SCILAB 5.X

Danusio Gadelha Filho

Laboratório de Vibrações

Universidade Federal do Ceará

SUMÁRIO

Laboratório de Vibrações

CAPÍTULO I

INTERFACE GRÁFICA DO SCILAB

Scilab é um ambiente voltado para o desenvolvimento de software para resolução de problemas numéricos. O Scilab foi criado em 1990 por um grupo de pesquisadores do INRIA – Institut de Recherche en Informatique et en Automatique e do ENPC - Ècole Nationale des Ponts et Chaussées.

Desde 1994, quando passou a ser disponível na Internet, Scilab é gratuito, free software, e distribuído com o código fonte, open source software. Além da distribuição com o código fonte, existem, também, distribuições pré-compiladas do Scilab para vários sistemas operacionais.

As principais características desse ambiente de programação numérica extremamente flexível são:

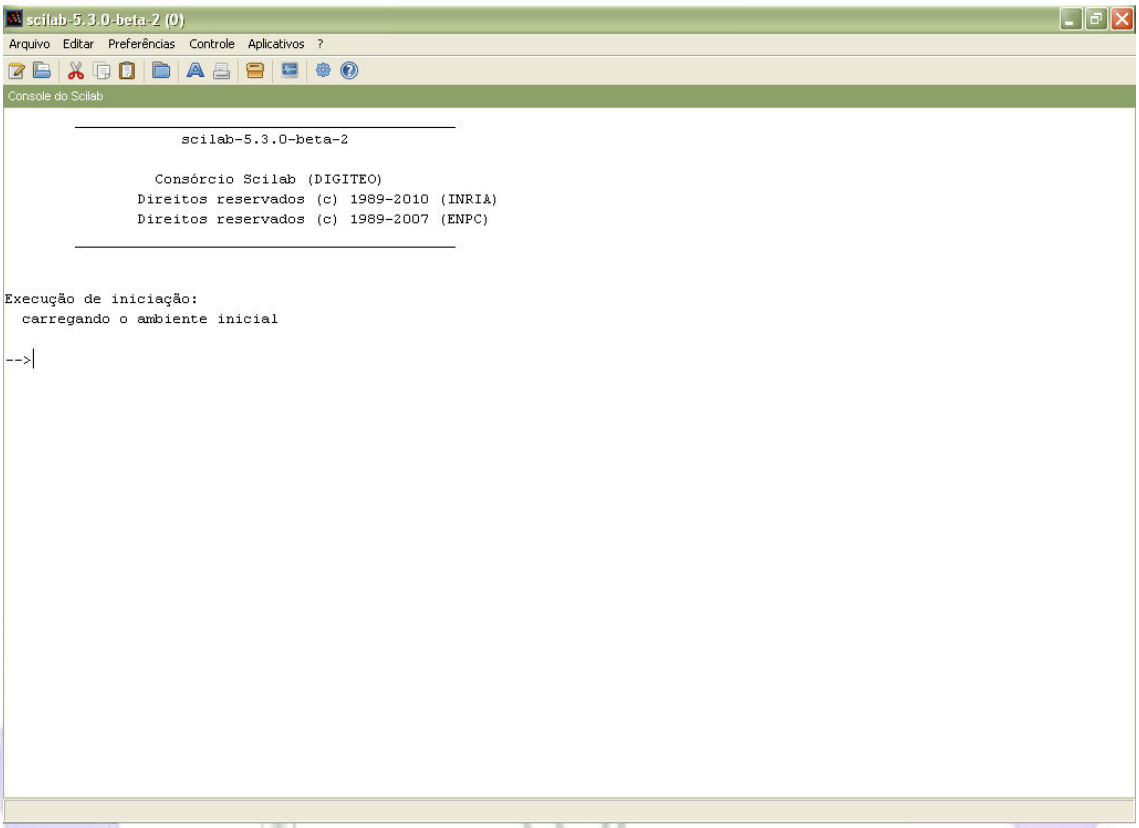
- Ambiente poderoso para geração de gráficos bi e tridimensionais, inclusive com animações;
- Manipulações com matrizes são facilitadas por diversas funções implementadas nos toolboxes;
- Permite trabalhar com polinômios, funções de transferência, sistemas lineares e grafos;
- Define funções facilmente;
- Permite acesso a rotinas escritas em FORTRAN e C;
- Pode ser acessado por programas de computação simbólica, como o MuPad;
- Permite o desenvolvimento de toolboxes.

Além dos toolboxes desenvolvidos pelo grupo Scilab, estão disponíveis outras complementares, igualmente gratuitas, como o ANN (*Artificial Neural Network*), o FISLAB (*Fuzzy Logic Inference*) e o FRACLAB (*Fractal, Multifractal and Wavelet Analisys*).

Algumas funções do Scilab estão alocadas em toolboxes bem – definidas, dadas as suas especificidades. Temos como exemplo:

- Funções de Álgebra Linear: bibliotecas LINPACK, EISPACK, LAPACK e BLAS;
- Funções para solução de Equações Diferenciais: bibliotecas ODEPACK e SLATEC;
- Funções de Otimização: biblioteca MINPACK.

Descritas algumas características internas do Scilab, é hora de conhecer a janela de trabalho (*Workspace*) do Scilab. Na versão 5.3.0 – beta 2, ela se apresenta da seguinte forma:



Podemos observar o *prompt* inicial, representado pelo símbolo -->, que faz a marcação da linha de comando e o cursor, ao lado do *prompt*. Nesse espaço, podem ser realizados cálculos e compilados programas sem serem salvas alterações. É através dele também que pode ser acessado o *help* do programa, através do comando **help** digitado no *prompt*. Será aberta uma janela depois de o comando ser executado (apertando – se a tecla ENTER), contento uma lista de todas as funções presentes.

Ainda na tela inicial estão presentes os menus *drop down* com sete opções: **File, Edit, Preferences, Control, Editor, Applications** e **?**. Cada menu possui os seguintes sub – menus:

File: New Scilab, Exec, Open, Load, Save, Change Directory, Get Current Directory, Print Setup, Print e Exit.

Edit: Select All, Copy, Paste, Empty Clipboard e History.

Preferences: Language, Colors, Toolbar, Files Association, Choose Fonts, Clear History, Clear Command Window e Console.

Control: Resume, Abort e Interrupt.

Editor

Applications: Scicos, Edit Graph, m2sci e Browser Variables.

?: Scilab Help, Configure, Scilab Demos, Web Links e About.

Conhecidos os menus da tela inicial, vamos analisar algumas peculiaridades da linguagem. O primeiro tópico a ser abordado é sobre variáveis especiais presentes no ambiente. Esses são valores pré – definidos que podem ser usados diretamente na programação. A maioria dessas variáveis é prefixada com o símbolo de porcentagem (**%**). Elas podem ser acessadas através da digitação do comando **whos** no prompt de comando; é importante lembrar que, se o usuário definir alguma variável antes de digitar **whos**, ela também aparecerá no workspace. Na tabela a seguir, temos a listagem de algumas delas, tal como aparecem na janela.

VARIÁVEL	TIPO
%F	Booleana
%T	Booleana
%z	Polinomial
%s	Polinomial
%nan	Constante
%inf	Constante
COMPILER	String
%gtk	Booleana
%gui	Booleana
%pvm	Booleana
%tk	Booleana
$	Polinomial
%t	Booleana
%f	Booleana
%eps	Constante
%io	Constante
%i	Constante
%e	Constante
%pi	Constante

Tabela 1 - Variáveis Especiais

Temos mais duas importantes variáveis: as *variáveis de localização* de diretórios SCI e PWD. Para entendê – las, digite – as no prompt de comandos.

SCI: diretório onde o Scilab foi instalado.

PWD: diretório onde o Scilab foi lançado, isto é, de onde seu script está rodando.

Existem ainda funções para manipular arquivos e diretórios. A função **pwd** mostra qual o diretório está sendo usado e **chdir** muda o diretório de trabalho. É importante lembrar que, depois de usada **chdir**, o valor de **pwd** muda, mas **PWD** permanece inalterado.

Todas as variáveis criadas durante os trabalhos no ambiente podem ser armazenadas em um arquivo. O comando **save** é usado para tal, com a seguinte sintaxe:

save('nome_do_arquivo.dat',variáveis)

Para recuperar os valores das variáveis, usa – se o comando **load** e o comando **clear** é usado para limpar variáveis não – protegidas:

load('nome_do_arquivo','variáveis')

O comando **help**, quando digitado sem caracterização, abre uma listagem com todas as funções presentes no programa. Se caracterizado, abre a mesma janela, porém com a função já aparecendo diretamente.

Além de armazenar variáveis, é possível criar uma memória de cálculo, salvando os comandos digitados em um arquivo, através do comando **diary**:

diary("nome_do_arquivo");

diary(0) // Fecha o comando

CAPÍTULO II

OPERAÇÕES PRIMÁRIAS

Scilab é um ambiente de programação numérica. Desse modo, existem duas formas de interação com o software: digitação diretamente no prompt, em que se tem uso de uma poderosíssima máquina de cálculo, ou como programação numérica propriamente dita, em que se delineiam linhas de código. Neste capítulo será abordada a primeira forma.

A primeira instrução a respeito do prompt é quanto ao uso do ponto – e – vírgula: ele faz com que o compilador interprete o fim da execução da linha de comando e esteja pronto para receber outra linha. Neste caso, o resultado da operação fica mascarado para o usuário. Se não for escrito o ponto – e – vírgula, a quebra de linha vai denotar fim da execução, mas o resultado será exibido para o usuário. Ambas as formas têm sua aplicação. Quando se está trabalhando com vetores de muitos elementos, não é conveniente esquecer o ponto e vírgula, pois isto pode acarretar perda de tempo e travamento da máquina (o Scilab tem uma ferramento útil para evitar esse infortúnio: a cada algumas linhas de exibição, aparece uma mensagem perguntando se se deseja continuar a exibição); já quando se quer testar uma operação, debugar um programa ou se as variáveis usadas forem de baixa ordem, o ponto – e – vírgula é por vezes necessário (no mínimo, facultativo).

Outra observação importante é que o Scilab é *case sensitive*. Quando o programador definir uma variável com letras maiúsculas e minúsculas, deve – se lembrar de manter a caixa das letras.

OPERAÇÕES MATEMÁTICAS – De um modo geral, as variáveis matemáticas usadas no Scilab são vetores complexos. O número imaginário é denotado por **%i** e, seguindo a notação padrão de um número complexo $z = a+bi$, temos as seguintes operações possíveis:

SOMA – "+"
SUBTRAÇÃO – "–"
MULTIPLICAÇÃO – "*"
DIVISÃO À DIREITA – "/"
DIVISÃO À ESQUERDA – "\"
POTENCIAÇÃO – "^"

ORGANIZAÇÃO DAS LINHAS DE COMANDO – A forma usual de se escreverem os comando é pôr uma instrução por linha. Também é permitido dispor as instruções numa mesma linha, desde que separadas pelo ponto – e – vírgula, e distribuir o comando em várias linhas, usando as reticências no final de cada linha (com exceção da última) e no começo da linha seguinte (com exceção da primeira).

AMBIENTE – Um ambiente é como um espaço de trabalho no prompt de comando do Scilab. Pode – se trabalhar com mais de um espaço de trabalho e é indicada esta prática caso sejam necessárias operações paralelas durante a execução de um programa principal. Para adicionar – se um ambiente, basta digitar ***pause*** no workspace ou usar o atalho ***ctrl – c*** no teclado; uma nova seta, acompanhada do cardinal

correspondente à hierarquia do ambiente, irá surgir. O retorno ao ambiente principal é dado pelo comando ***abort*** e ao ambiente anterior, pelos comandos ***return*** ou ***resume***.

