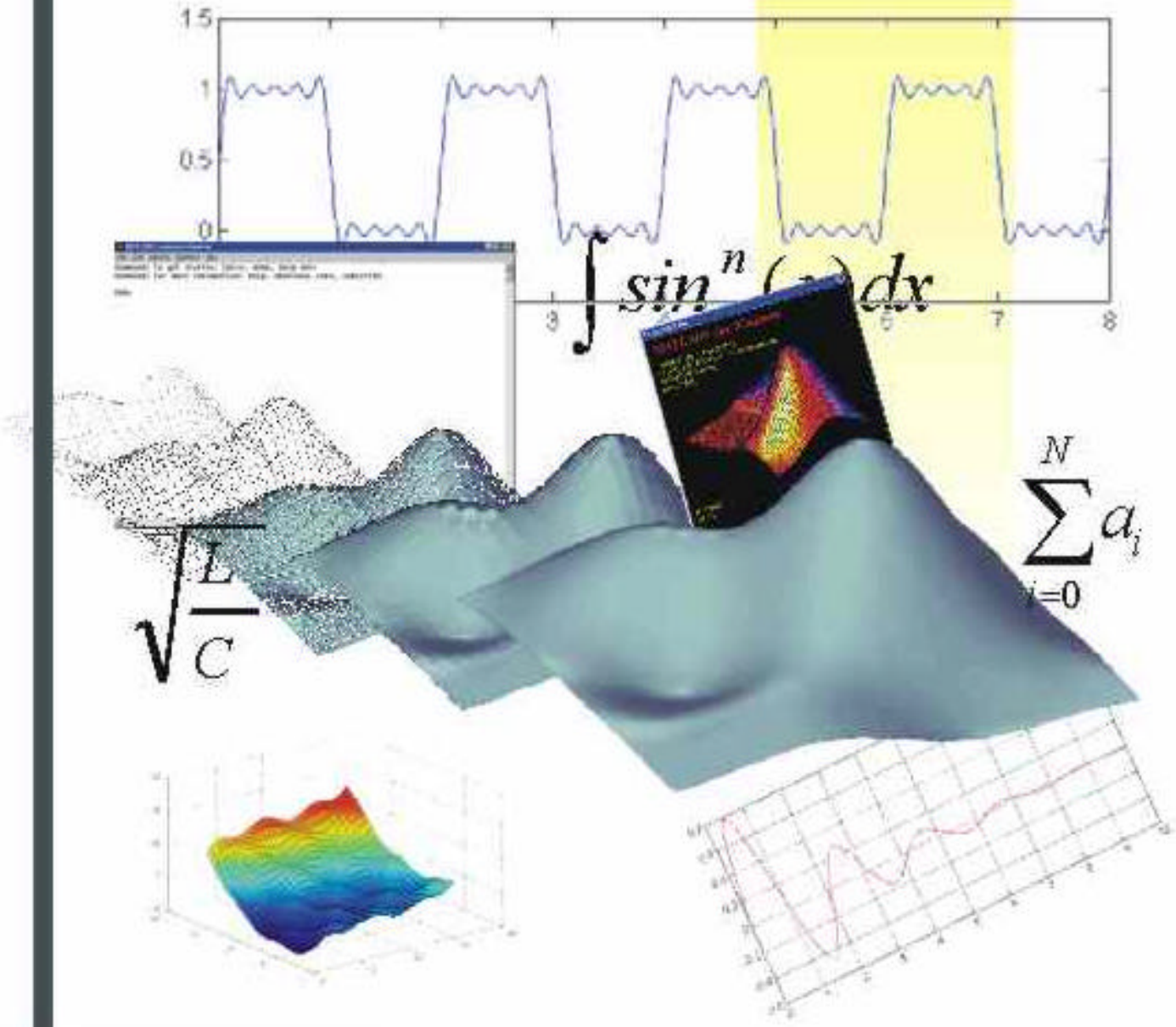
MatLab



Allan de Medeiros Martins

Engenharia Elétrica

ninguem@summer.com.br

Natal, 06 de maio de 1998 - UFRN

Nota do autor...

Esta apostila/tutorial foi elaborada com base no MatLab versão do estudante 4.0, com o ToolBox SigSys e Simbolic. Todos os programas apresentados foram testados nesta versão do MatLab, porem grande maioria dos capítulos tem uso com qualquer versão do MatLab. Os capítulos sobre tópicos avançados e gráficos possuem comandos de criação de controles do Windows que requer o MatLab versão do estudante ou uma versão superior.

Qualquer duvida, sugestão, crítica ou entrar em contato comigo, por favor me envie um e-mail.

Allan de Medeiros Martins, Engenharia Elétrica
ninguem@summer.com.br

Sumário

1 – Introdução
1.1 – Como o MatLab trabalha 4
1.2 – Ambiente de trabalho do MatLab 4
1.3 – Arquivos .m 5
1.4 – Janelas 5
2 – Comandos básicos
2.1 – Comandos de propósito geral 6
2.2 – Comandos do sistema operacional 6
2.3 – Erros no MatLab 6
2.4 – HELP 7
3 – Operações simples
3.1 – Operações [+ - / * ^ ‘] 8
3.2 – Números no MatLab 9
3.3 – Infinito, raiz de –1, indeterminações 9
4 – Variáveis
4.1 – Oque é uma variável 11
4.2 – Nomes de variáveis 11
4.3 – Tipos de variáveis 11
4.4 – Operações com variáveis 12
4.5 – A variável ans 12
4.6 – Manipulação 12
5 – Vetores e matrizes
5.1 – Vetores e Matrizes no MatLab 13
5.2 – Operações 13
5.3 – Operações elemento-elemento 14
5.4 – Operando com elementos 15
5.4.1 – Indexando mais de um elemento 16
5.5 – Operadores [ ] 17
5.6 – Funções matriciais 18
6 – Números complexos
6.1 – Sqrt(-1) 19
6.2 – Operações 19
6.3 – Usando números complexos 19
6.4 – Algumas funções com números complexos 19
7 – Funções básicas
7.1 – Uso de funções 20
7.2 – Funções que retornam mais de um valor 21
7.3 – Algumas funções básicas 21
8 – Gráficos
8.1 – Gráficos no MatLab 23
8.2 – Tipos de gráficos 23
8.3 – Partes de um gráfico 23
8.4 – Plotando gráficos 24
8.5 – Gráficos 2D 24
8.5.1 – Acabamento de um gráfico 25
8.6 – Gráficos 3D 26
8.7 – Algumas funções gráficas 29
8.8 – Gráficos avançados 29
8.8.1 – Plotando vários gráficos na mesma janela 29
8.8.2 – Mudando cores e outras propriedades 30
8.8.3 – Usando o mouse para posicionar textos 31
9 – Arquivos
9.1 – Uso de arquivos 32
10 – Programação
10.1 – Funções lógicas e operadores lógicos 33
10.2 – Noções de algoritmos 34
10.2.1 – Estrutura básica de um programa 35
10.2.2 – Regras de construção 35
10.3 – A linguagem do MatLab 36

10.3.1 – A variável path 36
10.4 – Loops, condicionais, e comandos de programação 37
10.4.1 – If 37
10.4.2 – For 38
10.4.3 – While 38
10.4.4 – Definindo funções 39
10.4.4.1 – Variáveis globais 40
10.5 – Considerações importantes sobre programação 40
11 – ToolBoxes
11.1 – Oque são ToolBoxes 41
11.2 – Exemplos 41
11.2.1 – ToolBox simbolic 41
11.2.2 – ToolBox sistema de controle 44
12 - Tópicos avançados
12.1 – Controles e janelas do Windows 47
12.2 – Criando janelas e controles no MatLab 47
12.3 – Os comandos SET e GET 49
12.4 – Controlando os controles 49
12.5 – uicontrol 50
12.5.1 – parâmetros do uicontrol 50
12.6 – uimenu 52
12.6.1 – Parâmetros do uimenu 52
13 – Exemplo de aplicação 53

Bibliografia 57

Apêndice 58

MatLab

1 – Introdução

O MatLab é uma ferramenta para uso de cálculos computacionais muito útil para engenharia em geral. É um programa que possibilita a construção de programas e a realização de cálculos comuns em engenharia. Hoje em dia o MatLab se encontra em sua versão para Windows e já possue muitos *Toolboxes* com aplicações específicas das mais variadas áreas da engenharia. Tudo isso graças a facilidade de manipulação numérica e matricial do MatLab.

1.1 – Como o MatLab trabalha

Por causa de sua grande variedade de utilizações e possibilidade programacional, o MatLab trabalha sob forma de linha de comandos. Sua *janela* é simples e possue um menu bem básico. Todos os comandos devem ser digitados no *prompt* do MatLab, e os programas devem ser guardados em arquivos especiais.

Por ex.: para somar dois números do MatLab simplesmente digite:

```
» 2+2

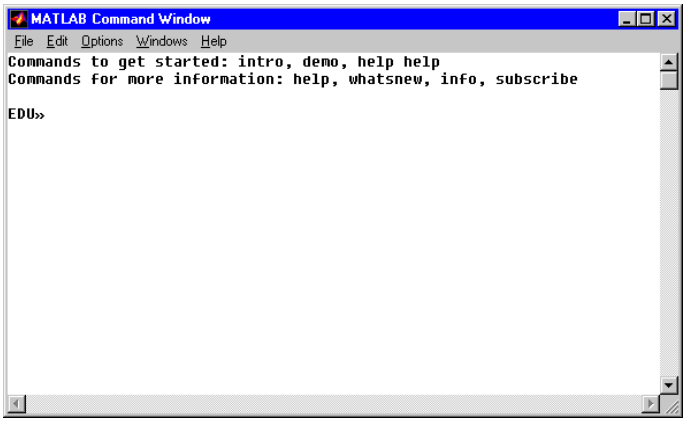
ans =

     4
```

Qualquer comando que queira executar digita-se diretamente no *prompt* e o resultado sai no ambiente de trabalho ou em janelas que são abertas automaticamente no próprio MatLab, que são outra forma de *interface* que o MatLab utiliza para mostrar ou receber dados. Por Exemplo, um gráfico é mostrado sempre em uma janela.

1.2 – Ambiente de trabalho do MatLab

O ambiente de trabalho do MatLab é na própria janela do programa sob forma de *prompt* onde o usuário entra com dados e comandos. A janela da área de trabalho possue uma *barra de rolagem* que permite acompanhar todas as saídas produzidas pelo MatLab enquanto o usuário está trabalhando. A janela de trabalho também possibilita a edição simples do Windows, ou seja, o usuário pode *colar* texto para o *prompt* e utiliza-lo como comando, da mesma forma pode-se também *copiar* textos ou saídas do MatLab para outros programas do Windows.



Área de trabalho do MatLab

1.3 – Arquivos .m

O MatLab permite que os comandos sejam executados de forma seqüencial em forma de programa, constituindo-se assim de uma linguagem interpretada. As seqüências de comandos são gravadas como texto em um arquivo especial com extensão .m e são executados simplesmente digitando-se o nome do arquivo direto no *prompt*.

Ex.:

```
» meuprog
 Programa de teste...
2+2=?

ans =

     4
```

Para que um arquivo seja executado dessa forma, o mesmo deve estar no diretório atual do MatLab ou então estar em alguns diretórios denominados *path* que será visto mais adiante.

1.4 – Janelas

Outra forma de comunicação do MatLab com o usuário é através de janelas. Elas podem servir de “quadro” para exibição de gráficos, ou como “formulário” para ser preenchido com dados entrados pelo usuário.

2 – Comandos básicos

Os comandos no MatLab podem ser divididos em vários “tipos” conforme seu propósito.

