)    %comando de ajuste polinomial
 plot(x,y,"o",   x, polyval(p,x));
 title("Dados e p – grau 2:")
 pause(3)
#Conclusão:  não ajustou bem
```

```
#c) Análise do grau: n=3
x=-1.2:0.1:1.2;y=x.^3-x;
 p=polyfit(x,y,3)    %comando de ajuste polinomial
 plot(x,y,"o",   x, polyval(p,x));
 title("Dados e p – grau 3:")
 pause(3)

 #Erro abs:
  plot(x,abs(y -  polyval(p,x)), "o");
  title("Erro Absoluto:")
  pause(3)

 #Erro relativo:
  plot(x,abs(y- polyval(p,x))/abs(y),"o");
  title("Erro Relativo:")
  pause(3)

 #Análise de coeficientes:
 #Em p – grau 3, p(2), p(4) são significativos?
 plot(x, p(1).*x.^3+p(2).*x.^2 +p(3).*x+p(4),
       x,  p(1).*x.^3+p(3).*x, "o");
 grid on ;
 title("p – grau3, Análise de Coeficientes.:")
 pause(3)

 #definindo ps
#ps - polinômio com coef significativos:
#ps=inline("p(1).*x.^3+p(3).*x")
ps=[p(1),0,p(3),0]
#Erro relativo do polinômio ps:

#ps(x) aproxima bem!!!
 plot(x,abs( (y-ps(x))/y ),"o" );
 title("Erro relativo:  ps – grau 3, com coeficiente significativos:")

# Conclusão final:
x=-1.2:0.1:1.2;y=x.^3-x;
plot(x,y,"o",   x, polyval(ps,x));
 title("FINAL: Dados e ps – grau 3:")
```

Exemplo: Ajuste com desvio padrão

```
##============================
##  AJUSTE COM DESVIO PADRÃO:
##============================
x=0:10;y=x.^2;   #gera uma base de dados
#y(2)=1.3;y(6)=27;y(9)=60;  # A) perturbação da base de dados
y(2)=1.1;y(6)=24;y(9)=65;  # B) perturbação da base de dados

##===============
# #Ajuste de grau=1:
##===============
[p,s] = polyfit(x,y,1);
xf = linspace(x(1),x(end),150);
[yf,dyf] = polyconf(p,xf,s,'ci'); # yf=p(x), dyf – desvio padrão
                                  # ci – intervalo confidencial,mais usado
#Grafico de yf, yf+dyf, yf-dyf
plot(xf, yf,       'b-;yf-poly;',
       xf, yf+dyf,'g.;dy+;',
       xf, yf-dyf, 'g.;dy-;',
       x,   y,        'r*;BD;');
title("Ajuste:  p1, dyf+, dyf-, BD")
pause(3)

#Grafico do erro=y-p, dfy, -dfy
plot(x,y-polyval(p,x),'r ;y-p1;',
      xf, dyf,              'g-;dyf+;',
      xf,-dyf,              'g-;dyf-;');
title("y-p1, dyf+, dyf-")
grid on
pause(3)

##===============
# #Ajuste de grau=2:
##===============
[p,s] = polyfit(x,y,2);
xf = linspace(x(1),x(end),150);
[yf,dyf] = polyconf(p,xf,s,'ci' );

#Grafico de yf, yf+dyf, yf-dyf
plot(xf, yf,       'b-;yf-poly;',
       xf, yf+dyf,'g.;dy+;',
       xf, yf-dyf, 'g.;dy-;',
       x,   y,        'r*;BD;');
title("Ajuste:  p2, dyf+, dyf-, BD")
pause(3)
```

```
#Grafico do erro=y-p, dfy, -dfy
plot(x,y-polyval(p,x),'r ;y-p2;',
     xf, dyf,          'g-;dyf+;',
     xf,-dyf,          'g-;dyf-;');
title(“y-p2, dyf+, dyf-”)
grid on
```

8.3 Curvas polinomiais por partes

Em situações em que um único polinômio é suficiente, uma solução é a utilização de polinômios por partes. A função “mkpp” cria um polinômio por partes, e “ppval” avalia a função criada por “mkpp” e “unmkpp” retorna informações detalhadas sobre a função.

O exemplo a seguir mostra como combinar duas funções lineares e uma quadrática em uma função. Cada uma dessas funções polinimiais estão definida em intervalos adjacente.

Exemplo: Curvas polinomiais por partes