As variáveis definidas num ambiente inferior não são salvas quando se sobe para um ambiente anterior. Caso se queira utilizar algum conteúdo de ambientes inferiores, deve – se usar a linha ***a=resume(a)***, onde ***a*** é a variável que se deseja salvar.

Se tratando de um software dedicado à computação numérica em tecnologia, o Scilab traz carregadas algumas funções utilizadas nessa área, tais como ***FFT, fatoração LU, determinante de matrizes, funções de Bessel***, dentre várias outras.

A seguir, temos uma tabela contendo várias funções básicas existentes no Scilab:

abs	abs(x)	módulo (argumento) de x
acos	acos(x)	arco – cosseno de x, em radianos
acosh	acosh(x)	arco – cosseno hiperbólico de x, em radianos
asin	asin(x)	arco – seno de x, em radianos
asinh	asinh(x)	arco – seno hiperbólico de x, em radianos
atan	atan(x)	arco – tangente de x, em radianos
atanh	atanh(x)	arco – tangente hiperbólico de x, em radianos
binomial	binomial (p,n)	binomial normalizado de p classe n*
ceil	ceil(x)	arredondamento para o maior inteiro posterior
cos	cos(x)	cosseno de x, com x em radianos
cosh	cosh(x)	cosseno hiperbólico de x, com x em radianos
cotg	cotg(x)	cotangente de x, com x em radianos
coth	coth(x)	cotangente hiperbólica de x, com x em radianos
cumprod	cumprod(v)	produto cumulativo do vetor v
cumsum	cumsum(v)	soma cumulativa do vetor v
diff	diff(v)	diferencial discreta do vetor v
eval	eval(s)	valor numérico de uma expressão de strings**
exp	exp(x)	exponencial euleriana de x
factor	factor(N)	fatores primos de N
find	find(k ‘s’ V)	encontra os índices dos elementos de V que tornam a comparação verdadeira (‘s’ → ==,<,>,~=,...)
fix	fix(x)	arredondamento para o inteiro mais próximo de zero
floor	floor(x)	arredondamento para o maior inteiro anterior
gsort	gsort(V,’r/c’,’i/d’)	ordenação dos elementos de V
imag	imag(z)	coeficiente da parte imagnária de z
int	int(N)	parte inteira de N
linspace	linspace(A,B,N)	vetor com N pontos entre A e B inclusos
log	log(x)	logarítmo neperiano de x
log10	log10(x)	logarítmo decimal de x
log2	log2(x)	logarítmo na base 2 de x
logspace	logspace (A,B,N)	vetor de N pontos com espaço logarítmico entre A e B inclusos
max	max(v)	máximo valor contido no vetor v
min	min(v)	mínimo valor contido no vetor v
modulo	modulo(m,n)	resto da divisão de m por n : **i = n - m .* int (n ./ m)**
ndims	ndims(M)	número de dimensões de M
pmodulo	pmodulo(m,n)	resto da divisão de m por n : **i = n - m .* floor (n ./ m)**
nextpow2	nextpow2(x)	potência de 2 maior e mais próxima de x
norm	norm(A)	norma de A
perms	perms(v)	todas as permutações do vetor v

primes	primes(N)	todos os números primos até N
prod	prod(v)	produto de todos os elementos de v
rand	rand(m,n)	matriz m x n de números aleatórios entre 0 e 1
real	real(z)	parte real do número complexo z
round	round(N)	número inteiro mais próximo de N
setdiff	setdiff(a,b)	retorna os elementos de ***a*** que não estão em ***b***
sign	sign(v)	retorna os sinais dos elementos de v: 1, se positivo, e -1.
sin	sin(x)	seno de x, x em radianos
sinh	sinh(x)	seno hiperbólico de x, x em radianos
sqrt	sqrt(x)	raiz quadrada de x
ssprint	ssprint(expr)	modifica a forma de exibição de um sistema linear
sum	sum(v)	soma de todos os elementos de v
tan	tan(x)	tangente de x, x em radianos
tanh	tanh(x)	tangente hiperbólica de x, x em radianos
tril	tril(M)	triangularização inferior de M
triu	triu(M)	triangularização superior de M
vectorfind	vectorfind(a,b,'s')	encontra o vetor ***a*** na matriz ***b, s=r*** ou ***s=c***

* Os termos estão localizados entre 0 e 1
** Um exemplo de uso:

```
-->a=1;
-->b=2;
-->z='2*a*b-b^2';
-->eval(z)
 ans  =
   0.
```

O comando ***solve*** soluciona um sistema linear simbólico, através da definição dos coeficientes das variáveis e dos resultados de cada equação, em formato de string.

```
-->A=['c1','c2';'c3','c4'];
-->b=['b1';'b2'];
-->solve(A,b)
 ans  =

!c1\(b1-c2*(c4\b2))  !
!c4\b2               !
```

O comando ***eval*** pode ser usado para avaliar o resultado para valores de todas as strings.

Dois tipos de função são imprescindíveis para a programação: os comandos de entrada e saída de dados. No Scilab, quando o programador quer que o usuário atribua um valor para uma variável, aquele faz o uso da função ***input***. A variável é atribuída da seguinte forma:

```
x=input("Expressão");
```

Uma forma mais elegante de pedir ao usuário que entre com dados necessários à execução do programa é usando caixas de diálogo, através da função ***x_dialog***, na seguinte sintaxe:

Laboratório de Vibrações

x=evstr(x_dialog('Expressão','Máscara'));

A função ***evstr*** é usada para converter os dados digitados pelo usuário (que são da forma *strings*) para um formato numérico. A máscara é um indicativo da forma dos dados que se querem obter. Por exemplo, se se deseja uma matriz tringular superior de terceira ordem, pode – se proceder da seguinte forma:

M=evstr(x_dialog('Digite a matriz U',['a b c','0 d e','0 0 f']));

Há outras caixas de diálogo mais específicas presentes no ambiente. A sintaxe de cada uma pode ser observada a seguir:

>> x_choices

```
l1=list('escolha 1',1,['botão a1','botão a2','botão a3']);
l2=list('escolha 2',2,['botão b1','botão b2','botão b3', 'botão b4']);
l3=list('escolha 3',3,['botão c1','botão c2']);
rep=x_choices('Menu de Escolhas',list(l1,l2,l3));
```

>> x_choose

```
n=x_choose(['item1';'item2';'item3'],['Escolha sua opção';'para a enquete']);
```

>> x_matrix

```
m=evstr(x_matrix('entre com uma matriz 3x3',zeros(3,3)));
```

>> **messagebox**

```
x=0.57;
messagebox('O valor de x é: '+string(x));
```

>> **uigetfile**

```
arquivo=xgetfile([title='string'])
```

O comando de saída tradicional é ***disp()***, que retorna o conteúdo entre parênteses. Uma forma mais elegante é feita com o uso de ***mtlb_fprintf*** ou ***printf***, que combina texto de exibição e variáveis. A seguir, exemplos de aplicação das funções:

-->printf('Meu nome é: %s','Danusio')
Meu nome é: Danusio

-->printf('O número é: %f',435.7666)
O número é: 435.766600

-->disp('Eu sou eu mesmo')

Eu sou eu mesmo

Laboratório de Vibrações

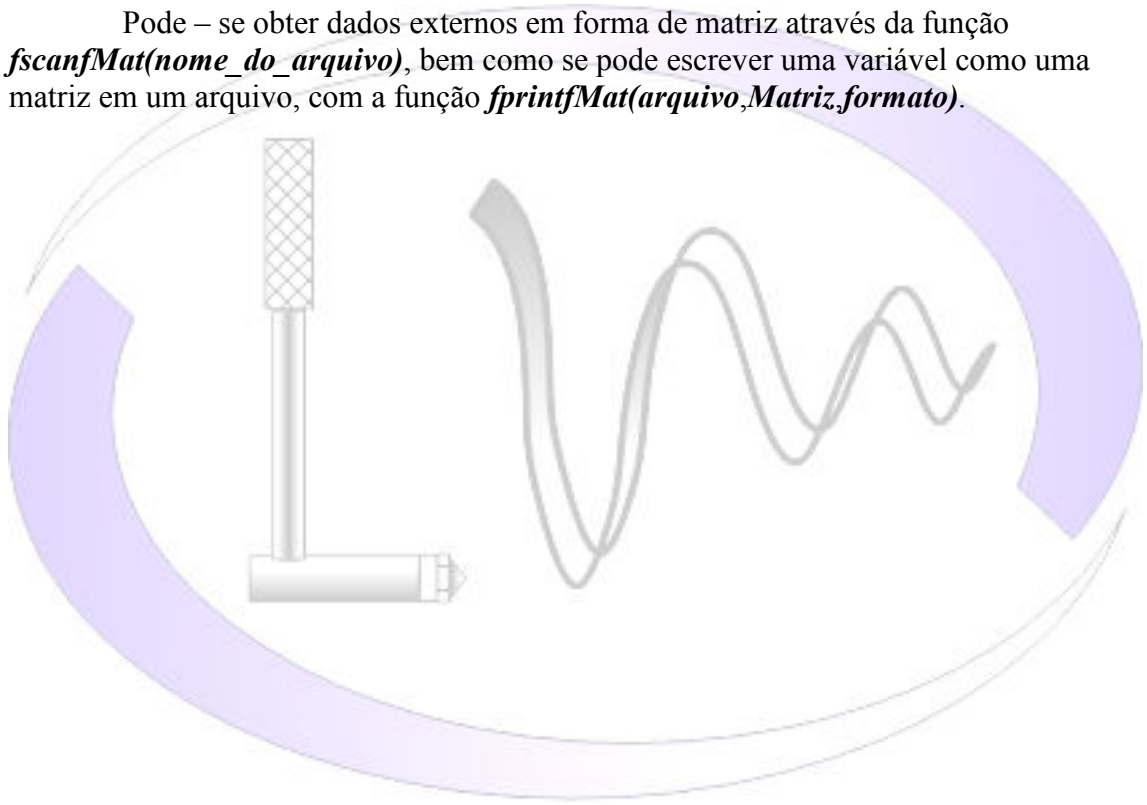
-->disp(14)

14.

-->disp(' ')

-->

Pode – se obter dados externos em forma de matriz através da função ***fscanfMat(nome_do_arquivo)***, bem como se pode escrever uma variável como uma matriz em um arquivo, com a função ***fprintfMat(arquivo*,*Matriz*,*formato)***.



Laboratório de Vibrações

CAPÍTULO III

COMANDOS DE FLUXO

Os laços são, de longe, a parte mais importante para se programar com o Scilab. Os laços reconhecidos pelo ambiente são: ***for, if,*** e ***while***.

O laço ***for*** segue a seguinte lógica: *para* um determinado índice variando de um *valor inicial* até um *valor final* com um determinado *incremento*, execute os comando sob o laço. Como exemplo, considere o seguinte exemplo, que carrega para uma matriz *M* os números pares de 1 a 100:

```
clc
clear
close

N=100;
M=[];

for i=1:2:(N-1)
  M=[M (i+1)];
end

disp(M)
```

Note que *N* é o número final do intervalo a ser varrido e que a variável *i* é, na verdade, entendido como um vetor.