2.1 – Comandos de propósito geral

Alguns comandos são de uso geral, e normalmente auxiliam na utilização da área de trabalho, são eles:

clc – limpa a tela
home – move o cursor para o topo da tela
disp – exibir um texto na área de trabalho
echo – ligar ou desliga a exibição dos comandos na execução de um programa
format FORMA – ajusta a exibição de casas decimais e formatos numéricos para FORMA.
help comando – mostra, na área de trabalho, o help do mando solicitado
who / **whos** – mostra as variáveis que estão atualmente na memória
clear – apaga uma ou mais variáveis da memória

2.2 – Comandos do sistema operacional

O MatLab possibilita a execução de alguns comandos do sistema operacional, através de comandos próprios.

dir – mostra o conteúdo do diretório corrente
what – mostra quais os nomes de arquivos presentes que tem a extensão .m e podem ser executados pelo MatLab
cd – muda o diretório corrente
type – mostra na área de trabalho o conteúdo de um arquivo

2.3 – Erros no MatLab

No MatLab os comandos e operações são avaliados e o resultado é posto na área de trabalho, caso o usuário digite um comando que não exista, ou uma expressão que não faça sentido para o MatLab, o mesmo envia SEMPRE para área de trabalho um *erro* devido a operação que o usuário executou.
Este erro é um comentário especificando o erro cometido e normalmente seguido da expressão digitada erroneamente pelo usuário. Alguns erros são comuns no MatLab e vale a pena lista-los:

```
??? Undefined function or variable xxx.
```
Este erro é causado devido a digitação de um nome de programa, função ou variável que não existe no MatLab, no caso de ser um programa o mesmo pode não estar no diretório corrente.

```
??? 23+4#
        |
Missing operator, comma, or semi-colon.
```

Este erro ocorre quando o usuário digita um símbolo ou operador que não faz sentido para o MatLab. Neste caso o MatLab aponta para onde está localizado o caracter inválido.

```
??? a+b*(2+5)-(3+5*(5-9)
                          |
A closing right parenthesis is missing.
Check for a missing ")" or a missing operator.
```

Este erro significa que algum parêntese está aberto errado ou está faltando na expressão como um todo.

```
??? Error using ==> sqrt
Too many input arguments.
```

Este erro se ocasiona devido ao uso indevido de uma função, o usuário chamou-a de forma indevida.

Alguns erros mais comuns com o uso de operadores de matrizes, e vetores são comuns, como a multiplicação e divisão de matrizes. O uso de operações com restrições como as matriciais devem ser executados com cuidado, observando sua precedência e validade. Erros como uso de funções com domínio restrito como sqrt, log, asin, acos etc... alguns apesar de não causar erros, podem comprometer o resultado final da operação.

2.4 – HELP

Todos os comandos do MatLab estão presentes no *help* do mesmo. todos os comandos possuem uma *sintaxe* que deve ser obedecida, e um número de parâmetros que deve ser utilizado. O comando “help comando” mostra na área de trabalho um *resumo* de como se deve operar com o comando especificado.

O help é muito útil para obtenção de informações sobre comandos que venham a ser utilizados durante um trabalho.

3 – Operações simples

O MatLab opera diretamente com números e matrizes bastando digita-los na área de trabalho ou em uma linha de programa. O MatLab distingue as operações dependendo to *tipo* de número que se queira usas (se for matriz, vetores, conjuntos, etc...)

3.1 – Operações [+ - / * ^ ‘]

As operações básicas que podem ser efetuadas diretamente no MatLab são soma, diferença, divisão, multiplicação e exponenciação e transposição, para isso usam-se os símbolos: + - / * ^ ‘ respectivamente. Algumas restrições devem ser lembradas quanto aos operadores divisão, multiplicação, e exponenciação, tipo: divisão por zero, zero como bases exponenciais negativas, ordem de matrizes em multiplicações e exponenciações.

No MatLab pode-se usar as operações básicas para operar em forma de conjunto, aplicando a operação à cada componente de um vetor ou matriz separadamente, bastando para isso usar um *ponto* antes do operador. Ex:

```
» a = [1 2 3; 4 5 6; 7 8 9]

a =

     1     2     3
     4     5     6
     7     8     9

» b = [9 8 7; 6 5 4; 3 2 1]

b =

     9     8     7
     6     5     4
     3     2     1

» a*b

ans =

    30    24    18
    84    69    54
   138   114    90

» a.*b

ans =

     9    16    21
    24    25    24
    21    16     9
```

Note que **a** e **b** são matrizes, mas a operação **a.*b** faz a multiplicação *elemento-por-elemento* das matrizes.

3.2 – Números no MatLab

No MatLab, os números são considerados valores que podem assumir forma real, complexa, matricial e vetorial (o MatLab entende um vetor como uma matriz coluna ou linha sendo portanto do mesmo tipo de matriz). Para cada *tipo* numérico existe uma forma de se usar as os operadores básicos e chamadas à funções.

Alguns exemplos de números no MatLab:

```
» real = 65.7896

real =

   65.7896

» complexo = 34.5 - 65.0i

complexo =

  34.5000 -65.0000i

» matriz = [1 2 3; 4 5 6; 7 8 9]

matriz =

     1     2     3
     4     5     6
     7     8     9

» vetor_linha = [1 2 3]

vetor_linha =

     1     2     3

» vetor_coluna = [1; 2; 3]

vetor_coluna =

     1
     2
     3
```

Observem que um vetor pode ser *linha* ou *coluna*.

OBS: uma matriz ou vetor podem ser compostos por elementos complexos e sua operação segue as mesmas regras de uma matriz real.

3.3 – Infinito, raiz de –1, indeterminações

O MatLab possue uma forma de tratar números *não reais* que aparecem em muitas situações, os mais comuns são: infinito, o complexo i, e as indeterminações. Para cada caso o MatLab possue uma forma de expressar e operar com esses números. Por exemplo, se operarmos com zeros em denominadores, logaritmos de zero, etc... obteremos uma resposta no MatLab seguinte:

```
» 1/0

Warning: Divide by zero

ans =

   Inf

» log(0)

Warning: Log of zero

ans =

  -Inf
```

Observem que em ambas as operações temos como resultado, um ***símbolo*** ao invés de um número (Inf) que significa que o resultado ***tende*** para um número muito grande (Inf).

Se operarmos com funções que dão resultados complexos, obtemos um número como o exemplo:

```
» sqrt(-23)

ans =

        0 + 4.7958i
```

um numero seguido de um ***i*** representa um número complexo no MatLab.

O MatLab também representa operações ***indeterminadas*** como 0/0, Inf*0. Quando temos um caso desses, obtemos um ***símbolo*** NaN que quer dizer (Not a Number) ex:

```
» 0/0

Warning: Divide by zero

ans =

   NaN
```

Dependendo do caso o MatLab informa que houve uma divisão por zero, para lembrar o usuário que a operação pode estar incorreta.

4 – Variáveis

A utilização do MatLab para resolver problemas numéricos, exige que sejam armazenados diversos valores para posterior consulta. Isto torna fácil a utilização do MatLab para resolver problemas que utilizem muitas operações com valores que podem *variar*. O MatLab pode armazenar números (reais, matrizes, etc...) em variáveis que ficam na memória e podem ser alterados e usados a qualquer hora.

4.1 – Oque é uma variável

Matematicamente uma variável é uma *letra* que pode assumir qualquer valor numérico. No MatLab, temos a mesma definição, sendo que alem disso, uma variável representa um espaço na memória do computador onde o valor é guardado.

4.2 – Nomes de variáveis

Um tópico muito importante é como deve ser posto os nomes das variáveis, existem algumas *regras* para os nomes das variáveis no MatLab, são listadas abaixo.

Regra 1 – Não se deve colocar nomes muito grandes em variáveis. Uma variável com até 10 caracteres está bom.
O limite do MatLab é 19 caracteres.

Regra 2 – O nome de uma variável **deve** começar com uma letra!

Regra 3 – O nome de uma variável só pode conter letras (Maiúsculas ou minúsculas) e números, o único caracter que pode existir fora letras e números é o *underscore* "_".

Regra 4 – Não é permitido caracteres como Espaço, virgula, ponto, acentos, parênteses, etc...

Exemplos de nomes de variáveis válidos e não-válido:

Abcdefghijklmnopqrstuvxz = 3.56 *(ERRADO mais de 19 caracteres)*
12_abc = [1 1 3] *(ERRADO não começa com letra)*
Termo1 = 1 + 2i *(CERTO)*
Coef_de_tempo = 0.45 *(CERTO)*
_ponto = 45.7 = *(ERRADO não começa com letra)*

OBS: No MatLab, letras maiúsculas são distinguidas de letras minúsculas, ou seja Mm é diferente de mM

4.3 – Tipos de variáveis

No MatLab não necessitamos de especificar um *tipo* de variável, no MatLab, uma variável pode ser de qualquer tipo, Matriz, real, complexa ou vetor.

4.4 – Operações com variáveis

As variáveis são operadas da mesma forma que os números, ou seja, os operadores básicos e chamadas de funções são feitas da mesma forma. Uma operação a mais que pode ser feita com uma variável é a operação atribuição, que cria ou atribui um valor qualquer a uma variável; Ex:

```
» a = [1 2 3]

a =

     1     2     3
```

Este comando cria uma variável chamada “a” e lhe atribui um valor igual a [1 2 3], ou seja, um vetor.

4.5 – A variável ans

No MatLab, temos uma variável *especial* cujos valores lhe são atribuídos automaticamente pelo próprio MatLab. Essa variável tem o nome de “ans” (que vem de answer) e seu valor é sempre igual ao valor da ultima operação efetuada **sem atribuição**. Ex:

```
» 5*(log(9)-45)

ans =

 -214.0139
```

4.6 – Manipulação

A manipulação de variáveis no MatLab é simples, existem alguns comandos e operações especiais para uso e manipulação de variáveis. A primeira delas é a atribuição, basta usas o sinal de igual entre o nome da variável e o valor, caso a variável não exista, ela é criada automaticamente. Para saber o valor de uma variável, basta digitar o nome da mesma, o MatLab irá imprimir o nome e o valor da mesma como no ex:

```
» retorno = 23.65 +5i

retorno =

  23.6500 + 5.0000i

» retorno

retorno =

  23.6500 + 5.0000i
```

Para mudar o valor de uma variável basta atribuir o valor à mesma.

Alguns comandos para manipulação de variáveis são bastante usados no MatLab:

Who – Mostra as variáveis que estão na memória
Whos – Mostra detalhes de cada variável
Clear – retira uma ou mais variáveis da memória.

5 – Vetores e Matrizes

Matrizes e vetores são *tipos numéricos* que requerem cuidados especiais, pois possuem restrições fortes nos operadores básicos. Por exemplo, não definimos a multiplicação de matrizes para qualquer ordem, alem do mais A*B é diferente de B*A se A e B são matrizes, alem operadores como * , / e ^ possuírem restrições de ordem.

5.1 – Vetores e Matrizes no MatLab

No MatLab, vetores e matrizes são interpretados da mesma forma, sendo que qualquer matriz linha ou coluna será interpretada como vetor ou matriz dependendo da operação, o MatLab faz isso automaticamente.