```
##============================
##  AJUSTE COM DESVIO PADRÃO:
##============================

## Três polinômios:
x = [-2,-1, 1, 2];
    p = [0, -1, 1;  1, -2, 0;  0,1, 0];
    pp = mkpp (x, p);
    xi = linspace (x(1), x(end), 100);
    yi = ppval (pp, xi);
    plot (xi, yi);
grid on

## Três polinômios com controle de x:
x = [-2,-1, 1, 2];
    p = [0, -1, 1;  1, -2, 0;  0,1, 0];
    pp = mkpp (x, p);
    xi = linspace (x(1), x(end), 100);
    yi = ppval (pp, xi);
    plot (xi, yi);
grid on

## Quatro polinômios com controle de x:
x = [1,2,3,4,5,];
    p = [0,-1,1;0,1,0;0,-1,1;1,0,0];
    pp = mkpp (x, p);
    xi = linspace (x(1), x(end), 100);
    yi = ppval (pp, xi);
    plot (xi, yi);
grid on
```

- yi = ppval (pp, xi)
  Avalie polinômio por partes pp nos pontos xi.
  Note que xi é uma partição do intervalo x

- pp= mkpp (x,p)
  Construir uma estrutura dos polinômios por partes no intervalo x, respeitando sua partição.

A linha i de p, 'p (i ,:)', contém os coeficientes de um polinômio sobre o intervalo i, ordenados do maior para o menor grau. Deve haver uma curva para cada intervalo em x, assim, linhas (p) == |x| - 1.

Podemos concatenar vários polinômios de graus variados, porem dever ser escritos na mesma estrutura, ou seja, se n é o maior grau dos polinômios, a matriz p tem o número de colunas igual a n+1.

8.4 Implementação vetorial de Gauss-Seidel

Seja o sistema Ax=b, com a matriz A diagonal dominante.

a) Algoritmo:

$$x_i^{k+1} = (b_i - \sum_{\substack{j=1 \\ j \neq i}}^{n} a_{i,j} x_i^k)/a_{i,i}$$

Ou, próprio para vetorizar

$$x_i^{k+1} = (b_i - \sum_{j=1}^{n} a_{i,j} x_j^k + a_{i,i} x_i^k)/a_{i,i}$$

b) Convergência:

A convergência é garantida se a matriz A é diagonal dominante, isto é:

$$|a_{i,i}| > \sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{i,j}|, \text{ i=1,2,...,n.}$$

c) Implementação vetorizada de Gauss- Seidel:

```
function [y,erro,j]=sistema_gs(A,b, conv)
#USO: sistema_gs(A,b, conv), Ax=b, A – diagonal dominante, conv - erro
# Teixeira 2010
[n,n]=size(A);
y=rand(n,1);  #chute inicial
j=0;

do
 j++;
 if (j>1000) return; endif; #controle
 x=y;
 for i=1:n
   y(i)=[b(i)-A(i,:)*x + A(i,i)*x(i)]/A(i,i);
 endfor
until norm(y-x)<conv;
erro=norm(A*y-b);
endfunction;
```

d) Alguns exemplos:

A seguir dois exemplos gerados randomicamente, fortemente diagonais. O primeiro, uma simples matriz de ordem 5, já o segundo, uma matriz de ordem 1000. O destaque, que nestes casos, que a convergência ocorre para um número de iterações j=9, j=20, valores muito pequenos e, o no segundo caso, o tempo de resposta é menor que 3 segundos.

Exemplo 01:

```
##
##Um simples exemplo: n=5
##
n=5; A=rand(n,n)+diag(i=n+1:2*n), b=rand(n,1)
 [x,erro,j]=sistema_gs(A,b,0.0001)

n1=sqrt(sum(sumsq(A*x-b)))
n2=norm(A*x-b)
```

Exemplo 02:

```
## Uma matriz 1000x1000, fortemente diagonal dominante
##   com um erro da ordem de e-005,
##   com j=20 iterações,
##   em menos de 3 segundos.
##Intel Centronic Duo, 2GB, HP

t0=cputime;

 n=3000;
conv=0.0000001;
 A=rand(n,n)+diag(i=n+1:2*n);
 b=rand(n,1);
 [x,erro,j]=sistema_gs(A,b,conv);
 erro,j

t1=cputime; t1-t0

n1=sqrt(sum(sumsq(A*x-b)))
n2=norm(A*x-b)
```

8.5 Manipulação Simbólica: Pacote symbols

a) Pacote symbols - Manipulação Simbólica (sym)

O Toolbox do Octave para Manipulação Simbólica é baseado GiNaC. O objetivo simplesmente é dar as capacidades dos GiNaC e suas facillidades no ambiente Octave. Isso não torna o Octave um ambiente de Computação Algébrica, mas simplesmente introduz facilidades da CA.

b) Limitações e Recursos:

Atualmente não há suporte para matrizes simbólicas. A fim de fazer aritmética exata que você precisa para lidar com o comando vpa. Por exemplo: vpa ("1") / vpa ("7") é representado internamente exatamente como 07/01. Abaixo é segue uma lista de função implementadas:

* vpa - seta uma constante na forma string, como variável simbólica de precisão dupla.
* sym - cia uma variável simbólica.
* is_vpa - retorna verdadeiro se o objeto é vpa.
* is_sym - retorna verdadeiro se o argumento é uma var. sym.
* is_ex - retorna verdadeiro se o objeto é expresão sym. (i.e. x+y)

* digits(N) - seta o número de dígitos da vpa criada
* Abs - valoss absoluto

* Sqrt - Sqrt(x) => x^(vpa(1)/2) or x^(1/vpa(2))
* Cos, Sin, Tan, aCos, aSin, aTan, aTan2 - trigonomotria
* Cosh, Sinh, Tanh, aCosh, aSinh, aTanh - trigo hiperbólica
* Exp, Log - exponencial e logarítmica
* Zeta, Tgamma, Lgamma, Beta - funções especiais
* Factorial, Binomial - clássicas

* subs - substituição em uma expressão
* differentiate - derivada de uma expressão
* expand - expande uma expressão: (x+y)*(x+z) => x^2+x*y+x*z+y*z
* collect - evidencia termos semelhantes numa expressão.

* coeff - retorna um coeficiente específico de um polinômio.
* lcoeff - coef. do maior termo de polinômio (4x^2+2x+5 => 4)
* tcoeff - coef. do menor termo de polinômio (4x^2+2x+5 => 5)
* degree - grau do polinômio (i.e. x^2+2x+1 => 2)
* ldegree - menor grau dos termos de um polinômio (i.e. x^2+2x+1 => 0)
* quotient - quociente
* remainder - resto

* Gcd, Lcm - max. e mín. divisor comum de polinômios
* numer, denom - numerador e denominador de uma expre.

* Series - tratamento de séries
* Pi - variável pi
* splot - plota uma função sym

c) Exemplo 01 - Abordagem Geral em Symbols

Exemplo 01:

```
## ================================================
% demo: sym - Octave
% Teixeira
% 2010
## ================================================

clc;
##================================================
##  CARREGANDO O PACOTE SYMBOLS
##  DEFININDO VARIÁVEIS ALGÁBRICAS:  x, y, z
##================================================
 symbols
 x=sym("x")
 y=sym("y")
 z=sym("z")
whos x y z

clc;
##================================================
##  DEFININDO f(x,y) NÃO ALGEBRICA:
##  CONTROLANDO AS VARIÁVEIS:
##================================================
 f=x^2+3*x*z-y^2
 vars = findsym (f)
 vars1 = findsym (f,1)
 vars2 = findsym (f,2)
 vars3 = findsym (f,3)
```

```
clc;
##=====================================================================
##  splot (F,X,I)
##     PLOTA A FUNCÃO PSEUDO-SIMBÓLICA F, NA VARIÁVEL X,
##     NO INTERVALO  I=[a,b].
##=====================================================================

f=x^2-5*x+6
splot(f,x,0:5)
grid on

 title ("FUNÇÃO SIMBÓLICA:");
 xlabel ("x");
 ylabel ("y");
 legend ("f=x^2-5*x+6");


clc;close;
##=====================================================================
##   vpa            SETA UMA CONSTANTE COMO ALGÉBRICA.
##   is_vpa(T)    VERIFICA SE T É ALGÉBRICA
##   is_sym(X)   VERIFICA SE X É SIMBÓLICA
##   is_ex(F)      VERIFICA SE F É EXPRESSÃO
##   digits(N)     CONVERTE UM VPA PARA N-DÍGITOS
##=====================================================================
is_sym(x)
f=x^2+3*x*z-y^2, is_ex(f)
t=vpa("1/3"); is_vpa(t)  #parametro algebrico
digits(5); t, t/2
digits(30); t, t/2, t^100

clc;
##=====================================================================
##   da_dx = differentiate(F,X [, N])
##      RETORNA A N-ESIMA DERIVADA DE F EM RELACÃO À X.
##      N=1, default.
##   expand(EX):
##      EXPANDE A EXPRESSÃO EX
##=====================================================================
clc;
#EX.:01   f - SEMI-ALGEBRICA ============================
f=x^2+3*x*z-y^2
f1=differentiate(f,x,1)
f2=differentiate(f,x,2)
```