O laço ***if*** funciona com a seguinte lógica: caso a *condição* seja verdadeira, execute os comandos. Considere o exemplo, para separar os números pares dos ímpares de 1 a 100:

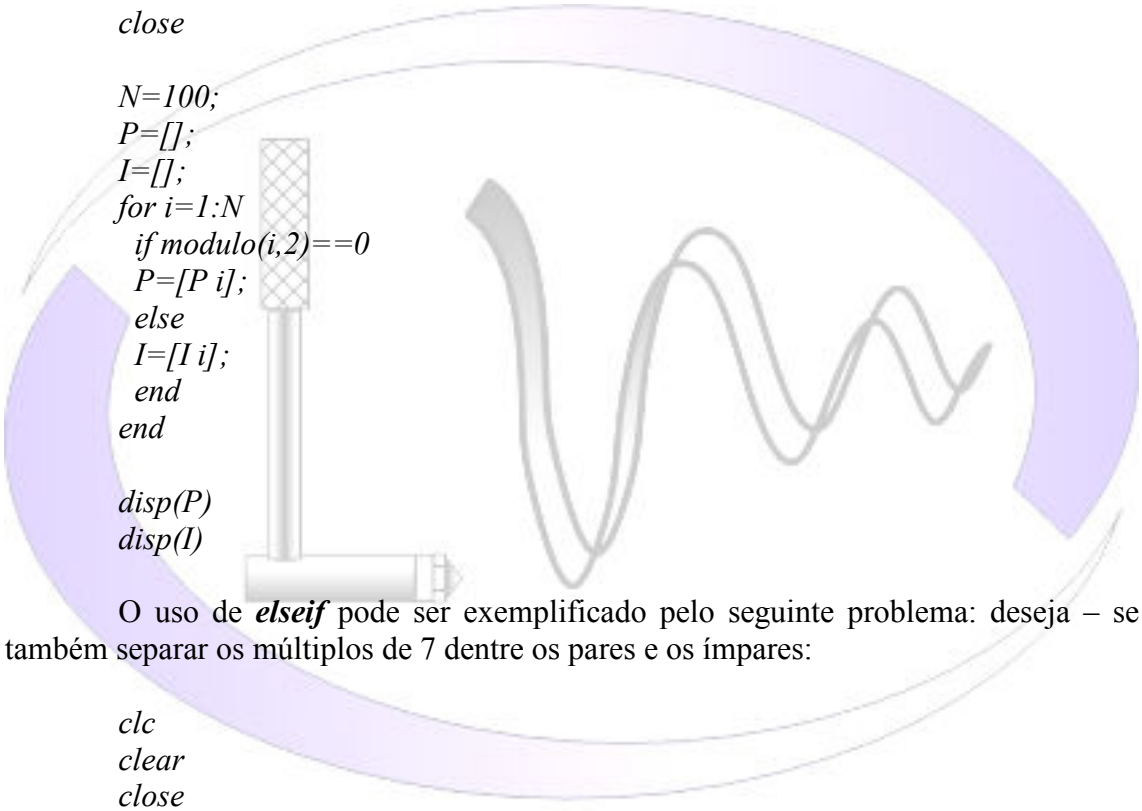
```
clc
clear
close

N=100;
P=[];
I=[];
for i=1:N
  if modulo(i,2)==0
  P=[P i];
  end
  if modulo(i,2)==1
  I=[I i];
  end
end

disp(P)
```

Laboratório de Vibrações

```
disp(I)
```

Nesse caso, a condição testada é o valor retornado pela função ***modulo*** (q.v.). A esse laço, podem ser agregados dois outros comandos: ***elseif*** e ***if***. Estes dois tornam a programação mais eficiente, elegante e "imune" a erros. O programa anterior, se utilizada a função ***else***, ficaria:

```
clc
clear
close

N=100;
P=[];
I=[];
for i=1:N
  if modulo(i,2)==0
  P=[P i];
  else
  I=[I i];
  end
end

disp(P)
disp(I)
```



O uso de ***elseif*** pode ser exemplificado pelo seguinte problema: deseja – se também separar os múltiplos de 7 dentre os pares e os ímpares:

```
clc
clear
close

N=100;
S=[];
P=[];
I=[];
for i=1:N
  if modulo(i,7)==0
  S=[S i];
  elseif modulo(i,2)==0
  P=[P i];
  else
  I=[I i];
  end
end

 disp(S)
disp(P)
disp(I)
```

Laboratório de Vibrações

É importante observar que a mudança da ordem de checagem das condições influencia o resultado. Por exemplo, em vez de testar primeiro se o número é múltiplo de 7, teste se ele é ímpar. O resultado é, na matriz S, os múltiplos pares de 7.

Os operadores lógicos mais comuns ao se usar a estrutura ***if*** são "e" (***&***) e "ou" (| ***- barra vertical***). Estes operadores são postos entre duas condições.

O comando ***while*** funciona como um misto de ***for*** e ***if***, na medida em que são executados comandos *até que* uma *condição* não seja mais satisfeita. Funciona como se os laços citados estivessem alinhados, porém sob a condição de haver um único ***if*** booleano. Um processo iterativo é um bom exemplo do uso deste recurso:

```
clc
clear
close

s=5;
N=0;

while s>1e-5
 s=s/10;
 N=N+1;
end
```

O contador *N* indica o número de operações realizadas até que o valor *s* seja menor que 1e-5.

A seguir, temos alguns códigos interessantes em que há os laços apresentados:

Código 1

```
clc
clear
close

disp('      Programa para carregar valores de uma função de duas variáveis para domínios de tamanhos diferentes')
disp(' ')
disp('                                  Danusio Gadelha Filho')
disp(' ')

x=1:1:5;
y=1:2:8;

disp(' Tamanho de x (')
disp(' ')
disp(length(x))
disp(' ')
disp(' Tamanho de y (')
disp(' ')
disp(length(y))
disp(' ')
```

Laboratório de Vibrações

```
disp(' Função de (x,y) (')
disp(' ')
disp(' f= (x²+y³)/(x+y) ')
disp(' ')

for i=1:length(x)
  for j=1:length(y)
    f(i,j)=(x(i)^2+y(j)^3)/(x(i)+y(j));
  end
end

disp(' Valores de f (')
disp(' ')
disp(f)

surf(f) //veja capítulo sobre gráficos
```

Código 2

```
clc
clear
close

disp('                Programa para simular a transcrição do RNA mensageiro')
disp(' ')
disp('                                Danusio Gadelha Filho')
disp(' ')


DNA=['A' 'G' 'C' 'A' 'G' 'T' 'H' 'C' 'C'];

disp(' DNA a ser transportado -->')
disp(' ')
disp(DNA)
disp(' ')

c=length(DNA);

for i=1:sum(c)
  if DNA(i)=='A'
    RNA(i)='T';
  elseif DNA(i)=='T'
    RNA(i)='A';
  elseif DNA(i)=='G'
    RNA(i)='C';
  elseif DNA(i)=='C'
    RNA(i)='G';
  else
```

Laboratório de Vibrações

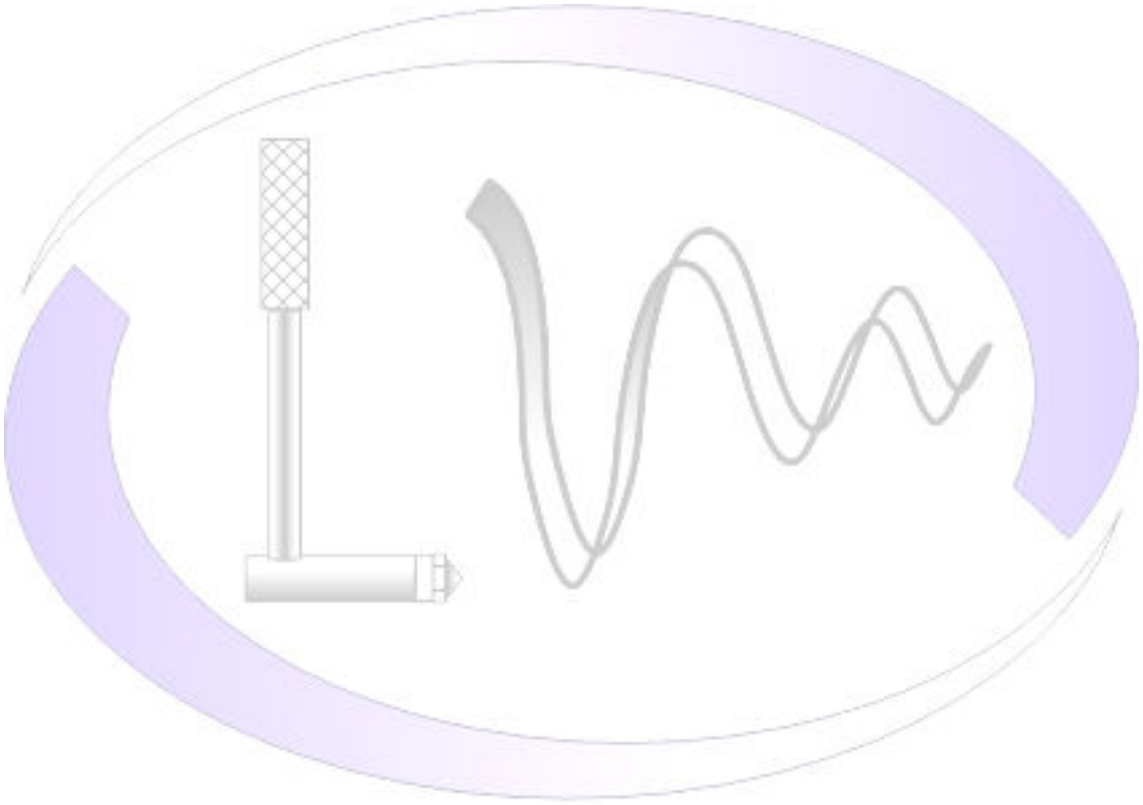
```
    continue
  end
end

disp(' Codificação do RNA mensageiro -->')
disp(' ')
disp(RNA')
disp(' ')
```

Código 3

```
clc
clear
close

disp('                Programa para encontrar a raiz de uma função em um
intervalo')
disp(' ')
disp('                        Danusio Gadelha Filho')
disp(' ')


prec=.0001;
X=0:prec:3;
N=length(X);
cont=1;
x=X(1);
err=x^3-2*x^2-5;

disp(' Intervalo -->')
disp(' ')
disp([string(X(1)),' --> ',string(X($))])
disp(' ')
disp(' Precisão -->')
disp(' ')
disp(prec)
disp(' ')
disp(' Função -->')
disp(' ')
disp(' x³-2x²-5')
disp(' ')

while abs(x^3-2*x^2-5) >= 10*prec & cont<N-1
  x=X(cont+1)^3-2*X(cont+1)^2-5;
  cont=cont+1;
  if abs(x)<abs(err)
    err=x;
    raiz=X(cont);
  end
end
```

Laboratório de Vibrações

```
disp(' Raiz(es) no intervalo -->')
disp(' ')
disp(raiz)
```



Laboratório
de Vibrações

CAPÍTULO IV

VETORES E MATRIZES

Vetores são um agrupamento de elementos em uma única fila (linha ou coluna). No Scilab, eles são sempre indicados entre colchetes; o ponto – e – vírgula denota mudança de linha, enquanto elementos de ma mesma linha são separados por vírgulas ou espaços.

A transformação de um vetor – linha em um vetor – coluna, ou vice – versa, chamada de transposição, é feita através do acréscimo de um apóstrofo ao lado direito do colchete de fechamento ou da variável que identifica o vetor.

Dois vetores estão sujeitos às seguintes operações entre si:

SOMA: **v1+v2**, desde que sejam da mesma dimensão.

SUBTRAÇÃO: **v1-v2**, desde que sejam da mesma dimensão.

MULTIPLICAÇÃO ESTRUTURAL: **v1.*v2**, desde que sejam da mesma dimensão.

DIVISÃO À DIREITA ESTRUTURAL: **v1./v2**, desde que sejam da mesma dimensão.