5.2 – Operações

Os operadores elementares usados com matrizes são os mesmos usados com números reais(+ - / *) adicionando-se o operador transposto ‘ que realiza a transposta da matriz bastando-se digitar a matriz (ou nome da variável) seguido do operador *aspa* ex:

```
» a = [1 2 3; 4 5 6]

a =

     1     2     3
     4     5     6

» a'

ans =

     1     4
     2     5
     3     6
```

para efetuar outras operações basta seguir os passos de uma operação real normal. ex:

```
» a=[1 2 3; 9 4 5; 3 5 3]

a =

     1     2     3
     9     4     5
     3     5     3

» a*inv(a)

ans =

    1.0000    0.0000    0.0000
    0.0000    1.0000    0.0000
    0.0000    0.0000    1.0000
```

Alem dos operadores básicos existe um *falso-operador* que serve para criar vetores ou matrizes que se constituem de uma seqüência de números, tal *falso-operador* é o ":". Por exemplo, para criar um vetor com os elementos 1 2 3 4 5 6 7 8 9 devemos prosseguir da seguinte forma: *primeiro_elemento*:*intervalo*:*ultimo_elemento* ex:

```
» 1:1:8

ans =

     1     2     3     4     5     6     7     8

» vetor = 3:0.5:10

vetor =

  Columns 1 through 7

    3.0000    3.5000    4.0000    4.5000    5.0000    5.5000    6.0000

  Columns 8 through 14

    6.5000    7.0000    7.5000    8.0000    8.5000    9.0000    9.5000

  Column 15

   10.0000
```

Como se vê acima, pode-se usar o operador atribuição para armazenar o vetor resultante um uma variável.

OBS.: Caso a matriz ou vetor possua muitos elementos ao ponto de não Ter espaço na tela, os elementos são mostrados em blocos, de colunas como é mostrado acima.

5.3 – Operações elemento-elemento

O MatLab possibilita a realização de uma operação especial quando temos matrizes ou vetores, é o tratamento das mesmas como *conjunto* e o operador age em cada elemento da matriz individualmente. Para isso basta colocar um ponto na frente do operador que se deseja efetuar a operação em conjunto. Ex:

```
» b = [1 2 3; 5 6 7; 7 8 9]

b =

     1     2     3
     5     6     7
     7     8     9

» a = [1 1 1; 2 2 2; 4 4 4]

a =

     1     1     1
     2     2     2
     4     4     4
```

```
» c = a.*b

c =

     1     2     3
    10    12    14
    28    32    36
```

veja que c=a.*b é diferente de c=a*b na primeira cada elemento da matriz c é o produto dos respectivos elementos de a e b.

OBS.: A operação soma e subtração obedece à operação por conjunto, portanto não faz sentido os operadores .+ e .-

5.4 – Operando com elementos

Cada elemento da uma matriz pode ser tratado como uma variável numérica simples (real ou complexa) bastando usar o nome da variável que guarda a matriz seguido de parênteses com os índices do elemento que se deseja usar separados por virgulas, se for um vetor, será apenas um índice. Ex:

```
» matriz = [1 2 3; 4 5 6; 7 8 9]

matriz =

     1     2     3
     4     5     6
     7     8     9

» x = matriz(2,2)

x =

     5

» y = matriz(1,2)

y =

     2

» vetor = [2 ; 4 ; 5]

vetor =

     2
     4
     5

» z = vetor(3) + vetor(1)

z =

     7
```

Pode-se usar o operador atribuição afim de mudar apenas um valor dos elementos da matriz ou vetor. Ex:

```
» matriz = [1 2 3; 4 5 6; 7 8 9]

matriz =

     1     2     3
     4     5     6
     7     8     9

» matriz(1,1)=0

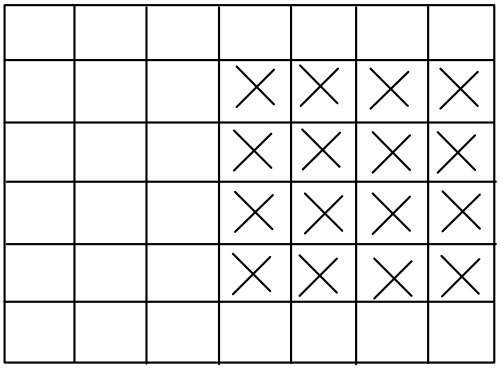
matriz =

     0     2     3
     4     5     6
     7     8     9
```

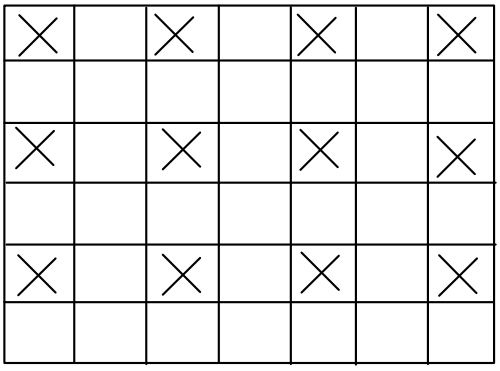
5.4.1 – Indexando mais de um elemento

No MatLab, podemos *acessar* um elemento de uma matriz através de seus índices (ex.: M(2,3) será o elemento da Segunda linha, terceira coluna). É possível também acessar mais de um elemento de uma matriz ou vetor, formando assim, *sub-matrizes* ou *sub-vetores* originários de vetores ou matrizes quaisquer. Para isso usa-se a marcação de “:” para designar a seqüência dos índices da matriz ou vetor. Segue algumas formas de marcação de seqüências.

M(2:5,4:7) = Elementos que estão compreendidos desde a linha 2 coluna 4 até a linha 5 coluna 7.



M(1:2:7,1:2:5) = Elementos *saltados* da matriz, contendo das linhas 1 até 7 e colunas 1 até 5 pulando de 2 em 2 elementos.



M(:,2) = Vetor linha correspondente a Segunda linha da matriz

M(2:4,:) = Matriz com 3 linha e *"n"* colunas formada pelas linhas 2, 3 e 4 da matriz original (*n* é o número de colunas de M)

5.5 – Operadores [ ]

Podemos usar os operadores [ ] para fazer a *junção* de matrizes, ou seja, se tivermos 2 matrizes 3 por 3 podemos obter uma matriz 6 por 3 ou 3 por 6 que é a junção das duas matrizes. Para isso colocamos dentro dos [ ] as matrizes ou nomes de variáveis que contem as matrizes menores separadas ou não por ponto e virgula ;. Ex:

```
» a = [1 0 0; 0 1 0; 0 0 1]

a =

     1     0     0
     0     1     0
     0     0     1

» b = [2 3 4; 3 4 2; 1 1 2]

b =

     2     3     4
     3     4     2
     1     1     2

» c = [ a b ]

c =

     1     0     0     2     3     4
     0     1     0     3     4     2
     0     0     1     1     1     2

» c2 = [a ; b]

c2 =

     1     0     0
     0     1     0
     0     0     1
     2     3     4
     3     4     2
     1     1     2
```

OBS.: Neste tipo de operação, deve-se tomar cuidado com as dimensões das matrizes. Não pode-se juntar uma matriz 2 por 2 com uma 3 por 3 por exemplo.

5.6 – Funções matriciais

Algumas funções são aplicadas para matrizes, algumas são descritas abaixo.

Zeros – cria uma matriz contendo 0 em todos os elementos.
ones(m,n) – cria uma matriz m por n contendo 1 em todos os elementos;
inv(M) – Retorna a inversa da matriz M;
size(M) – Retorna a dimensão da matriz M;
dot(V1,V2) – Produto escalar de V1 por V2;
cross(V1,V2) – Produto vetorial de V1 por V2;
trace(M) – Traço da matriz M (soma dos elementos na diagonal principal);
det(M) – Determinante da matriz M;
[A B] = eig(M) – Retorna em A os auto-vetores e em B os auto-valores de M

6 – Números complexos

O MatLab opera com números complexos com uma facilidade muito grande, para operar com números complexos basta usa-los como se fosse números normais, os operadores básicos são usados da mesma forma.

6.1 – Sqrt(-1)

No MatLab a unidade imaginária *"i"* que é representada por $\sqrt{-1}$ (no MatLab a função sqrt extrai a raiz de um número) é simplesmente a variável especial "i" ou "j".

6.2 – Operações

As operações básicas com números complexos é idêntica às operações com números reais. O usuário deve tomar cuidado apenas no uso de parênteses, quando se tratar de um número complexo com parte real e imaginária.

6.3 – Usando números complexos

Para usas um número complexo escreve-se a parte real, o operador de soma ou diferença seguido da parte imaginária junto com o "i" ou "j" **sem espaço**. Ex:

```
» a = 45.5 + 56.0i

a =

  45.5000 +56.0000i

» a = (1.1 - 4j)/4i

a =

  -1.0000 - 0.2750i
```

Como está mostrado, se o número tiver apenas parte imaginária, basta colocar o número seguido **sem espaço** de "i" ou "j".

6.4 – Algumas funções com números complexos

Abaixo vão algumas funções para uso com números complexos;

real(C) – Retorna a parte real de um número complexo C;
imag(C) – Retorna a parte imaginária de um número complexo C;
conj(C) – Retorna o conjugado de um número complexo C;
angle(C) – Retorna a fase de um número complexo C;
abs(C) – Retorna o módulo de um número complexo C;

7 – Funções básicas

Funções no MatLab são nomes que estão associados a alguma operação aritmética mais complicada, tipo: Raiz quadrada, logaritmos, funções trigonométricas, hiperbólicas, etc...

Existe uma diferença entre função e comando no MatLab, funções são procedimentos que retornam valores e comandos não retornam nada e geralmente não requerem parâmetros.

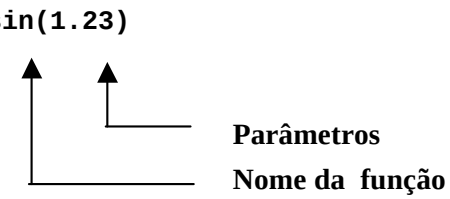
7.1 – Uso de funções

Funções são formadas por duas partes básicas: O nome da função e seus parâmetros. O nome de uma função obedece a mesma regra de formação de uma variável comum, o nome de uma função é seguido de parênteses onde dentro estão os parâmetros separados por virgulas. Ex:

```
» sin(1.23)

ans =

    0.9425
```



Uma função sempre retorna algum valor, podemos usar o operador de atribuição para atribuir este valor a uma variável. Ex:

```
» val = log(12.45)

val =

    2.5217
```

Neste caso foi criada uma variável com o nome de val que assumiu o valor do logaritmo natural de 12.45.