```
#EX.:02   f – ALGEBRICA ==========================
    f=x^vpa("2")+vpa("2")*x*y-y^vpa("3")
    f,fx=differentiate(f,x,1)
    f,fy=differentiate(f,y,1)
    fx,fxy=differentiate(fx,y,1)


 clc;
##=====================================================================
##   coeff(A,X,N)   RETORNA O N-ÉSIMO COEFICIENTE DE A, COMO POLINÔMIO EM X.
##   lcoeff(A,X)    COEFICIENTE DO TERMO PRINCIPAL (4x^2+2x => 4)
##   tcoeff(A,X)    COEFICIENTE DO MENOR TERMO. (4x^2+2x => 2)
##   degree(A,X)    RETORNA O GRAU A, COMO POLINÔMIO EM X.
##                  (degree(x^10+x*y^12+x,x) => 10)
##   subs(A,X,Y)    RETORNA B, CÓPIA DE A, ONDE X FOI SUBSTITUIIDO POR Y.
##                  A PERMANECE INAUTERADO.
##   splot(F,X,I0)  PLOTA F NA VARIÁVEL X NO INTERVALO I0
##                  PLOTA 2D - X DEVE SER ÚNICA!
##=====================================================================

#COEFICIENTES: ======================
p=x^vpa("10")-y^vpa("5")*x
lc=lcoeff(p,x),
tc=tcoeff(p,x)

#CONTROLE DA EXP: ===================
g=(x+1)^vpa("5")
g1=subs(g,x, 1)
gz=subs(g,x, z)

f=x^2+3*x*y-y^2;
y1 = subs (f,x,1)
y2 = subs (f,{x,y},{1,1/3})

#GRAFICO: ===========================
g=(x+1)^vpa("5")
splot(g,x,[-2, 0.1])
title("Gráfico: splot(g,x, [a,b])")
#grid on
```

```
clc;close;
##============================================================
##   expand(A) - EXPANDE UMA EXPRESSÃO ALGÉBRICA
##   degree(A,X) - GRAU A EXCRITO COMO UM POLINÔMIO EM X
##   coeff(A,X,N) - COEFICIENTE DE GRAU N DE A, NA VARIÁVEL X.
##   collect(A,X) - ESCREVE A COMO POLINÔMIO EM X
##============================================================
p1=(x+y+x*y)^vpa("3")
p2=expand(p1)
p3=collect(p2,x)

#RECONSTRUINDO O POLINOMIO PELOS SEUS COEFICIENTES:
 n= degree(p3,x);  n0=to_double(n)

 q=0;
 for j=0:n0
   q=q+coeff(p3,x,j)*x^j;
 endfor;

#VERIFICANDO O RESULTADO:
erro=p3-q
```

d) Exemplo 02 – Sistemas Numéricos em symbols

Os próximos exemplos mostram como resolver um sistema numérico, manipulando variáveis no ambiente sys.

Exemplo 02:

```
clc;close;
## ================================================
##  SYMBOLS – symfsolve()
## ================================================

#{
[ X,INF,MSG ] = symfsolve (...)
   RESOLVE UM CONJUNTO DE EQUACOES PSEUDO-ALGEBRICAS USANDO fsolve().
   VARIAVEIS LIVRES NÃO INICIALIZADAS, SERAO INICIALIZADAS COM VALOR  ZERO
#}

##================================================
# SISTEMAS NUMÉRICOS EM SYMBOLS:  Ex.02a
##================================================
clc;clear all;close
symbols, x=sym("x"), y=sym("y"), z=sym("z")
   f=x^2+y^2-1, g=y-x
     [r,INF,MSG] = symfsolve(f,g,[0.1,1])
     [s,INF,MSG] = symfsolve(f,g,[-1,-0.1])
     #symfsolve(f,g,[0 1]) #equivale: x=0,y=1