DIVISÃO À ESQUERDA ESTRUTURAL: **v1.\v2**, desde que sejam da mesma dimensão.

MULTIPLICAÇÃO: **v1*v2,** desde que o número de colunas de **v1** seja igual ao número de linhas de **v2**.

DIVISÃO À DIREITA: **v1/v2**, desde que sejam da mesma dimensão.

DIVISÃO À ESQUERDA: **v1\v2**, desde que sejam da mesma dimensão.

Matrizes são conjuntos de elementos dispostos em múltiplas filas. Todas as operações definidas para vetores podem ser usadas com matrizes, desde que a compatibilidade entre as dimensões seja observada.

Existem algumas funções úteis no tratamento com matrizes, tanto como recurso de programação quanto para debugar a rotina.

As funções ***size*** e ***length*** retornam, respectivamente, a ordem e a maior dimensão de uma matriz.

A função ***rank*** retorna o rank da matriz (rank é o número de filas linearmente independentes). A função ***norm*** retorna a norma de uma matriz.

Com o comando ***trace*** podemos obter a soma dos elementos da diagonal principal da matriz (ou *traço* da matriz). A diagonal principal pode ser extraída com o comando ***diag(M,n)***, onde *n* é a localização da matriz principal (de baixo para cima, variando de *-L* a *L*, onde *L* é a maior dimensão de *M* subtraída de um).

Uma aplicação interessante do uso das matrizes é na resolução de sistemas lineares. Dado um sistema linear no formato matricial ***Ax=b***, as variáveis contidas em ***x*** podem ser obtidas simplesmente por ***A\b***.

A priori, todas as variáveis para o Scilab são matrizes; deste modo, todas as operações acima descritas são válidas para qualquer variável definida no programa.

Definida uma matriz *M*, o elemento da linha *i* e coluna *j* é acessado através da digitação de ***M(i,j)***. O caractere especial *$* faz com q o último elemento seja acessado: ***M($)***. Quando se quer varrer toda uma sequência de filas, usam – se os dois pontos

entre o número das filas. Por exemplo, se a ordem de *M* é 1 x 15, as colunas de 7 a 11 poderiam ser acessadas pelo comando *M(7:11)*. Se se deseja ler toda uma sequência de filas, põem – se apenas os dois pontos, sem número algum: *M(:,:)*. Caso seja digitado apenas *M(:)*, o resultado será um vetor - coluna contendo todos os elementos de *M*.

Outra forma de selecionar elementos é substituir os elementos desejados por %T (verdadeiro) e os não selecionados por %F (falso). Por exemplo, se *M=[1 2 3 4 5]*, o comando *M([%T %F %F %T %F])* exibirá apenas o "2" e o "4".

Os elementos de uma fila podem ser organizados em outra matriz através do comando *M([n1 n2 ... nZ], c)*, caso se queira selecionar os elementos de ordem *n1, n2* até *nZ* da coluna *c*, ou *M(l, [n1 n2 ... nZ])*, caso a operação deva ser aplicada na linha *l*.

Existem matrizes especiais já implementadas dentro do ambiente do Scilab, tais como a matriz identidade, a matriz nula e a matriz de uns, criadas, respectivamente, por ***eye***, ***zeros*** e ***ones***, seguidas pela ordem.

A função ***bdiag(M)*** faz a diagonalização da matriz. A matriz inversa pode ser calculada pelo comando ***inv(M)***. Uma alternativa à resolução de sistemas lineares é a função ***linsolve(A,b)***, onde *A* e *b* são as matrizes de coeficientes e resultados, respectivamente. A linha de comando ***[L U] = lu(M)*** faz a fatoração LU da matriz *M*. Outros dois tipos de fatoração presentes são a de Choleski e a fatoração QR, chamadas pelos comandos ***chol(M)*** e ***qr(M)***. O comando ***orth(M)*** retorna a base ortogonal de um espaço vetorial definido pela matriz *M*. A forma sintática ***[r,a]=polar(M)*** transforma a matriz *M* de coordenadas retangulares para coordenadas polares.

Uma operação que pode ser útil quando se trabalha com matrizes é a eliminação de linhas e/ou colunas. Se se quer eliminar a fila *k* da matriz *M*, basta atribuir uma matriz vazia à linha: ***M(k,:)=[ ]*** ou ***M(:,k)=[ ]***.

Em diversas aplicações, costuma – se denotar certas características de um sistema atribuindo – se 0 ao índice correspondente na matriz que representa o sistema. Por vezes é necessário avaliar a quantidade de zeros, a posição deles, dentre outras. A seguir, algumas funções relacionadas a zeros:

nnz(M) → Número de elementos não nulos de ***M***.
sparse(M) → Transforma ***M*** numa matriz esparsa.
spzeros(M) → Retorna uma matriz esparsa do tamanho de ***M***.
spget(M) → Converte uma matriz esparsa para o formato padrão (índices dos elementos não nulos, valor dos elementos não nulos e tamanho da matriz):

```
-->M
 M  =

   1.   2.   3.
   0.   1.   0.
   0.   0.   1.

-->A=sparse(M)
 A  =

(   3,   3) sparse matrix

(   1,   1)        1.
(   1,   2)        2.
(   1,   3)        3.
```

Laboratório de Vibrações

```
(   2,    2)        1.
(   3,    3)        1.

-->[ij,val,size_]=spget(A)
 size_  =

    3.    3.
 val  =

    1.
    2.
    3.
    1.
    1.
 ij  =

    1.    1.
    1.    2.
    1.    3.
    2.    2.
    3.    3.
```

Mais funções a respeito podem ser encontradas na biblioteca S*parse*.

A seguir, um exemplo de uso da fatoração QR para a extração dos autovalores de uma matriz:

```
clc
clear
close

// EXTRAI OS AUTOVALORES DA MATRIZ A POR MEIO DA FATORAÇÃO QR
// CÓDIGO ESCRITO POR DANUSIO GADELHA FILHO
// PERMITIDA REPRODUÇÃO COM OS DEVIDOS CRÉDITOS

format (8) // formato de exibição dos números: um  menor número de casas decimais

disp(' ')
disp('              Programa de extração de autovalores por fatoração QR')
disp('                        Danusio Gadelha Filho')
disp(' ')

A=[2.5 -1 0;
   -1 5 -sqrt(2);
   0 -sqrt(2) 10];

L=size(A);
I=eye(L(1),L(2));

disp('Matriz a ser fatorada -->')
disp(' ')
```

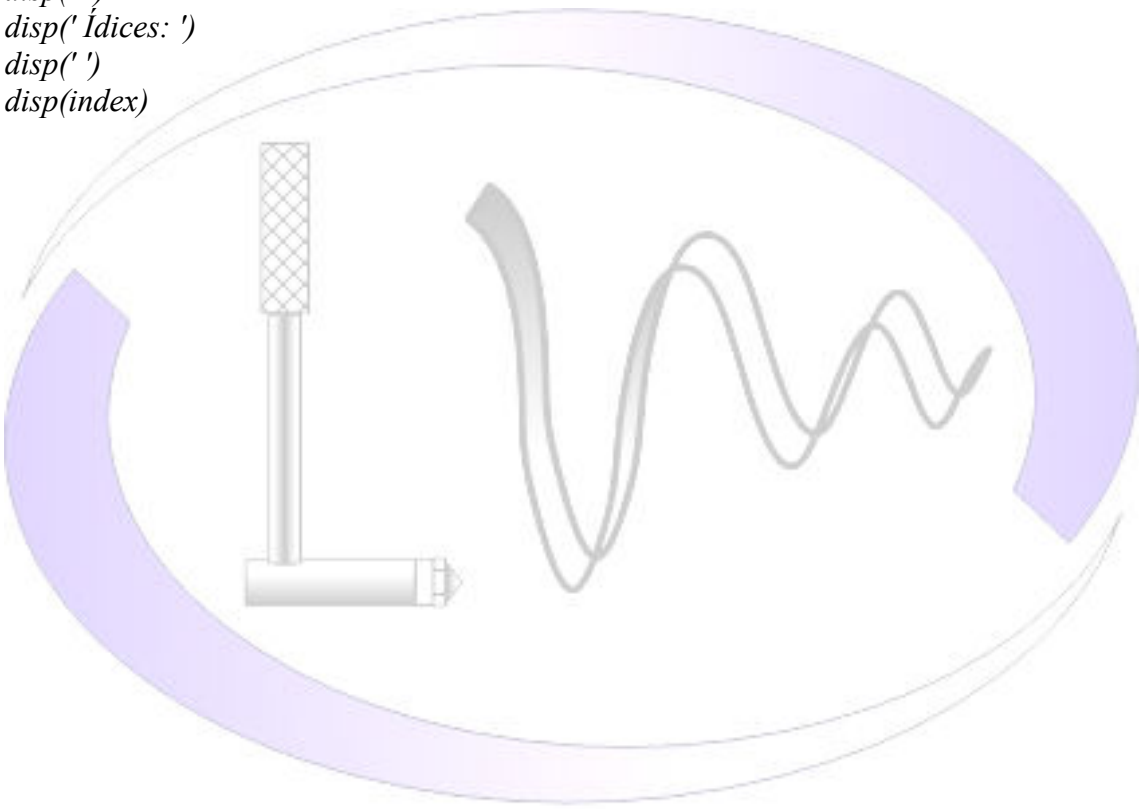
```
disp(A)
disp(' ')

format (16) // um maior número de casas decimais

[Q R]=qr(A);

N=100; // número de iterações
Q1=Q;
Q2=Q';
for i=1:N
   [Q R]=qr(A);
   Q1=Q*Q1;
   Q2=Q2*Q';
   A=R*Q;
end

printf(' Para %d iterações, a matriz de autovalores é: \n',N);
disp(' ')
disp(clean(A))
disp(' ')
```

O código apresentado abaixo retorna os valores de uma matriz ***M*** que estão acima ou abaixo de um valor de corte ***w***. Observe que não foram usados comandos condicionais (q.v. capítulo 2):

```
clc
clear
close

disp('                                Programa Filtro Lógico')
disp('                                 Danusio Gadelha Filho')
disp(' ')
disp('        Este script permite eliminar os elementos de uma matriz abaixo ou acima
de um determinado valor')
disp(' ')

M=[1/9 3/4 -1 7/6;
  1/4 1/3 1/2 1;
  1 1/4 1/9 -1/16];

L=size(M);

w=.3;
disp(' Matriz a ser reduzida: ')
disp(' ')
disp(M)
disp(' ')
printf(' Valor de corte: %f \n',w);
for i=1:L(1)
 for j=1:L(2)
```

Laboratorio de Vibrações

```
   A=bool2s((M(i,j)>w)); // sinal ">" --> script elimina x se x<=w
   M(i,j)=A*M(i,j);
  end
end
[index,M,sizeM]=spget(sparse(M));

disp('Matriz atualizada: ')
disp(' ')
disp(M)
disp(' Índices: ')
disp(' ')
disp(index)
```



Laboratório
de Vibrações

CAPÍTULO V

POLINÔMIOS

Esta é uma classe de variáveis, tal como *real* ou *lógica*. Um polinômio é gerado no Scilab através da função ***poly***. Caso sejam definidos dois polinômios sob a mesma variável, estes podem ser somados, subtraídos, multiplicados e divididos entre si e com números complexos.