Funções podem ter vários parâmetros, e nem sempre estar associadas a funções matemáticas. Ex:

```
» M = ones(3,4)

M =

     1     1     1     1
     1     1     1     1
     1     1     1     1
```

A função ones recebe 2 parâmetros “m e n” que são as dimensões de uma matriz e cria uma matriz m por n preenchida com 1’s

7.2 – Funções que retornam mais de um valor

No MatLab uma função pode retornar mais de um valor. Neste caso fazemos a atribuição em uma matriz de variáveis como mostra exemplo abaixo:

```
» [x y] = meshgrid(1:3,1:3)

x =

     1     2     3
     1     2     3
     1     2     3


y =

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

7.3 – algumas funções básicas

São mostradas aqui algumas funções básicas que são muito usadas na solução de problemas.

log(x) – Logaritmo natural de x;
log10(x) – Logaritmo base 10 de x;
exp(x) – Exponencial de x;
sqrt(x) – Raiz quadrada de x;
log2(x) – Logaritmo base 2 de x;
pow2(x) – 2 elevado a potência x;
sin(x) – Seno de x (x em radianos);
cos(x) – Coseno de x (x em radianos);
tan(x) – Tangente de x (x em radianos);
sec(x) – Secante de x (x em radianos);
csc(x) – Cosecante de x (x em radianos);
sinh(x) – Seno hiperbólico de x;
cosh(x) – Coseno hiperbólico de x;
sech(x) – Secante hiperbólica de x;
csch(x) – Cosecante hiperbólica de x;
tanh(x) – Tangente hiperbólica de x;
ceil(x) – Arredonda x na direção +∞;
round(x) – Arredonda x para o inteiro mais próximo;
floor(x) – Arredonda x na direção -∞;
fix(x) - Arredonda x na direção de 0;
inv(X) – Inversa de X;
norm(X) – Norma de X;
eig(X) – Auto valores de X;
diag(X) – Cria uma matriz com diagonal X (X é um vetor);
det(X) – Determinante de X;
poly(X) – Coeficientes do polinômio característico de X;
real(x) – Parte real de x;
imag(x) – Parte imaginária de x;
angle(x) – Angulo de fase de x;
abs(x) – Módulo de x;
sum(X) – Suma de todos os elementos de X;
prod(X) – Produto de todos os elementos de X;

roots(X) – Raízes do polinômio X (X é u vetor com os coeficientes do polinômio);
FFT(X) – Transformada rápida de Fourier do vetor X;
IFFT(X) – Inversa da transformada rápida de Fourier do vetor X;
GAMMA(x) – Função Gama (GAMA(x)=X!{ x inteiro });
mim(X) - Menor elemento da matriz X;
max(X) - Maior elemento da matriz X;
spline(X,Y,XI) - Interpolação Spline da função Y=f(X) nos pontos XI;
interp1(X,Y,XI) - Interpolação simples da função Y=f(X) nos pontos XI;
sort(X) - Organiza os elementos do vetor ou matriz X em ordem crescente;
hist(X,N) – Desenha o histograma de X com N barras;
mediam(X) – Valor médio da matriz X;
bessel(alpha,X) – Função Bessel de X de ordem alpha;
linsolve(A,B) – Solução do sistemas linear AX = B;
rand(m,n) – Gera matriz aleatória m por n;

8 – Gráficos

Gráficos são muito importantes em problemas que envolvem cálculos com muitos dados e principalmente com dados discretos, onde temos vetores ou matrizes que representam medidas ou valores de pontos de funções. Um gráfico pode mostrar o comportamento de certo sistema muito melhor do que uma expressão calculada.

8.1 – Gráficos no MatLab

No MatLab, gráficos são *desenhos* de uma matriz ou vetor, que pode assumir várias formas, cores e estilos. A seguir veremos como usas gráficos no MatLab para plotar funções e mostrar graficamente uma matriz ou vetor.

8.2 – Tipos de gráficos

No MatLab, os gráficos são divididos em dois grupos; gráficos de vetores e gráficos de matrizes. Gráficos de vetores formam gráficos bidimensionais, e gráficos de matrizes formam gráficos tridimensionais. Cada tipo desses pode assumir diversos *estilos* que são as maneiras de se plotar o gráfico em questão. Gráficos bidimensionais dão formados por linhas, pontos, etc... e podem assumir várias cores, podendo também possuir uma grade, legendas etc... Gráficos tridimensionais são mais complexos e podem ser representados de várias formas; podem ser uma superfície sólida, *aramada*, *renderizada*, ou até mesmo uma representação em 2D com cores para diferenciar os valores de Z.

8.3 – Partes de um gráfico

Todo gráfico é representado na tela através de uma janela, onde nesta, estão *desenhadas* as diversas partes do gráfico. As mais importantes são:

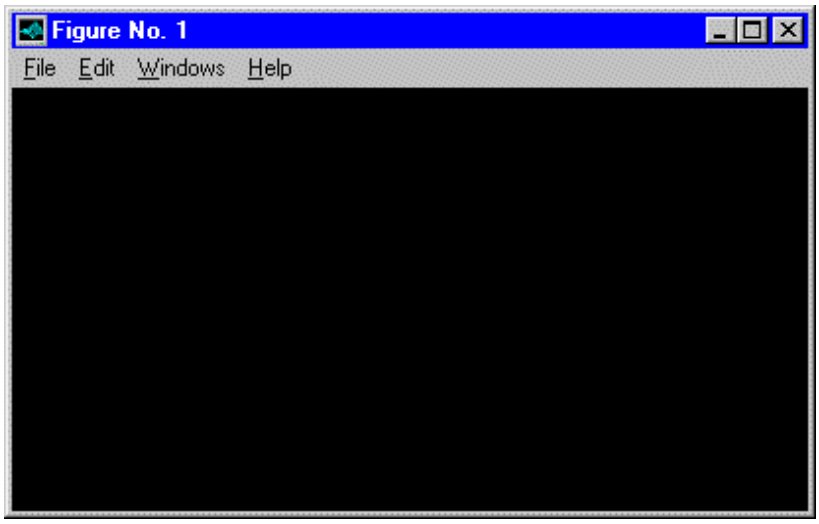
Título: É um texto situado acima do gráfico que contem alguma descrição do gráfico.

Nomes dos eixos: São os nomes das variáveis que os eixos X,Y,Z

Fundo: O fundo de um gráfico é a área da janela onde o gráfico e os seus componentes são desenhados.

Área de plotagem (Axes): É a área onde o vetor ou matriz é plotado(a).

Eixos (Axis): São os eixos coordenados que marcam os valores de índice ou valores de elementos da matriz/vetor.



Janela onde são plotados os gráficos

8.4 – Plotando Gráficos

Para plotar um gráfico, seja ele de vetor ou matriz, deve-se proceder da seguinte forma;

1 – Gere o vetor ou matriz a ser plotada.
2 – Escolha o tipo e o estilo de gráfico que deseja plotar.
3 – Use os comandos para plotar o gráfico
4 – *enfeite* o gráfico com titulo, grade etc...

Sempre que for necessário a construção de um gráfico deve-se observar essa seqüência pode-se adota-la também para a parte de programação. Todo comando que se relacione com gráficos é aplicado ao gráfico que estiver aberto, portanto, se o comando *grid* por exemplo for digitado, ele vai colocar a grade no gráfico que estiver plotado.

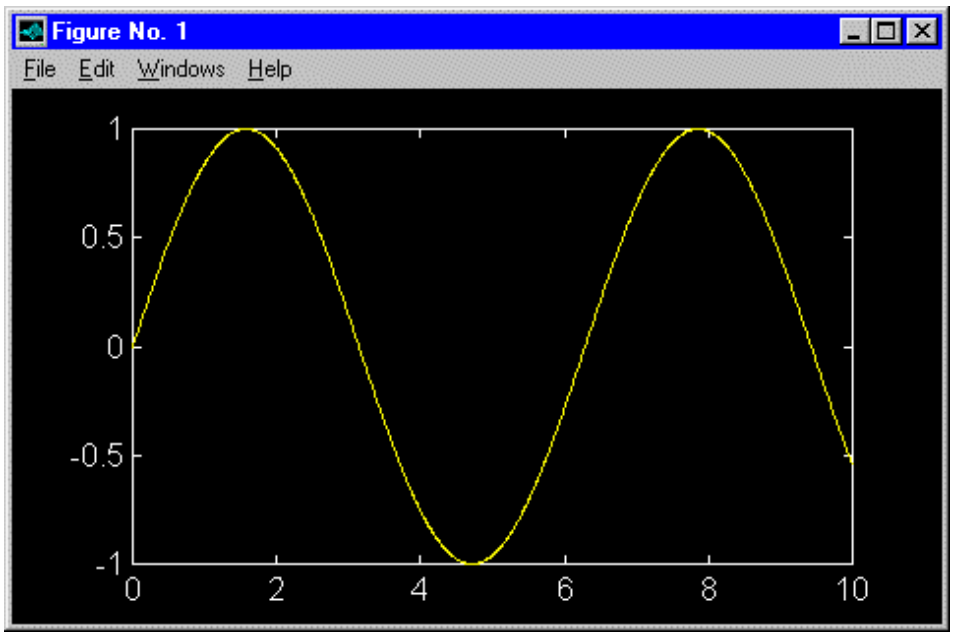
Podemos também plotar um gráfico por cima do outro, bastando usas o comando hold, para que o próximo gráfico se sobreponha ao anterior, digite **hold on** para que o próximo gráfico se desenhe apagando o anterior digite **hold off**.
OBS.: Após dar o comando *hold on* a escala fica mantida na escala do último gráfico desenhado, portanto os dois ou mais gráficos devem Ter mesma escala para uma melhor visualização.

8.5 – Gráficos 2D

Gráficos 2D são gráficos de vetores, a forma que o MatLab desenha esse vetor é a seguinte; No Eixo horizontal são colocado valores de um vetor de *n* elementos e no eixo vertical também valores de outro vetor de *n* elementos e para cada *par* de elementos dos vetores temos um ponto. Caso seja dado apenas um vetor, os valores dos mesmos serão os valores da abscissa do gráfico enquanto que o *índice* desse vetor comporá a ordenada. O comando para plotar gráficos bidimensionais é o comando plot;

Plot(X,Y); X e Y são vetores de *n* elementos onde os elementos de Y serão os valores das abscissas e os de X as ordenadas.

Pode-se usas o comando plot para plotar vários gráficos em uma mesma janela, bastando para isto usas: Plot(X1,Y1,X2,Y2,...)



Exemplo de um gráfico 2D

8.5.1 – Acabamento de um gráfico

Após desenhar o gráfico, passamos para o passo seguinte que é *enfeitar* o mesmo com títulos, nomes de eixos, legendas etc... Para isto usamos algumas funções que são descritas a seguir como exemplo:

title('titulo') - Coloca o título "titulo" na posição acima do gráfico.

grid on <off> - *grid on* mostra uma "grade" na janela do gráfico, *grid off* apaga a grade.

xlabel('nome do eixo X') - Coloca um nome para o eixo X do gráfico.

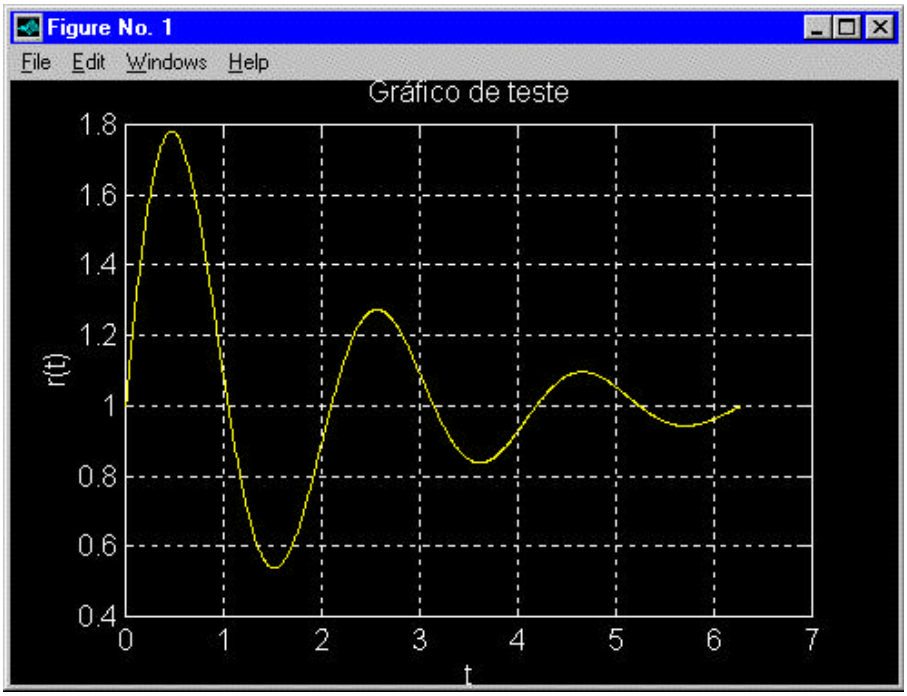
ylabel('nome do eixo Y') - Coloca um nome para o eixo Y do gráfico.

zlabel('nome do eixo Z') - Coloca um nome para o eixo Z do gráfico.

axis on <off> - *axis on* desenha os eixos coordenados, *axis off* apaga os eixos.

axis([xmin xmax ymin ymax zmin zmax]) – ajusta os limites das coordenadas da janela onde está o gráfico. X variando de xmin à xmax, o mesmo para Y e Z;

text(X,Y,'texto') - Coloca um texto na posição (X,Y) do gráfico. (caso o gráfico seja tridimensional use text(X,Y,Z,'texto');



Exemplo de gráfico com titulo e "labels"

8.6 – Gráficos 3D

Para desenhar um gráfico 3D temos de ter uma matriz m por n onde os elementos em m serão os valores de Z para os pontos X e n para os pontos Y. Imagine o plano cartesiano XY como uma matriz, onde cada elemento da matriz A(i,j) corresponde a uma altura Z para cada valor de X=i e Y=j.

Para plotar um gráfico 3D no MatLab temos de Ter essa matriz de alturas, as formas de gera-la podem ser as mais diversas. A maneira mais corriqueira é quando temos uma função de 2 variáveis Z=f(X,Y) e queremos plotar 0 gráfico da mesma. Para isso temos que criar uma matriz M por N onde M será o números de pontos em X e N o número de pontos em Y e cada elemento da A(i,j) da matriz será igual a altura correspondente ao X(i) e Y(j), quanto maior o número de elementos da matriz mais "definido" fica o gráfico. Ex:

Plotar o gráfico Z=exp(-x^2-y^2); com X variando de –3 a 3 e y variando de –3 a 3. Se escolhermos uma matriz 10 por 10 teremos o seguinte:

```
z =
   0.0000    0.0000    0.0000    0.0000    0.0001    0.0001    0.0001    0.0000    0.0000    0.0000
   0.0000    0.0000    0.0001    0.0007    0.0022    0.0032    0.0022    0.0007    0.0001    0.0000
   0.0000    0.0001    0.0015    0.0093    0.0273    0.0392    0.0273    0.0093    0.0015    0.0001
   0.0000    0.0007    0.0093    0.0561    0.1653    0.2369    0.1653    0.0561    0.0093    0.0007
   0.0001    0.0022    0.0273    0.1653    0.4868    0.6977    0.4868    0.1653    0.0273    0.0022
   0.0001    0.0032    0.0392    0.2369    0.6977    1.0000    0.6977    0.2369    0.0392    0.0032
   0.0001    0.0022    0.0273    0.1653    0.4868    0.6977    0.4868    0.1653    0.0273    0.0022
   0.0000    0.0007    0.0093    0.0561    0.1653    0.2369    0.1653    0.0561    0.0093    0.0007
   0.0000    0.0001    0.0015    0.0093    0.0273    0.0392    0.0273    0.0093    0.0015    0.0001
   0.0000    0.0000    0.0001    0.0007    0.0022    0.0032    0.0022    0.0007    0.0001    0.0000
```

Vejam a variação de X e Y de –3 a 3 com 10 elementos. Cada valor Z(i,j) corresponde a um f(x,y).

Para plotar a matriz Z podemos usar um dos comandos abaixo:

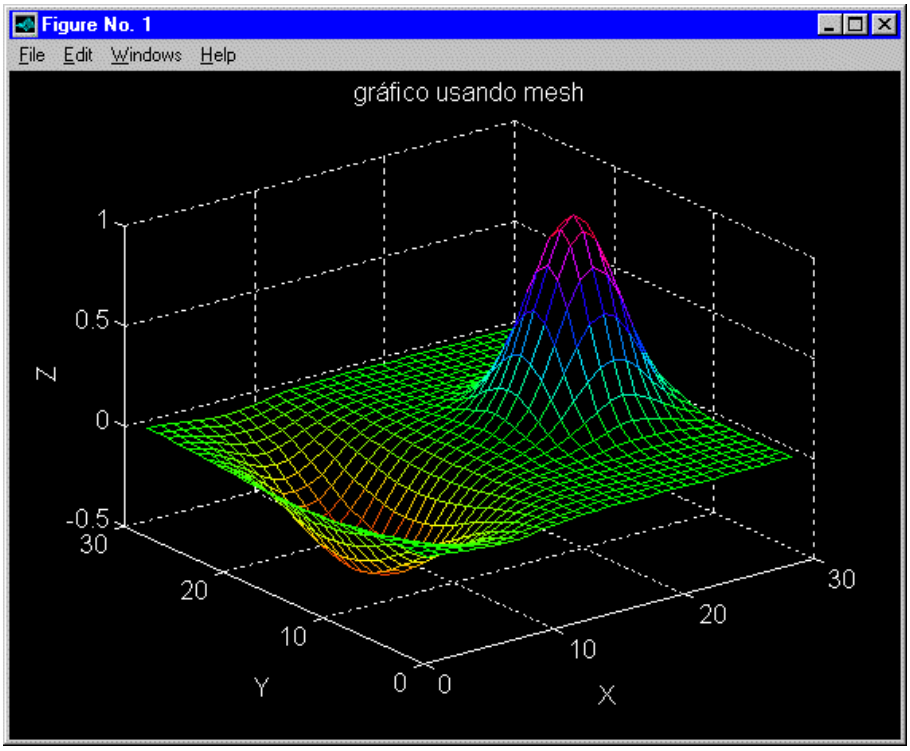
mesh, surf, contour (estes são os principais). Serão explicados abaixo.

MESH – O comando mesh permite plotar um gráfico 3D. O mesmo deve estar no formato de matriz como explicado acima. Tendo em mãos a matriz com os valores de Z usamos o comando mesh da seguinte forma;

```
mesh(Z);
```

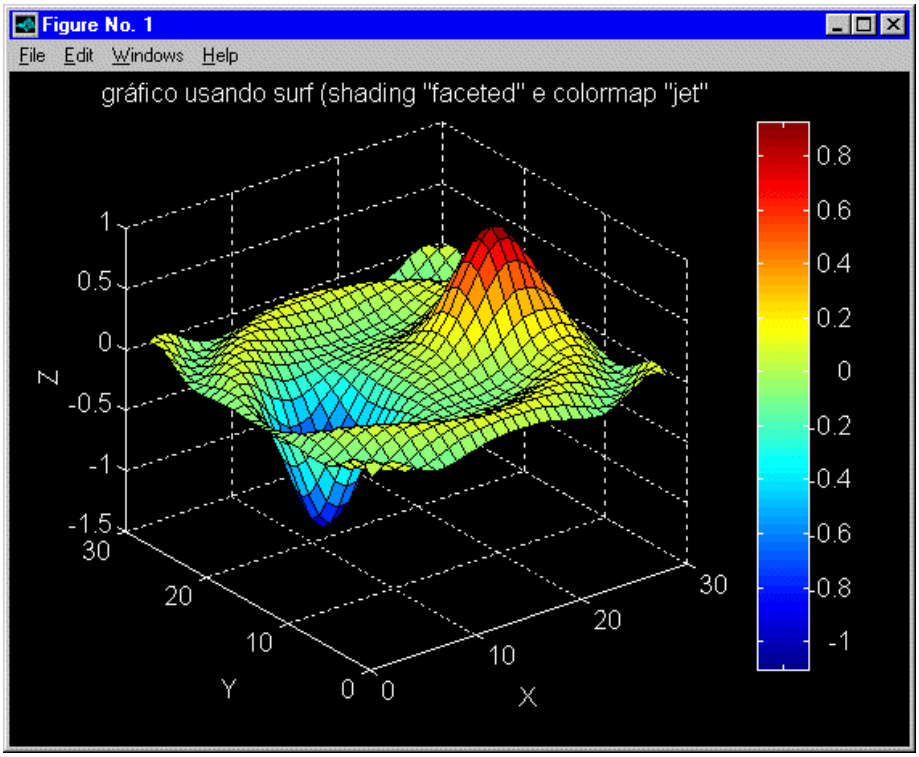
os eixos X e Y terão os valores dos índices da matriz e o gráfico será plotado com as cores proporcionais a altura do gráfico. Caso queiramos que os eixos X e Y tenham os valores de X e Y reais, deveremos ter uma matriz para X e uma para Y de tamanho igual a matriz Z sendo que com as linhas com valores iguais e iguais aos valores de X e Y. Para gerar tais matrizes, pode-se usar o comando ***meshgrid*** visto mais adiante. De posse da matriz X e Y basta usas o comando mesh(X,Y,Z); que será plotado o gráfico de Z com os eixos numerados conforme X e Y.

Podemos também usar uma matriz C, do mesmo tamanho de Z, que será as cores do gráfico 3D gerado por mesh. Cada elemento da matriz C(i,j), será uma cor que será usada para plotar o gráfico de Z.



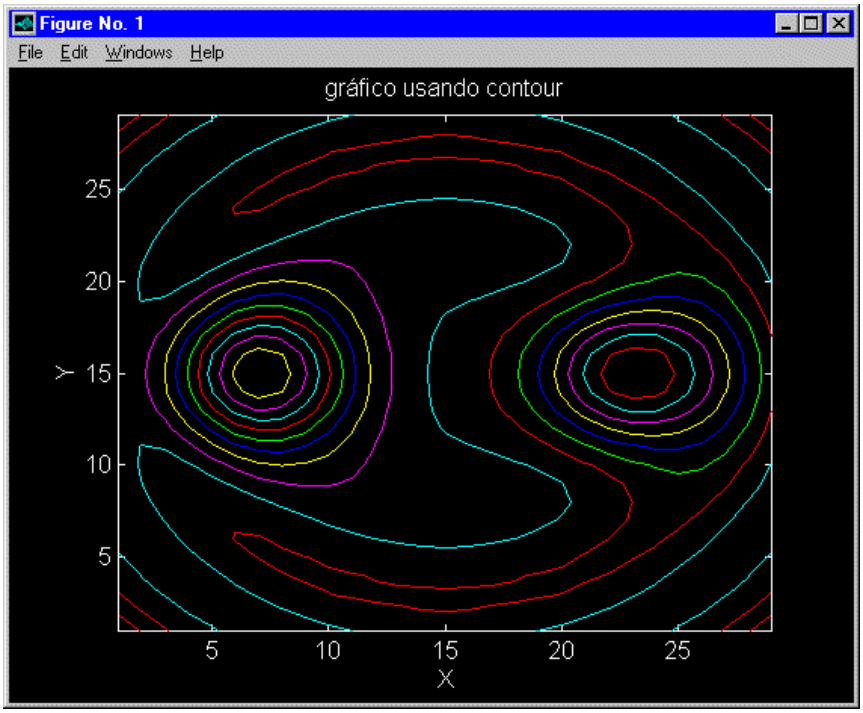
Exemplo de gráfico usando o comando *mesh*

SURF – O comando surf é exatamente igual ao comando mesh, sendo que este preenche as faces do gráfico de Z de forma a obtermos um gráfico "cheio". O tipo de preenchimento pode ser selecionado com o comando ***shading*** que será visto mais a frente.



Exemplo de gráfico usando *surf*

CONTOUR – O comando *contour* plota um conjunto de curvas correspondentes ao conjunto de equações f(x,y)=h onde h são valores de alturas especificas. O comando é usado da mesma forma que o comando mesh, contour(X,Y,Z) plota os contornos das curvas Z=h onde h é selecionado automaticamente pelo MatLab.



Exemplo de gráfico usando *contour*

Para gerar uma "grade" de X e Y para usar com os comandos mesh, contour e surf usamos o comando ***meshgrid***. Este comando gera duas matrizes X e Y contendo valores de intervalos dados. Ex:

Para formar uma grade com X=0:0.5:2 e Y=0:0.5:2 usamos ***meshgrid*** da seguinte forma:

```
» [X, Y] = meshgrid(0:0.5:2,0:0.5:2)

X =

         0    0.5000    1.0000    1.5000    2.0000
         0    0.5000    1.0000    1.5000    2.0000
         0    0.5000    1.0000    1.5000    2.0000
         0    0.5000    1.0000    1.5000    2.0000
         0    0.5000    1.0000    1.5000    2.0000
```

```
Y =

         0         0         0         0         0
    0.5000    0.5000    0.5000    0.5000    0.5000
    1.0000    1.0000    1.0000    1.0000    1.0000
    1.5000    1.5000    1.5000    1.5000    1.5000
    2.0000    2.0000    2.0000    2.0000    2.0000
```

Vejam que X e Y são matrizes 5 por 5, isto é necessário para aplicarmos a uma função 3D.

8.7 – Algumas funções gráficas

Aqui falaremos sobre algumas funções úteis para uso com gráficos. Serão apresentadas as funções e uma breve descrição.

semilogx(X,Y) - plota o gráfico XY com o eixo X aplicado ao logaritmo de base 10;
semilogy(X,Y) – mesmo que SemilogX, só que o eixo logaritmo é o Y;
loglog(X,Y) – mesmo que SemilogX e SemilogY só que aplica o logaritmo à ambos os eixos;
clf – apaga a janela onde está o gráfico atual;
hold on <off> - *Hold on* faz com que o próximo gráfico seja colocado ma mesma janela sem apagar o anterior. *Hold off* desliga essa opção;
meshgrid(ix,iy) – Gera as matrizes X e Y para construir uma matriz Z (xi e yi) são os intervalos;
surface(X,Y,Z) – Plota o gráfico de Z preenchido;
line(X,Y) – Coloca linhas ligando os pontos (xi,yi);
colormap(MAP) – Muda as cores de um gráfico 3D para as cores contidas em MAP (MAP e uma matriz 3x64 onde as linhas são os índices de cores e as colunas são as componentes R G B das cores;
shading *shade* – Muda o tipo de preenchimento para o tipo de *shade* (*shade* pode ser: Flat, Interp, Faceted)
quiver(X,Y,dX,dY) – Desenha setas em cada (xi,yi) apontando na direção de (dxi,dyi);
gradient(Z) – Retorna o gradiente de Z na forma de vetores (xi,yi);
view(x,y) – Ajusta a direção do observador para x e y graus de rotação e inclinação.
colorbar – coloca uma barra com o mapa de cores na janela do gráfico

8.8 – Gráficos avançados

Para o caso de um problema onde tenhamos que interpretar um gráfico de diversas formas, temos a possibilidade de plotar na mesma vários gráficos distribuídos em uma mesma janela. Para isso usamos alguns comandos mais avançados que serão mostrados abaixo.

8.8.1 – Plotando vários gráficos na mesma janela

É possível plotar vários gráficos em uma mesma janela sem que os mesmos estejam sobre os mesmos eixos e nas mesmas escalas. Alguns comandos possibilitam a divisão da janela em várias partes onde

cada parte podem ser colocados eixos coordenados (2D ou 3D) e um gráfico qualquer. Veremos esses comandos a seguir.

subplot – O comando *subplot* permite dividir a janela gráfica em subjanelas formando uma *matriz* de janelas (Ex. 2x2, 2x3, 3x3, etc...). S*ubplot* é usado da seguinte forma:

Subplot(m,n,p)

Onde "m x n" será configuração das subjanelas, e "p" será a *janela corrente* ou seja, onde o próximo gráfico irá ser plotado. "p" será a posição já subjanela contada em linhas da esquerda pra direita começando subjanela superior esquerda, como mostra a figura a baixo.

Figura de subplot

axes – O comando *axes* permite também dividir a janela em subjanelas só que este permite colocar a subjanela em qualuer posição dentro a janela gráfica corrente. Para colocarmos uma subjanela em uma posição (x,y) da janela usamos *axes* da seguinte forma:

Axes('position',[x y largura altura])

Onde *x* e *y* é o ponto superior esquerdo da subjanela, *largura* é a largura da janela e *altura* é a altura da mesma. Os parâmetros *x,y, largura e altura* são valores entre 0 e 1 onde (0,0) e (1,1) são os cantos superior esquerdo e inferior direito da janela principal, e 1 corresponde a largura total e altura total da janela principal também.

8.8.2 – Mudando cores e outras propriedades

Uma opção mais avançada na construção de gráficos é a possibilidade de modificar parâmetros como cor, posição, nome etc. de objetos na tela como gráficos, legendas, etc. Isto se faz com o uso de manipuladores, que são certas variáveis que controlam o comportamento de tais objetos. O comando para mudar tais propriedades é o comando *set* e o mesmo é usado da seguinte maneira:

SET(manipulador, propriedade, valor, propriedade, valor, ...)

Manipulador é a variável que armazena o objeto que você quer modificar, esta é criada a partir das funções que geram os objetos. *Propriedade* é o nome (sempre entre apóstrofos) da propriedade que deseja-se modificar, por exemplo, se quisermos mudar a cor de algum objeto a propriedade será 'color'. *Valor* é o valor (geralmente entre apóstrofos) da propriedade escolhida.

Ex.: se tivermos a seguinte linha de comando:

```
» H=plot(x,f)

H =

   57.0001
```

Embora H esteja com um valor numérico, H guarda consigo as propriedades do gráfico gerado por "plot" portanto se quisermos mudar a cor do gráfico para azul basta usas o comando: **SET(H,'color',blue');** Abaixo estão algumas funções que retornam manipuladores e que ,portanto, podem ser modificadas com o comando SET.

- PLOT
- AXES
- MESH
- TEXT

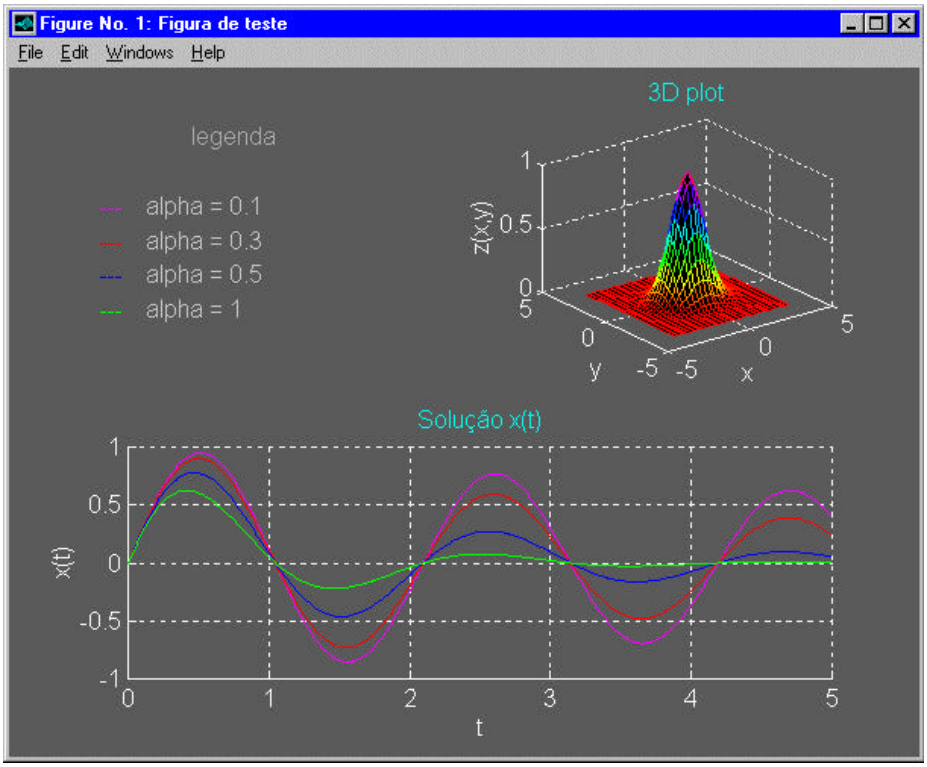
- SUBPLOT
- FIGURE

Alguns comandos podem Ter suas propriedades modificadas nos parâmetros do próprio comando. Para isso usando: COMANDO(parametro1, parametro2, ..., propriedade, valor, propriedade, valor, ...);
COMANDO é o nome do comando a ser usado, ***parametro1, 2...*** são os parâmetros normais do comando e o resto são os valores e propriedades a serem modificadas. Ex.:

```
» TITLE('Teste','color','blue')
```

8.8.3 – Usando o mouse para posicionar textos

Podemos usas o mouse para escolher pontos na janela gráfica onde iremos colocar determinados objetos. Isto se faz com a função GINPUT. O comando [x,y] = GINPUT(N) faz com que o cursor da janela gráfica torne-se uma crus e fique até o usuário clicar em N pontos



Exemplo de gráfico avançado

9 – Arquivos

9.1 – Uso de arquivos

Quando se trabalha com matrizes e vetores tem-se a necessidade de gravar os valores dessas matrizes e vetores em um arquivo. No MatLab pode-se gravar em disco em alguns formatos específicos. Quando se salva um vetor ou matriz no MatLab o mesmo fica gravado em um arquivo .mat no diretório corrente, este arquivo contem o nome da variável e seu valor (matricial) e pode ser carregada a qualquer hora pelo usuário. Se a matriz for salva em um arquivo de texto (veja opções do comando save) o nome da variável será o nome do arquivo que a contem. O conteúdo do arquivo de texto gerado são os valores dos elementos da matriz igualmente separados por espaços e quebras de linha (*ENTER's*).

O comando para salvar uma matriz é o comando *save* e para carrega-las *load*.

9.2 – Comandos de arquivos

Os principais comandos para uso com arquivos são mostrados a seguir, incluindo a forma de como usa-los.

Save fname X –option - Este comando salva a matriz X no arquivo fname com as opções "-option"
-option = -ascii (salva em formato texto)
-option = –double (salva com dupla precisão)
-option = –tabs (salva com tabs para separar os valores)
Se não for colocado –option o MatLab salva no formato binário e adiciona a extensão .mat ao nome do arquivo.

load fname - Carrega o arquivo fname e restaura a variável da matriz que está gravada. (se fname for texto o nome da variável será *fname*)

diary fname - Salva o conteúdo da área de trabalho do MatLab no arquivo fname.

10 – Programação

O MatLab possue um interpretador de comandos que permite que usemos seus comandos de forma seqüencial e com blocos de comparação e *loops* formando assim um programa que usa os comandos do próprio MatLab. Esta possibilidade de se programar no MatLab é muito usada para construir programas destinados a estender os usos do próprio MatLab.

A "linguagem" do MatLab é interpretada, ou seja, o usuário precisa do MatLab para rodar os programas, os erros só aparecerão quando o programa estiver rodando e os comandos são executados em seqüência.

10.1 – Funções lógicas e operadores lógicos

Antes de começar a construir um programa no MatLab, vamos dar uma olhada em um tipo de função e um tipo de valor muito usado em programação; as funções e valores lógicos.

Um valor lógico (muitas vezes chamado de *boolean*) pode assumir apenas 2 estados: *true* e *false* (verdadeiro ou falso) e uma função lógica é aquela que retorna um valor lógico. No MatLab, valores lógicos são expressos em termos de 1 e 0; 1 para *true* e 0 para *false*, portanto uma função lógica retorna 1 ou 0.

Abaixo veremos algumas funções lógicas no MatLab que são muito utilizadas na construção de programas.

exist('V') – Verifica se a variável ou função V está definida.

any(M) – Retorna 1 se algum elemento da matriz ou vetor M for *true*, 0 caso contrário.

all(M) – Retorna 1 se todos os elementos da matriz ou vetor M são *true*, 0 caso contrário.

isnan(X) – Retorna uma matriz de mesmo tamanho de X com 1's nas posições onde tive NaN's e 0's onde não tiver NaN's.

isinf(X) - Retorna uma matriz de mesmo tamanho de X com 1's nas posições onde tive NaN's e 0's onde não tiver NaN's.

finite(X) - Retorna uma matriz de mesmo tamanho de X com 1's nas posições onde tive NaN's e 0's onde não tiver NaN's.

isempty(M) – retorna 1 se a matriz M está vazia, 0 caso contrário.

isreal(M) – Retorna 1 se todos os elementos da matriz são reais, 0 caso contrário.

issparse(M) – Retorna 1 se a matriz é esparsa, 0 caso contrário.

isstr(X) – Retorna 1 se X é uma string, 0 caso contrário.

isglobal('v') – Retorna 1 se v é uma variável global, 0 caso contrário.

Paralelo as funções lógicas, o MatLab possue operadores lógicos, que retornam apenas 1 ou 0 como resultado da operação realizada pelo usuário. Um operador lógico tem mesma forma es estrutura de um operador qualquer e seu uso se faz da mesma forma: valor *operador* valor.

Os operadores lógicos existentes no MatLab são:

A **==** B - Comparação de igualdade. Retorna 1 se A for igual a B, 0 caso contrário. Neste caso A e B são variáveis comuns (numéricas).

A **<** B – Operador de menoridade. Retorna 1 se A for menor que B, 0 caso contrário. Neste caso A e B são variáveis comuns (numéricas).

A **>** B – Operador de maioridade. Retorna 1 se A forma maior que B, 0 caso contrário. Neste caso A e B são variáveis comuns (numéricas).

A **&** B – AND lógico. 1 somente se A e B forem 1's, 0 caso contrário. Neste caso A e B são variáveis lógicas.

A **|** B – OR lógico. 0 somente se A e B forem 0's, 1 caso contrário. Neste caso A e B são variáveis lógicas.

~A – NOT lógico. Retorna o complementar de A. 1 se A for 0 e 0 se A for 1. Neste caso A é uma variáveis lógicas.

A **~=** B – Diferente. Retorna 1 se A for diferente de B e 0 caso contrário.

xor(A,B) – XOR lógico. Retorna 1 se A for diferente de B, 0 caso contrário. Neste caso A e B são variáveis lógicas.

10.2 – Noções de algoritmos

Um algoritmo é um "esquema" de um programa que se deseje construir. É como se fosse o manual de instruções de construção de determinado programa. Um algoritmo tem a aparência de um programa só que o mesmo está indicando passos à serem realizados para o programador, passos estes que se resolvem qualquer problema de mesma espécie do que se trata o programa que irá ser construído.

Para construir programas no MatLab não necessariamente teremos que construir o algoritmo primeiro, mas é sempre bom ter em mente um esquema de como será o programa final antes de começar a faze-lo.

Algoritmos podem ser também diagramas de blocos em seqüência de forma que o programador separe os blocos como partes de pequenos programas onde as saídas de um bloco sejam usadas pelo próximo ou no final do programa para calculo de um resultado final.

OBS.: Nem todo programa se destina a calcular alguma coisa, sempre que for mencionado o cálculo de alguma grandeza, podemos interpretar como execução de um passo qualquer.

10.2.1 – Estrutura básica de um programa

A estrutura básica de um programa é um conjunto de instruções (em forma de algoritmo) que praticamente todo programa deve ter se quer realizar algo útil. Uma estrutura básica de programas feitos em MatLab está sugerida a seguir:

Inicio Declaração e inicialização das variáveis usadas no programa
Entrada de valores do usuário, parâmetros do problema que estão em forma de variável.
Programa em sí. Cálculos e ações que deverão ser executadas pelo programa Atribuição dos resultados à variáveis de saída do programa.
Exposição dos resultados ao usuário. Fim

Como vemos, a maioria dos programas tem a mesma seqüência de operação e basicamente trabalha em cima de valores de entrada e valores de saída. Normalmente um programa é um elemento que toma valores de certos parâmetros de entrada e gera outros valores como saída. Oque difere um programa de outro são os parâmetros de entrada e os de saída ou ainda a forma como os parâmetros de saída são calculados.

10.2.2 – Regras de construção

Alguns cuidados devem ser tomados quando se constrói um programa em qualquer linguagem. São uma espécie de “regras” que se seguidas ajudam para que o programa não se torne difícil de ser interpretado e corrigido e além do mais torna o programa mais organizado, sendo mais provável o sucesso em poucas tentativas de execução após construído.

Abaixo estão algumas dessas regras que por experiência própria são as maiores causadoras de problemas na construção de programas em qualquer linguagem e de qualquer tipo.

- Procure nomes de variáveis nem muito grandes e nem muito pequenos
- Observe o tipo de parâmetro quando chamar uma função

- Cuidado com a diferença entre operações booleanas e aritméticas
- Cuidado com expressões muito grandes
- Faça o programa por partes
- Não altere valores de variáveis de controle de *loop*
- Padronize sua própria estrutura de programas
- Não economize espaço!!!
- Preste atenção no tipo da variável antes de operar com ela.
- Vá testando o programa a medida que o constrói
- Comente o máximo de linhas possíveis no programa
- **SAIBA MUITO BEM SOBRE O PROBLEMA!!!**

10.3 – A linguagem do MatLab

O MatLab permite que se grave um arquivo (com extensão .m) que contenha comandos que o usuário queira que sejam executados em seqüência pelo próprio MatLab. Além disso o MatLab dispõe de alguns comandos de programação que possibilitam que seja construído um programa completo.

Um programa no MatLab é um arquivo de texto com extensão .m, contendo os comandos que o programador queira executar. Este arquivo deve estar no diretório corrente ou nos diretórios que o MatLab procura em *path* especificado.

Para rodar um programa no MatLab, certifique-se que o mesmo está no diretório corrente ou que o *path* do MatLab esteja observando o diretório que o mesmo se encontra. Eito isso basta digitar o nome do arquivo que o programa foi salvo.

No MatLab temos dois comandos que servem de entrada e saída de dados muito utilizados nos programas em geral; são os comandos **input** e **disp** a forma de utiliza-los está mostrada abaixo.

V = input('prompt') – Este comando faz com que o MatLab pare a execução do programa, mostre a mensagem 'prompt' e espere um valor ser digitado para então continuar a execução do programa. O valor digitado será armazenado na variável "V".

disp('mensagem qualquer') – Este comando simplesmente mostra na área de trabalho a mensagem 'mensagem qualquer' ou se no lugar de 'mensagem qualquer' for uma variável (disp(A) por exemplo) o valor dessa variável será mostrado na área de trabalho.

10.3.1 – A variável PATH

Como foi mencionado o MatLab possue um *path* que contém diversos diretórios para os quais o MatLab executa programas .m mesmo sem ter que estar com os mesmos como diretório corrente. O MatLab possue uma variável "escondida" (que não aparece no comando WHO) chamada PATH. Esta variável contém uma lista de diretórios que são o *path* do MatLab. Para Adicionar um diretório no *path* do MatLab, o usuário não deve alterar o valor da variável simplesmente com o operador de atribuição, ao invés disso usa-se o "comando" path. Como segue:

path(P) – Muda o *path* do MatLab para o *path* contido na string P.

path(PATH,P) – Adiciona o *path* contido na string P ao *path* do MatLab.

10.4 – Loops, Condicionais, e comandos de programação

Como já foi dito, o MatLab possue comandos de controle de luxo de programa que podem ser usados na construção dos mesmos. Além disso, tem a possibilidade de se programar funções, ou seja, programas que podem receber retornar valores e serem usados como unções normais.

Esses comandos quando estão presentes nos programas são chamados de blocos controladores de luxo de programa.

10.4.1 – If

O primeiro e mais importante dos comandos de controle de luxo de programa é o **if**. O **if** executa um bloco de programa ou outro dependendo do valor de uma variável lógica.

Seu uso tem a seguinte forma:

```
if variavel_logica1
      comando1;
      comando1;
          .
          .
elseif variavel_logica2
      comando2;
      comando2;
          .
          .
elseif variavel_logica3
      comando3;
      comando3;
          .
          .
else
      comando4;
      comando4;
          .
          .
end
```

O funcionamento é o seguinte:

O programa ao chegar no **if**, verifica o valor de variavel_logica1, se for verdadeira (true) o programa executa a seqüência de comandos1; se for falsa (false) ele passa direto para o **elseif** e testa o valor de variavel_logica2, se for verdadeiro executa a seqüência de comandos2; se for falsa passa para o próximo **elseif** e assim por diante, até que quando não houver mais **elseif**'s e nenhuma variavel_logica tiver sido verdadeira o programa finalmente executa a seqüência de comandos correspondente ao **else**, ou seja: comandos4;

Em uma sentença **if** não é preciso que sempre tenha blocos **elseif** ou **else**. Um bloco pode Ter apenas **if** e **end**.

Variavel_logica deve ser alguma expressão ou unção que retorne 0 ou 1, ou seja, um valor lógico.

Exemplo do uso de um bloco **if**

```
if a == b
   M(a,a) = 0;
else
   M(a,b) = 1;
end
```

10.4.2 – For

Um dos blocos de controle de luxo mais usados também é o bloco **for.** Um bloco **for** também é chamado de *loop incondicional* ou seja um bloco que fica se repetindo incondicionalmente até que se chegue no limite de uma variável de controle. Seu uso tem a seguinte forma:

```
for(A = valor_inicial:passo:valor_final)
   comandos1;
       .
       .
end
```

O funcionamento de um *loop* for é simples. Inicialmente a variável "A" terá valor igual a *valor_inicial* e a seqüência de comandos1; serão executadas até que a variável "A" tenha valor igual a *valor_final*. E a cada vez que a seqüência for executada a variável de controle "A" terá seu valor incrementado por *passo*. Se *passo* não for indicado será assumido passo=1;

Valor_inicial, passo, e valor_final podem ser números ou variáveis com valores numéricos.

Exemplo de um bloco **for**

```
for(a=0:0.1:V_fin)
  x=w*t;
  A=sin(x+phi);
end
```

10.4.3 – While

Outro bloco de controle de fluxo que pertence a categoria dos *loops* é o bloco while. O bloco while é conhecido como *loop condicional* por ser um bloco de *loop* que usa variáveis lógicas para seu controle. O uso do bloco while é simples:

```
while variavel_logica
   comandos1;
      .
      .
      .
end
```

Seu funcionamento é o extremamente simples, simplesmente a seqüência de comandos1; será executada até que variavel_logica tenha valor false.
Obs.: O controle dos valores da variável variavel_logica não é feito automaticamente como no *loop* for, portanto ao se trabalhar com o bloco while deve-se lembrar de controlar o valor de variavel_logica, afim de não entrar em um *loop infinito*.
varialvel_logica pode ser uma expressão ou função que retorne um valor lógico.

Exemplo de um *loop* while:

```
while cont<100
   cont=cont+1;
   b=f_aproc(cont);
  if b<0 cont=0;
end
```

10.4.4 – Definindo funções

O MatLab permite que façamos nossas próprias funções, ou seja, programas .m que aceitam parâmetros e retornam valores (estas funções são normalmente denominadas de funções definidas pelo usuário). Para isto utilizamos a palavra chave **function**. Para que um programa aceite valores como argumento e retorne um valor o mesmo deve ter uma estrutura particular como se segue.