#plot:
t=0:0.01:1;
     figure(1); plot(t,t, t, sqrt(1-t.^2))
     figure(2); plot(-t,-t, -t, -sqrt(1-t.^2))
     pause(5);close;

#ezplot: plot implicito
ezplot(@(x,y)y-x,[-1,1])
hold on
ezplot(@(x,y)x.^2+y.^2-1)
plot(r(1),r(2),"*")
plot(s(1),s(2),"*")
title("symfsolve: y=x e x^2+y^2=1 ")
```

```
clc;close;   ##limpa tela e gráfico
##=============================================
# SISTEMAS NUMÉRICOS EM SYMBOLS:  Ex.02b
##=============================================
symbols, x=sym("x"), y=sym("y"), z=sym("z")
   f=x^2+y^2/4-1,
   g=x^2/4+y^2-1,

## Plotando as curvas:
##            Pesquisando ponto P, próximo da raiz,
##            como ponto de partida

t=-1:0.01:1;
hold on;
     plot(t, 2.*sqrt(1-t.^2))
     plot(t, -2.*sqrt(1-t.^2))

t=-2:0.01:2;
     plot(t, sqrt(1-t.^2/4))
     plot(t, -sqrt(1-t.^2/4))

##Encontrando as raízes:
     [raiz1,INF,MSG] = symfsolve(f,g,[0.8,0.8]) # ponto P(0.8, 0.8)
     [raiz2,INF,MSG] = symfsolve(f,g,[0.8,-0.8])
     [raiz3,INF,MSG] = symfsolve(f,g,[0.-8,0.8])
     [raiz4,INF,MSG] = symfsolve(f,g,[0.-8,-0.8])

     #a1 = symfsolve(f,g,[0 1]) #equivale: x=1,y=5

##Plotando as raízes:
##plot(x0,y0,"*") onde x0=raiz(1), y0=raiz(2)
     plot(raiz1(1),raiz1(2),"*")
     plot(raiz2(1),raiz2(2),"*")
     plot(raiz3(1),raiz3(2),"*")
     plot(raiz4(1),raiz4(2),"*")
#hold off   ##FECHANDO HOLD
```

```
clc;close;   ##limpa tela e gráfico
##=============================================
# SISTEMAS NUMÉRICOS EM SYMBOLS:  Ex.02c
##=============================================
symbols, x=sym("x"), y=sym("y"), z=sym("z")
   f=x^2+y^2-4,
     g=x*y-1,

## Plotando as curvas:
##            Pesquisando ponto P, próximo da raiz,
##            como ponto de partida

t=-2:0.01:2;
hold on;  # BOM FECHAR NO FINAL DO PROGRAMA
     plot(t, sqrt(4-t.^2))
     plot(t,-sqrt(4-t.^2))
t=0.2:0.005:2.5;
     plot(t, 1./t)
     plot(-t, -1./t)

##Encontrando as raízes:
     [raiz1,INF,MSG] = symfsolve(f,g,[2.0,0.5]) # ponto P(0.8, 0.8)
     [raiz2,INF,MSG] = symfsolve(f,g,[0.5,2.0])
     [raiz3,INF,MSG] = symfsolve(f,g,[-2.0,-0.5])
     [raiz4,INF,MSG] = symfsolve(f,g,[0.-2,-3.0])

     #a1 = symfsolve(f,g,[0 1]) #equivale: x=1,y=5

##Plotando as raízes:
##plot(x0,y0,"*") onde x0=raiz(1), y0=raiz(2)
     plot(raiz1(1),raiz1(2),"*")
     plot(raiz2(1),raiz2(2),"o")
     plot(raiz3(1),raiz3(2),"*")
     plot(raiz4(1),raiz4(2),"o")
hold
grid on
```

```
clc;close;   ##limpa tela e gráfico
##=============================================
# SISTEMAS NUMÉRICOS EM SYMBOLS:  Ex.02d
# OPS. DE USO
##=============================================

x=sym("x");   y=sym("y");
f=x^2+3*x-1;  g=x*y-y^2+3;

#Ponto inicial: P(1,5), Sol.: x = 0.30278   1.89004

# symfsolve: forma de uso 01
 x = symfsolve(f,g,x,1,y,5)

# symfsolve: forma de uso 02
 x = symfsolve(f,g,[1,5])

# symfsolve: forma de uso 03
[x,inf,msg]= symfsolve(f,g,[1,5])
```