Existem duas formas de se criar um polinômio no Scilab: especificando suas raízes ou seus coeficientes. Para a primeira forma, são permitidas três sintaxes:

```
P=poly([a b],'s');
P=poly([a b], 's', 'r');
P=poly([a b], 's', 'roots');
```

Quando se deseja a segunda alternativa, as opções são apenas duas:

```
P=poly([c d], 's', 'c');
P=poly([c d], 's', 'coeff');
```

Caso o usuário não queira trabalhar extensamente com uma variável polinomial (se quer apenas a raiz de um polinômio qualquer, por exemplo), pode – se trabalhar com as duas variáveis polinomiais disponibilizadas pelo Scilab: **%s** e **%z**.

As raízes de um polinômio podem ser obtidas com o comando ***roots(P)*** e seu valor numérico, pelo comando ***horner(P,N)***. Uma observação pertinente se refere à divisão de polinômios: caso não seja exata, é possível obter o resto e o quociente através da operação ***pdiv***:

```
[r q]=pdiv(P,Q);
```

A função ***coeff(P,n)*** retorna o coeficiente do polinômio ***P*** associado ao termo de grau ***n***.

O comando **clean** zera na matriz todos os elementos que são menores que a tolerância definida pelo usuário. A sintaxe é a que segue:

```
Q=clean(P,ta,tr)
```

Os parâmetros “ta” e “tr” são, respectivamente, as tolerâncias absoluta e relativa, abaixo das quais os termos serão zerados:

```
>>x=[1,2*%s-1,(3+%s^2)/(%s-2);-2+%s,3/(2*%s^3-%s),2];
>>clean(x,.9,.5)
// ------------------------------------------------------------------------------
>>x=[1,2*%s-1,(3+%s^2)/(%s-2);-2+%s,3/(2*%s^3-%s),2;1,2*%s-5, +3/%s];
>>x*inv(x)
>>clean(x*inv(x))
```

Laboratório de Vibrações

Este comando é muito em procedimentos numéricos, quando se opera com vetores extensos e se deseja reduzir o tempo de execução.

A função **coffg** retorna as matrizes inversas do numerador e denominador da matriz polinomial em questão:

```
x=[1,2*%s-1,(3+%s^2)/(%s-2);-2+%s,3/(2*%s^3-%s),2;1 2*%s -5+3/%s];
[N,D]=coffg(x);
N/D-inv(x)
```

O quociente da divisão dos dois termos de saída da função **coeffg** é exatamente a matriz inversa da matriz de polinômios inicial.

Dada uma matriz polinomial, o comando **degree** retorna uma matriz de mesma ordem do argumento de entrada com o grau de cada elemento.

As funções **denom** e **numer** retornam, respectivamente, os denominadores e os numeradores de uma matriz polinomial racional.

Um dos comandos mais úteis para polinômios é o **derivat**, que deriva todos os termos de uma matriz polinomial (racional inclusive).

O determinante de uma matriz polinomial pode ser calculado pela função **detr**.

A função **hrmt** retorna o fator comum de um vetor linha polinomial e uma matriz *U* de determinante igual a 1 e tal que o produto *v*U* (onde *v* é o vetor linha em questão) é igual a um vetor *1xn* (onde *n* é o comprimento de v) onde o último termo é o fator comum de *v*.

```
>>x=poly(0,'x');
>>v=[x*(x^2-1),x^2*(x+1),(x-2)*(x^2+2*x+1),(3*x^2+2)*(x+1)^3];
>>[pg,U]=hrmt(v);
U=clean(U)
//pg = fator comum
>>det(U)
```

A triangularização da matriz polinomial *M* pode ser feita através do comando ***htrianr(M)***. A matriz de entrada não pode conter termos fracionários.

Se uma matriz polinomial *M* for dada na forma $M = C * (xA - I)^{-1} * B$, onde x é a variável dos polinômios de *M*, os termos *A, B* e *C* podem ser encontrados pela função ***pol2des(M)***. Os termos *A,B* e *C* são chamados de matrizes descritoras de *M*.

A função ***polfact(P)*** retorna os fatores mínimos do polinômio (fatores que, multiplicados, resultam no polinômio). O argumento de saída é um vetor contendo os mínimos fatores.

Uma fração polinomial pode ser simplificada através do uso da função ***simp***, que retorna a forma irredutível da matriz.

A seguir, uma forma de encontrar os autovalores de uma matriz ***M*** utilizando polinômios:

```
clc
clear
close

// EXTRAI OS AUTOVALORES DA MATRIZ A POR MEIO DAS RAÍZES DO POLINÔMIO CARACTERÍSTICO
// CÓDIGO ESCRITO POR DANUSIO GADELHA FILHO
```

```
// PERMITIDA REPRODUÇÃO COM OS DEVIDOS CRÉDITOS

format (8) // formato de exibição dos números: um  menor número de casas decimais

disp(' ')
disp('              Programa de extração de autovalores por polinômio característico')
disp('                        Danusio Gadelha Filho')
disp(' ')

A=[2.5 -1 0;
  -1 5 -sqrt(2);
  0 -sqrt(2) 10];

L=size(A);
I=eye(L(1),L(2));

disp('Matriz a ser fatorada -->')
disp(' ')
disp(A)
disp(' ')

polcar=det(A-I*%s);

disp('O poliômio característico é: ')
disp(' ')
disp(polcar)
disp(' ')

format (16) // um maior número de casas decimais

lambda=roots(polcar);

disp(' Os autovalores são: ');
disp(' ')
disp(lambda)
disp(' ')
```

Script com funções adicionais que utilizam polinômios:

```
clc
clear
close

format (8) // formato de exibição dos números: um  menor número de casas decimais

disp(' ')
disp('                  Algumas funções extra para polinômios')
disp('                        Danusio Gadelha Filho')
disp(' ')

disp('                          Fatoração Espectral')
disp(' ')

z=poly(0,'z');
```

Laboratório de Vibrações

```
p=(z-1/2)*(2-z)*(3.5-z)*(z-2/7) // Cada raíz de p tem um inverso simétrico que também é raíz de p
w=sfact(p) // as raízes de w são as raízes antiestáveis de p
disp(' ')
disp(' Verificação: ')
disp(' ')
disp(w*numer(horner(w,1/z))) // igual a p
disp(' ')

disp('                          Equação de Bezout')
disp('                            p*a+q*b=r')
disp(' ')

p=poly([1 2],'x','c');
q=poly([10 .2],'x','c');

a=poly([2 1/2],'x','c');
b=poly([-5 1],'x','c');

r=p*a+q*b;
[x,err]=diophant([p,q],r)

rapproach=clean(p*x(1)+q*x(2))
r
disp(' ')

disp('                           Forma Hermitiana')
disp(' ')

s=poly(0,'s');

disp(' Matriz a ser transformada -->')
disp(' ')
p=[s, s*(s+1)^2; 2*s^2 s-1];
disp(p)
[Ar,U]=hermit(p'*p);
disp(' ')
disp(' Forma Hermitiana de p: ptrans*p*Utrans -->')
disp(' ')
clean(Ar)
disp(' Matriz unimodular U -->')
disp(U)
disp(' ')
detU=clean(det(U))
```

Laboratório de Vibrações

CAPÍTULO VI

LISTAS

São outra classe de variáveis, parecidas com as matrizes, mas que agrupam conteúdo de diferentes tipos.

São implementadas através do comando ***list***, seguido dos elementos os quais se desejam agrupar.

```
-->L = list('Danusio',rand(3,2),(1/3*1/5*1/7-1),[%F %T %T %F])
```

Uma operação comum quando se trabalha com listas é substituir um determinado elemento da lista por outra lista:

```
--> L(4)=list('God',[%T %T],12*12);
--> L
--> L(4)(2)(1,2)
```

As listas são consideradas *estruturas de dados* e têm sua biblioteca própria. A seguir, serão explicados diversos comandos e operações relacionadas a estruturas de dados.

O comando ***cell*** cria um vetor de matrizes vazias segundo uma ordem definida. Esta ordem vem definida por outra matriz, que é uma matriz de índices. Considere o exemplo a seguir, em que é criada uma matriz nula *c* de ordem 2 x 3 x 4 x 2:

```
-->m=[2 3;4 2];

-->c=cell(m)
 c  =


(:,:,1,1)

!{}  {}  {}  {}  !
!                !
!{}  {}  {}  {}  !
(:,:,2,1)

!{}  {}  {}  {}  !
!                !
!{}  {}  {}  {}  !
(:,:,3,1)

!{}  {}  {}  {}  !
!                !
!{}  {}  {}  {}  !
(:,:,1,2)
```

Laboratório de Vibrações

```
!{}  {}  {}  {} !
!               !
!{}  {}  {}  {} !
(:,:,2,2)

!{}  {}  {}  {} !
!               !
!{}  {}  {}  {} !
(:,:,3,2)

!{}  {}  {}  {} !
!               !
!{}  {}  {}  {} !
```

No exemplo, temos uma matriz 2 x 3 cujos elementos são matrizes 4 x 2 nulas. A utilidade de matrizes nulas reside na necessidade de inicialização de matrizes variáveis, destacadamente quando são usados laços.

O comando ***hypermat*** inicializa uma matriz N-dimensional através de uma série de dados:

```
-->M=hypermat([2 3 2 1],[rand(1,5) eye(1,5) zeros(1,2)]);

-->M
 M  =

(:,:,1,1)

   0.1280058   0.2119030   0.6856896
   0.7783129   0.1121355   1.
(:,:,2,1)

   0.   0.   0.
   0.   0.   0.
```

Note que o produto dos índices entre colchetes deve ser igual ao número de elementos da matriz ao lado.

O comando ***struct*** é a forma mais simples de se criar uma estrutura de dados. Apesar da simplicidade, esta função dá uma boa dinâmica à estrutura, permitindo uma substituição intuitiva dos campos e a aquisição de novos campos.

```
-->dados=struct('Nome','Danusio','Idade',18,'Renda Mensal',100000);
```

Neste caso, os campos são “Nome”, “Idade” e “Renda Mensal”, e os valores armazenados são “Danusio”, “18” e “100000”. Um dado pode ser acessado através do nome da estrutura seguido por ponto e o nome do campo: *dados.Nome*. Um novo campo pode ser adicionado seguindo a mesma sintaxe:

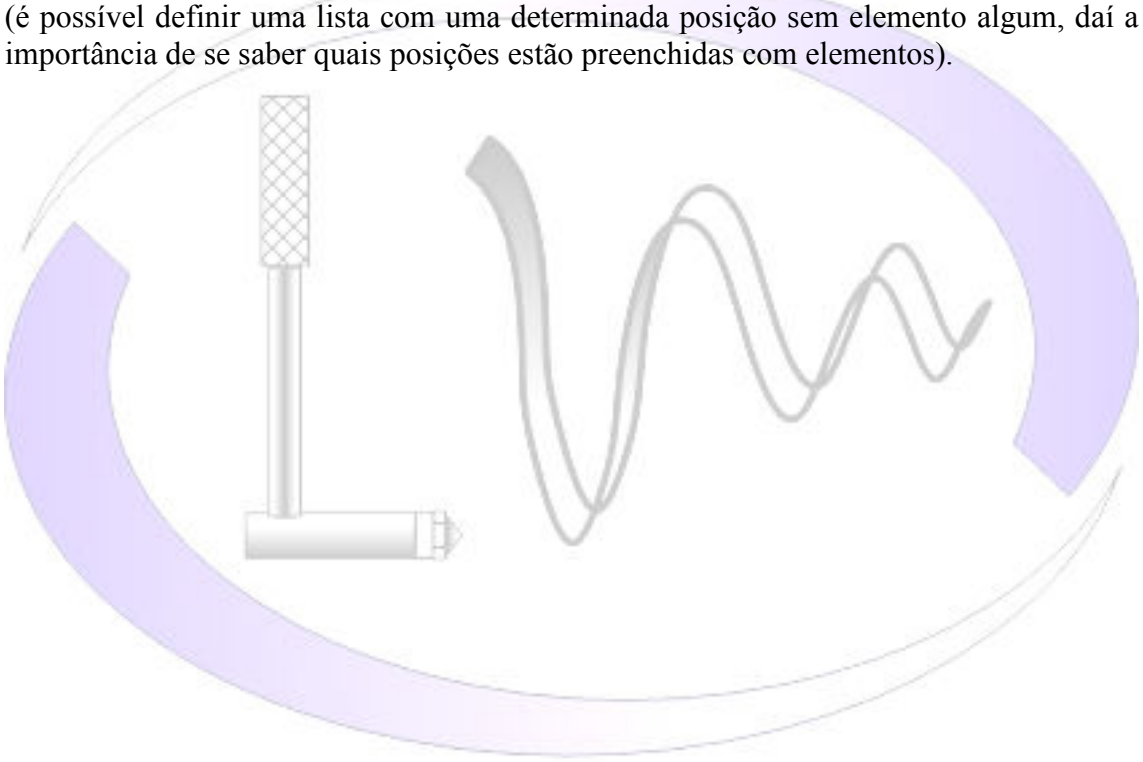
```
-->dados.Altura=182;
```

Laboratório de Vibrações

```
-->dados
 dados  =

   Nome: "Danusio"
   Idade: 18
   Renda Mensal: 10000
   Altura: 182
```

A função ***lstcat*** concatena várias listas numa variável só: ***lstcat(l1,l2,...,ln)***.

Por fim, a função ***definedfields*** retorna os índices definidos dentro de uma lista (é possível definir uma lista com uma determinada posição sem elemento algum, daí a importância de se saber quais posições estão preenchidas com elementos).



Laboratório de Vibrações

CAPÍTULO VII

FUNÇÕES

Funções são procedimentos sintetizados em uma só operação de comando. Considere a operação de obter os números pares até um número *N*. Podemos sintetizá – la em uma função, por exemplo *pares*. Os comandos obrigatórios para a definição de uma função são ***function*** **outputs=*nome_da_função*** **(inputs)** e ***endfunction***.

```
function n=pares(N)
  n=0;
  for i=1:N
    if modulo(i,2)==0
      n=n+1;
    end
  end
endfunction
```

A partir da execução deste script, a função ***pares*** pode ser executada normalmente, como se fosse um comando interno do Scilab.

Caso se tenha uma relação direta entre o *output* e os *inputs*, é mais conveniente fazer uso da função ***deff***. Considere o exemplo a seguir:

```
deff('r=quadr(a,b,c)','r=[(1/(2*a)*(-b+sqrt(b^2-4*a*c)));(1/(2*a)*(-b-sqrt(b^2-4*a*c)))]');
```

A definição anterior retorna as raízes de uma equação quadrática em função dos coeficientes.

```
-->quadr(1,-5,6)
 ans  =

    3.
    2.
```

O comando ***deff*** requer duas strings: o escopo da função (*r=quadr(a,b,c)*) e a relação entre o output e os inputs. Este tipo de função que retorna apenas um argumento é também chamado de procedimento.

A seguir, temos um exemplo de função recorrente em cálculo vetorial:

```
// função que calcula as diferenças entre elementos sucessivos de um vetor x

function x=diff(X)
  n=length(X);
  for i=1:(n-1)
    x(i)=X(i+1)-X(i);
  end
```

Laboratório de Vibrações

```
endfunction

// Calcula o comprimento da linha formada pelos pontos de coordenada (x,y)

deff('L=compl(x,y)',['dx=diff(x)';'dy=diff(y)';'L=sum(sqrt(dx.^2+dy.^2))']);
```

O uso de de funções e procedimentos se torna quase obrigatório em programas muito longos, na medida em que permitem subdividir o código, o que facilita a debugagem, e realizar comandos comuns na rotina de forma mais prática. A seguir, uma função muito útil em problemas de interpolação: permite achar os valores imediatamente acima e abaixo de um pré – determinado em uma série de dados:

```
function [sup,inf]=limits(X,c)
B=bool2s(X>=c);
C=bool2s(X<=c);

S1=B.*X;
S2=C.*X;

[v,posS]=min((1-S1)^-1);
[v,posI]=max(S2);

sup=X(posS);
inf=X(posI);
endfunction

X=[-2 3 5 1 8];
c=4;

[s,i]=limits(X,c)
```

Laboratório de Vibrações

CAPÍTULO VIII

GRÁFICOS

GRÁFICOS BIDIMENSIONAIS – São constituídos de dois eixos, sendo necessário, portanto, dois argumentos de entrada para a execução, que, na verdade, são vetores com a mesma dimensão. As funções responsáveis pela plotagem 2D são ***plot***, ***plot2d, fplot2d e contour2d***. É importante notar que, caso um eixo seja função do outro, todas as operações devem ser realizadas elemento – a – elemento .

O comando ***plot2d*** plota uma curva bidimensional. Aceita quatro especificações (considere um eixo –x definido de 0 a 5):

plot2d1: plotagem padrão (linear):

```
clc
clear
close

x=0:.01:5;
y=sin(x.^2);
plot2d1(x,y)
```

plot2d2: plotagem discreta (histograma ou degraus):

```
clc
clear
close

x=0:.1:5;
y=sin(x.^2);
plot2d2(x,y)
```

plot2d3: plotagem em barras verticais:

```
clc
clear
close

x=0:.05:5;
y=sin(x.^2);
plot2d3(x,y)
```

plot2d4: plotagem em setas:

```
clc
clear
close
```

Laboratório de Vibrações

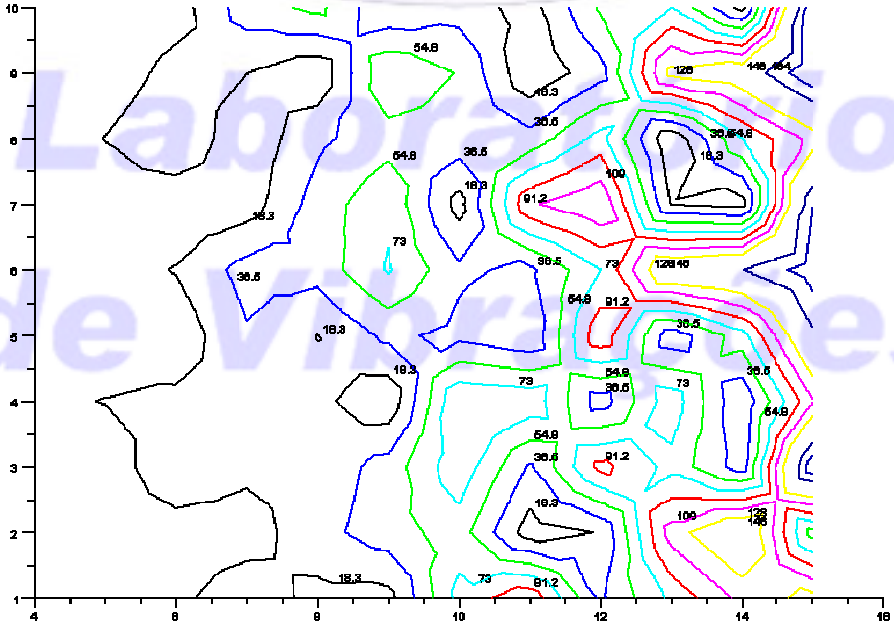
```
x=0:.05:5;
y=sin(x.^2);
plot2d4(x,y)
```

O comando ***fplot2d*** plota uma função definida por ***function*** ou ***deff***.

```
clc
clear
close

deff("x=s(t)",["x=2*sin(.3*t)"]);
t=0:.1:16*%pi;
fplot2d(t,s)
```

O comando ***contour2d*** plota curvas de nível com base numa matriz de dados geométricos da superfície a ser modelada:

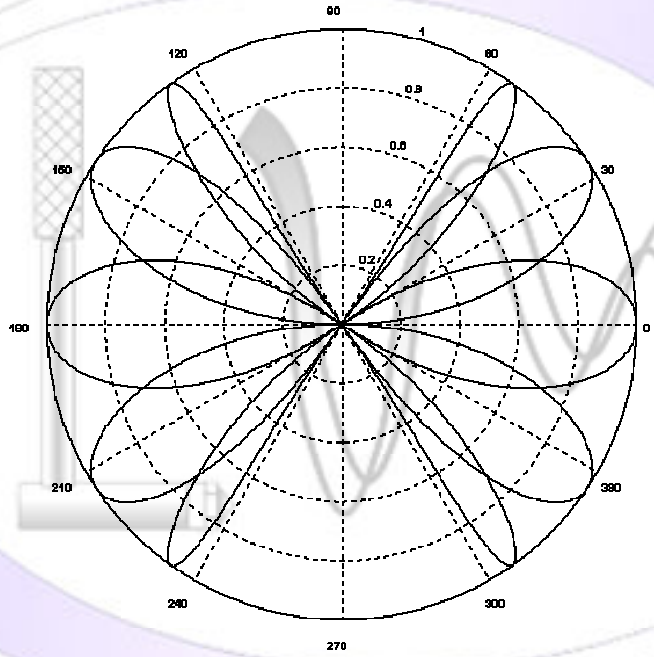
```
clc
clear
close

nz=10; // número de níveis
a=eye(5,10)+rand(5,10)+ones(5,10);// matriz para a plotagem
z= min(a) + (1:nz)*(max(a)-min(a))/(nz+1); //valor numérico de cada nível
x=size(a);
contour2d(1:x(1),1:x(2),a,nz);
```



Laboratório de Vibrações

O comando ***polarplot*** plota as variáveis em coordenadas polares:

```
clc
clear
close

t= 0:.01:2*%pi;
polarplot(sin(2*t),cos(5*t));
```



O comando ***plot*** realiza as operações de plotagem de forma mais geral, com o acréscimo de propriedades:

POSIÇÃO DOS EIXOS:

```
x=0:0.1:4*%pi;

plot(x,sin(2*x))
a=gca(); // definição da janela de plotagem

a.y_location ="right";// left, middle
a.x_location ="top"; // bottom, origin
a.isoview='on'; //posição centralizada na janela
```

PROPRIEDADES DE TRAÇO:

MARCADOR	SIGNIFICADO
-	Solid line (default)

Laboratório de Vibrações

--	Dashed line
:	Dotted line
-.	Dash-dotted line
r	Red
g	Green
b	Blue
c	Cyan
m	Magenta
y	Yellow
k	Black
w	White
+	Plus sign
o	Circle
*	Asterisk
.	Point
x	Cross
'square' or 's'	Square
'diamond' or 'd'	Diamond
^	Upward-pointing triangle
v	Downward-pointing triangle
>	Right-pointing triangle
<	Left-pointing triangle
'pentagram'	Five-pointed star (pentagram)
'none'	No marker (default)

EXEMPLO:

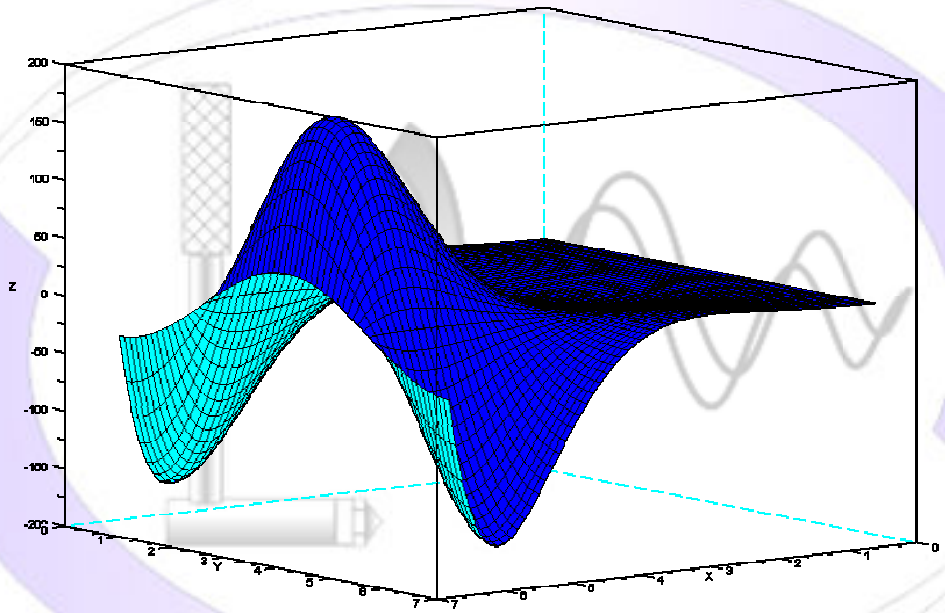
```
x=0:0.1:2*%pi;
plot(x,sin(2*x),'rpentagram')

a=gca(); // definição da janela de plotagem

a.y_location ="middle"; // left, middle
a.x_location ="middle"; // bottom, origin
```

GRÁFICOS TRIDIMENSIONAIS: O comando mais utilizado é o ***plot3d***. O seu uso pode ser observado na rotina a seguir:

```
clc
clear
close

t=[0:0.1:2*%pi]';
z=(sin(t).*exp(t))*cos(t');
plot3d(t,t,z)

-->size(t)
```

Laboratório de Vibrações

```
 ans  =

    63.    1.

-->size(z)
 ans  =

    63.    63.
```

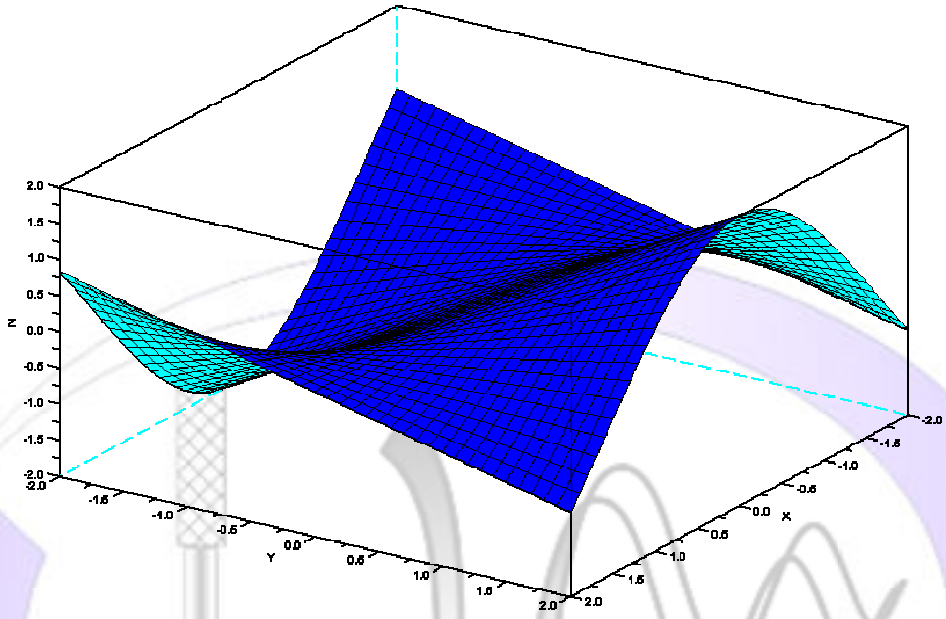


Observe que, se o eixo base tiver a dimensão *1 x N*, a matriz imagem deste eixo tem dimensão *N x N*. Esta função tem uma derivação, ***plot3d1***, que gera uma figura em cores múltiplas.

Um modo mais rápido é gerar superfícies a partir de funções definidas pelo comando ***deff***, através do comando ***eval3d***:

```
clc
clear
close

x=-2:.1:2;
 y=x;
 deff('[z]=f(x,y)',['z= cos(x).*y']);
 z=eval3d(f,x,y);
 plot3d(x,y,z);
```

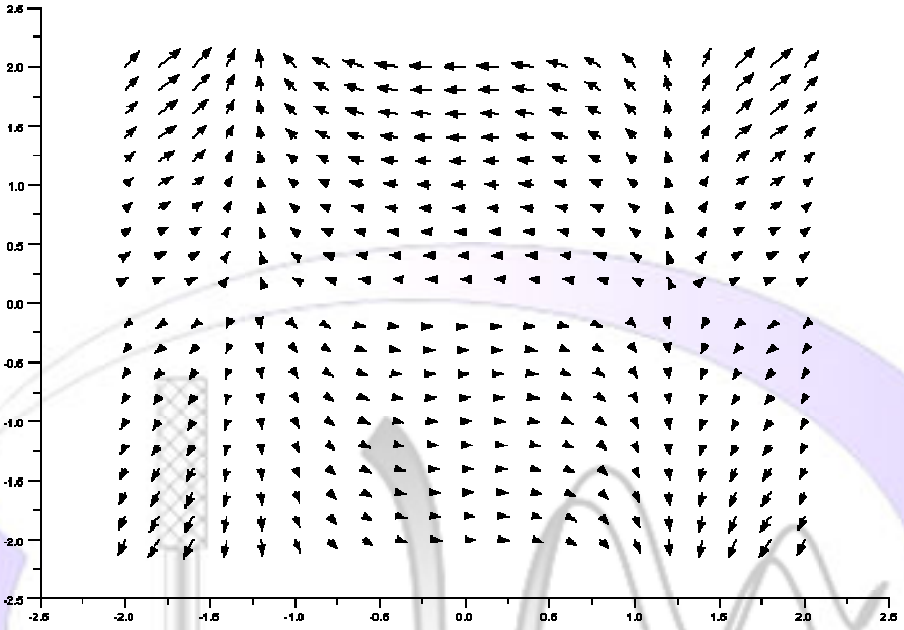
Laboratório de Vibrações



CAMPO VETORIAL: Um campo vetorial é descrito por duas funções ***fx(x,y)*** e ***fy(x,y)***, que são entendidas como matrizes contendo os valores das duas componentes em função das coordenadas x e y. Observe o código abaixo, em que a função ***champ*** plota um campo vetorial (use a função ***champ1*** para plotar em cores):

```
clc
clear
close

x=-2:.2:2;
y=-2:.2:2;

fx=-cos(x.^2)'*(y.^(2/3)); // matriz com as coordenadas x
fy=(sin(x).^2)'*y; // matriz com as coordenadas y

champ(x,y,fx,fy)
```

Laboratório de Vibrações



Uma curva tridimensional pode ser obtida através do comando ***param3d***:

Código 1

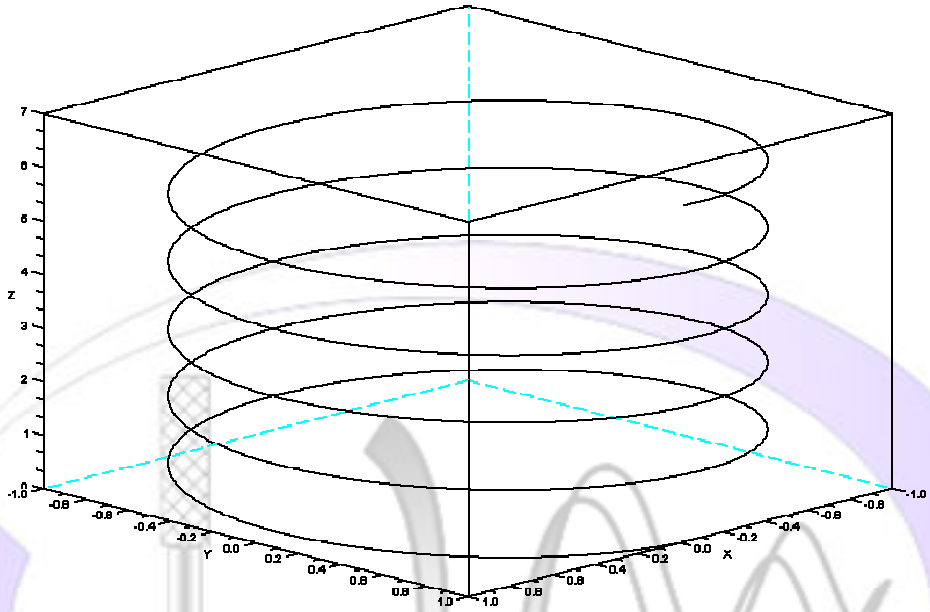
```
clc
clear
close

t=0:0.1:10*%pi;
param3d(sin(t),cos(t),t/5)
```

Código 2

```
clc
clear
close

t=[0:0.1:5*%pi]';
param3d1([sin(t),sin(2*t)],[t,cos(2*t)],[t t]);
```

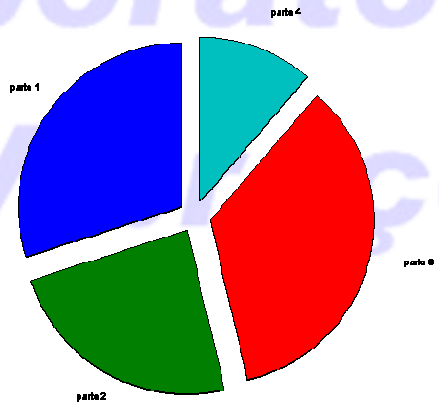
A função ***param3d1*** permite a plotagem de várias curvas na mesma janela.

Laboratório de Vibrações



O comando ***pie*** plota um gráfico em pizza. Abaixo estão explicitados todos os atributos deste comando:

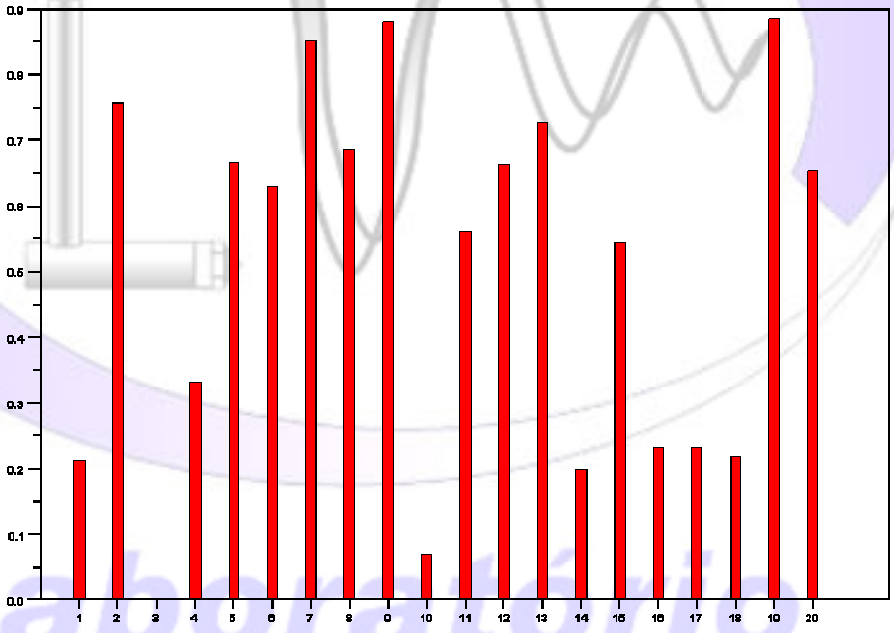
```
clc
clear
close
pie([53 42 62 20],[2 1 1 1],["parte 1","parte 2","parte 3","parte 4"]);
// Valores, distâncias entre as fatias, rótulos.
```



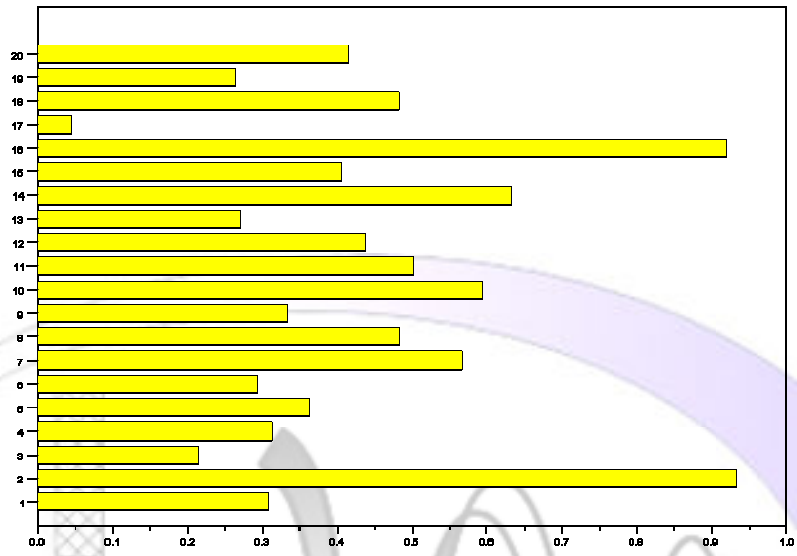
Laboratório de Vibrações

Um histograma é gerado pelas funções ***bar*** e ***barh*** aplicado numa matriz. Define – se também a largura e a cor das barras plotadas:

```
clc
clear
close

y=rand(1,20);

scf(1)
bar(y,.25,'r')
scf(2)
barh(y,.75,'y') // barras horizontais
```



Laboratório de Vibrações



Caso se ponha uma matriz para ser plotada (os exemplos anteriores são para um vetor), o eixo das abscissas (ou ordenadas, no caso de ser usado ***barh***) terá valores cardinais iguais à ordem das linhas da matriz.

Outras funções podem ser encontradas na biblioteca ***Graphics Library***. As figuras presentes neste material foram exportadas para formato de imagem a partir do menu da janela de gráfico gerada: **File --> Export to...** O menu ainda oferece a possibilidade de se alterar as propriedades do gráfico e da janela.

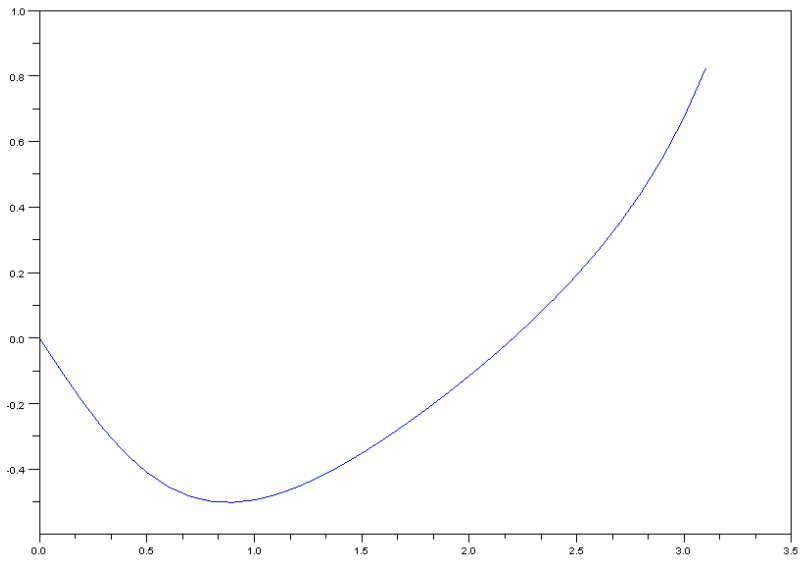
Laboratório de Vibrações

APÊNDICE

EQUAÇÕES DIFERENCIAIS

A função ***ode*** resolve numericamente uma equação ordinária de primeira ordem, considerando como variável a derivada primeira da função y. A relação entre a função e sua derivada tem que ser explicitada através de uma função, como no exemplo a seguir:

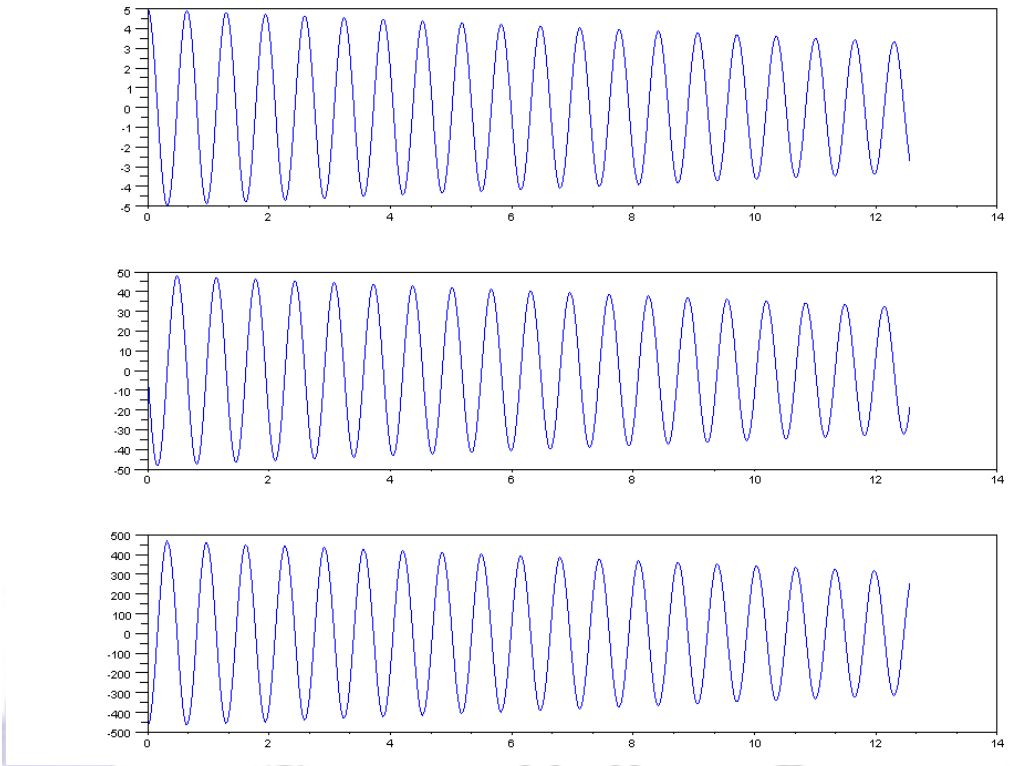
```
clc
clear
close

// dy/dt=y^2-y sin(t)-cos(t), y(0)=0

function ydot=f(t,y)
  ydot=y^2-y*sin(t)-cos(t);
endfunction

y0=0; // valor inicial de y no tempo inicial
t0=0; // tempo inicial
t=0:0.1:%pi; //escla de tempo

y=ode(y0,t0,t,f);

plot(t,y)
```



Para soluções de ordem superior, substitui – se ***y*** e ***ydot*** por dois vetores, como é mostrado a seguir, que resolve uma equação de segunda ordem:

```
clc
clear
close

function s=sgn(v)
  N=length(v);
  for i=1:N
    if v(i)==0
      s(i)=0;
    else
      s(i)=abs(v(i))/v(i);
    end
  end
endfunction

function ydot=f(t,y)
  x=y(1,:);
  v=y(2,:);
  ydot(1,:)=v;
  ydot(2,:)=-(3.8724846^2)*2*%pi*x-0.2096573*9.81*[sgn(v)]';
endfunction

y0=[5;0];
t0=[0;0];
t=[0:0.01:4*%pi];

y=ode(y0,t0,t,f);
ydot=f(t,y);

a=ydot(2,:);
v=y(2,:);
x=y(1,:);

subplot(3,1,1)
plot(t,x)
subplot(3,1,2)
plot(t,v)
subplot(3,1,3)
plot(t,a)
```

Laboratório de Vibrações



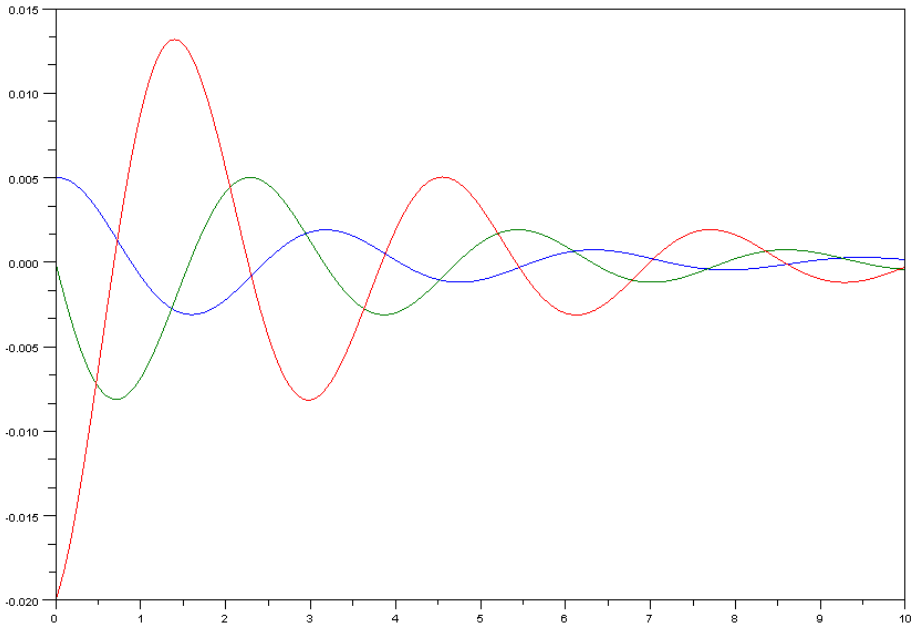
Observe que, em um vetor, temos as derivadas da função da ordem 0 (a própria função) até à ordem n-1, onde *n* é a ordem da equação, e no outro, as derivadas de ordem 1 à n.

Uma alternativa ao uso da função ***ode*** é utilizar laços para obter os valores da função em cada passo. Considere a equação ***y" + 4y' + 0,8y = 0***. O script para resolvê – la está abaixo, juntamente com o gráfico gerado:

```
clc
clear
close

deff('y2dot=f(ydot,y)','y2dot=-4*y-0.8*ydot');

y(1)=5e-3;
ydot(1)=0;
y2dot(1)=f(ydot(1),y(1));

dt=.05;
t=0:dt:10;
for i=2:length(t)
  ydot(i)=ydot(i-1)+dt*y2dot(i-1);
  y(i)=y(i-1)+dt*ydot(i-1);
  y2dot(i)=f(ydot(i),y(i));
end

Y=[y ydot y2dot]; // [azul,verde,vermelho]
X=[t' t' t'];
```

Laboratório de Vibrações

plot(X,Y)



O princípio é simples e bastante intuitivo: dados os valores iniciais de velocidade e espaço, obtemos a aceleração pela substituição na equação diferencial; os valores seguintes de deslocamento e aceleração são obtidos pela expressão (discretização da definição de derivada): ydot(passo i)=ydot(passo i-1)+dt*y2dot(i-1) e y(passo i)=y(passo i-1)+dt*ydot(i-1). A partir daí, o processo se repete.

Laboratório de Vibrações