```
function [saida1, saida2] = nome_da_funcao(arg1, arg2, arg3)

     comandos do programa e cálculos realizados.
            .           .
            .           .
            .           .
            .           .

saida1 = valor1;
saida2 = valor2;

end
```

Estrutura de uma função construída pelo usuário

saida1 e *aida2* são os valores que serão usados como saída da função, podemos usas quantas saídas forem necessárias. *arg1, arg2* e *arg3* são os parâmetros de entrada da função e podem ser matrizes, números complexos etc... assim como os argumentos de saída.
Uma condição muito importante para o uso de funções definidas é que o nome do arquivo seja igual ao nome da função. Por exemplo: o nome do arquivo do esquema mostrado acima deveria ser: *nome_da_funcao.m*
Quando se chama uma função não é necessário chama-la com todos os argumentos de entrada e saída. Por esse motivo o MatLab possue duas funções que são próprias para uso com funções definidas pelo usuário. As funções **nargin** e **nargout**. nargin retorna o numero de argumentos de entrada usados na chamada a função e nargout retorna o número de argumentos de saída. Ex:

[S1, A] = func(b,1.0,'casa'); - nesta chamada à função func, nargin retornará 3 e nargout retornará 2.

10.4.4.1 – Variáveis globais

Quando se constrói uma função definida pelo usuário, as variáveis que são criadas só tem validade, ou seja, só são "vistas" dentro do *escopo* da função que estamos definindo. A palavra escopo representa o bloco ou arquivo em que a função está definida, portanto se uma variável só é vista em determinado *escopo* significa que a mesma é criada quando se executa o bloco correspondente e em seguida é removida da memória.

Se o usuário quiser compartilhar variáveis de uma função com outras ou mesmo com o próprio MatLab, as mesmas devem ser "declaradas como globais" ou seja, definidas antes de serem usadas. Para isto usamos a palavra chave *global* como mostrado a seguir.

global V – Declara a variável V como global, ou seja, pode ser usada em todas as funções.

Para compartilhar variáveis entre funções e com o próprio MatLab as mesmas devem ser declaradas como globais tanto no MatLab como na função que irá usar-las.

10.5 – Considerações importantes sobre programação

Programas de computador em geral, possuem forma e organização semelhantes. Ao se construir um programa para resolver determinada tarefa o mesmo geralmente não assume formas muito diferentes das formas apresentadas na maioria dos casos. Na construção de um programa podemos ter blocos de controle de fluxo encadeados, ou seja, um for dentro de um if, um while dentro de um for ou até mesmo um bloco dentro de um bloco de mesmo tipo (for dentro de for por exemplo). Nestes casos a seqüência de execução é sempre obedecida, se tivermos por exemplo um for dentro de outro for, para cada iteração do primeiro for teremos todas as iterações do for mais interno e assim por diante.

Uma consideração importante são os tipos de variáveis, como o MatLab é uma ferramenta que mostra uma facilidade muito grande em trabalhar com matrizes, o programador deve tomar cuidado com os tipos de dados que está usando em certas circunstancias como controle *loops*, contadores e quando se usa expressões com operadores que diferem para matrizes como multiplicação e divisão.

11 – ToolBoxes

Devido a grande utilidade do MatLab nas diversas áreas de estudo, o mesmo possue um possibilidade de *expansão* para uso com áreas mais específicas. Esta *expansão* se da através de ToolBoxes. Existem ToolBoxes para as mais diversas áreas da ciência, desde as engenharias até a manipulação simbólica de expressões, muito utilizada ma Matemática.

11.1 – Oque são ToolBoxes

ToolBoxes são um conjunto de arquivos .m e .mat (chamados de *scripts* e *kernels*) que são acrescentados ao MatLab e são específicos para realização de uma determinada tarefa ou solucionar determinado problema. Existem ToolBoxes com programas para as mais diversas áreas como já foi citado. Um ToolBox quando *instalado* no MatLab fica sob forma de um diretório que contem vários arquivos destinados a realizar as operações à que se destina o ToolBox. A instalação de ToolBoxes é normalmente diferente para cada caso, dependendo o fabricante do ToolBox.

11.2 – Exemplos

11.2.1 – ToolBox Simbolic

O ToolBox *simbolic* é um ToolBox fabricado pela própria MathWorks (fabricante do MatLab) e tem por finalidade operar com variáveis literais. O ToolBox *simbolic* implementa diversos comandos que são mostrados e explicados a seguir. Antes falaremos sobre como o ToolBox *simbolic* trata variáveis simbólicas.

Uma variável simbólica é uma *matriz de caracteres* formando uma *string* contendo a expressão que a variável representa. O valor de uma variável *string* é posto sempre entre aspas, como mostra o exemplo abaixo:

```
» poli = 'x^2+3*y+4'

poli =

x^2+3*y+4
```

Alguns comandos principais do ToolBox *simbolic* são mostrados a seguir:

eval(S) - Faz com que a string S seja interpretada como comandos do MatLab; Ex.:

```
» eval('x=5; x^2+2')

ans =

    27
```

solve(S,X) – Faz com que o MatLab resolva a equação S para a variável X. Ex.:

```
» solve('B*x^2+A*x=0')

ans =

[   0]
[-A/B]
```

dsolve(S,C) – Resolve a equação diferencial S com as condições iniciais C. Ex.:

```
» dsolve('A*D2x+C*x=x','x(0)=1','Dx(0)=0')

ans =

cos((C-1)^(1/2)/A^(1/2)*t)
```

sym(S) – Gera uma matriz simbólica representada por S. Ex.:

```
» sym('[a b c; a^2 b^2 c^2; 1/a 1/b 1/c]')

ans =

[   a,  b,  c]
[ a^2,b^2,c^2]
[ 1/a,1/b,1/c]
```

symop(S1,O,S2) – Realiza a operação O entre os operandos S1 e S2. Ex.:

```
» symop('sin(x)*exp(-x)','/','y^2+3*x')

ans =

sin(x)*exp(-x)/(y^2+3*x)
```

simplify(S) – Simplifica a expressão S. Ex.:

```
» simplify('(x^2+x-6)/(x+3)')

ans =

x-2
```

int(S,X,a,b) – Integra a expressão S na variável X no intervalo [a,b]. Ex.:

```
» int('exp(-x^2)','x',-Inf,Inf)

ans =

pi^(1/2)
```

diff(S,X,n) – Deriva n vezes a expressão S com respeito a X. Ex.:

```
EDU» diff('1/cos(x)^2','x',3)

ans =

24/cos(x)^5*sin(x)^3+16/cos(x)^3*sin(x)
```

expand(S) – Expande a expressão S. Ex.:

```
» expand('(x-3)*(x+2)*(x-2)')

ans =

x^3-3*x^2-4*x+12
```

transpose(S) – Transpõe a matriz simbólica S. Ex.:

```
» transpose(sym('[a b c; a^2 b^2 c^2; a^3 b^3 c^3]'))

ans =

[a, a^2, a^3]
[b, b^2, b^3]
[c, c^2, c^3]
```

factor(S) – Fatora a expressão S. Ex.:

```
» factor('x^2+2*x+1')

ans =

(x+1)^2
```

laplace(S,'x') – Acha a transformada de Laplace de S na variável x. Ex.:

```
» laplace('sin(t)*t','t')

ans =

2/(t^2+1)^2*t
```

fourier(S,'x') – Acha a transformada de Fourier de S na variável x. Ex.:

```
» fourier('exp(-t^2)','t')

ans =

pi^(1/2)*exp(-1/4*t^2)
```

invfourier(S,'x') – Acha a transformada inversa de Fourier de S na variável x. Ex.:

```
» invfourier('pi^(1/2)*exp(-1/4*t^2)','t')

ans =

exp(-t^2)
```

invlaplace(S,'x') – Acha a transformada inversa de Laplace na variável x. Ex.:

```
» invlaplace('1/(s^2+4*s+2)')

ans =

1/2*exp(-2*t)*sinh(2^(1/2)*t)*2^(1/2)
```

pretty(S) – Imprime na tela a expressão S de forma “bonitinha”. Ex.:

```
 » pretty('(x^2+3*x+3)/(sin(x)^2+cos(x)^(4^x))')

                                  2
                                 x  + 3 x + 3
                               --------------------
                                                 x
                                2          (4 )
                               sin(x)  + cos(x)
```

finvert(S,’x’) – Acha a inversa de S na variável x e avisa se a inversa não é única. Ex.:

```
» finverse('x^2','x')
Warning: finverse(x^2) is not unique

ans =

x^(1/2)
```

11.2.2 – ToolBox Sistemas Controle

Um ToolBox também muito usado é o ToolBox de sistemas de controle. Ele possue muitas funções que trabalham com sistemas de controle, de forma a possibilitar representações em espaço de estados, funções de transferência, matrizes de observabilidade, controlabilidade, zeros, etc... Existem muitos comandos. Citaremos alguns logo mais.

bode(num,den) – Plota a curva de BODE para o polinômio num(s)/den(s). Num e dem são vetores com os coeficientes dos polinômios na forma não fatorada. Ex.:

```
» bode([2 3],[1 2 4])
```

impluse(num,den) – Plota a resposta ao impulso de um sistema cuja função de transferência é num(s)/den(s). Num e dem são vetores com os coeficientes dos polinômios na forma não fatorada. Ex.:

```
» impulse([2 3],[2 1 3])
```

step(num,den) – Plota a resposta ao degrau de um sistema cuja função de transferência é num(s)/den(s). Num e dem são vetores com os coeficientes dos polinômios na forma não fatorada. Ex.:

```
» step([2 3],[2 2 3])
```

[num den] = ss2tf(A,B,C,D) – Converte a representação em espaço de estados (Matrizes A,B,C e D) em representação por função de transferência: num(s)/den(s)

```
» [nem den] = ss2tf([-2/3 1/3; 1 0 ] ,[ 1; 0], [1/3 2/3] , 0)

nem =

         0    0.3333    0.6667
```

```
den =

    1.0000    0.6667   -0.3333
```

[z p G] = ss2zp(A,B,C,D) – Converte a representação em espaço de estados (Matrizes A,B,C e D) na representação de pólos e zeros onde z e p são vetores contendo as raízes do numerador de denominador respectivamente e G o ganho do sistema. Ex.:

```
» [z p G] = ss2zp([-2/3 1/3 ; 1 0 ] , [ 1 ; 0], [1/3 2/3] , 0)

z =

    -2


p =

   -1.0000
    0.3333


G =

    0.3333
```

[A B C D] = tf2ss(num,den) – Transforma a representação por função de transferência (num(s)/den(s)) para espaço de estados (A, B, C e D). Ex.:

```
» [A B C D] = tf2ss([3 4 1],[2 3 1])

A =

   -1.5000   -0.5000
    1.0000         0


B =

     1
     0


C =

   -0.2500   -0.2500


D =

    1.5000
```

[z p G] = tf2zp(num,den) – Transforma a representação por função de transferência (num(s)/den(s)) para representação de pólos e zeros onde z e p são vetores contendo as raízes do numerador de denominador respectivamente e G o ganho do sistema. Ex.:

```
» [z p G] = tf2zp([3 4 1],[3 2 1])
```

```
z =

   -1.0000
   -0.3333


p =

  -0.3333 + 0.4714i
  -0.3333 - 0.4714i


G =

     1
```

[num den] = zp2tf(z,p,G) - Transforma a representação por pólos e zeros onde z e p são vetores contendo as raízes do numerador de denominador respectivamente e G o ganho do sistema, para a representação por função de transferência num(s)/den(s). Ex.:

```
» [num den] = zp2tf([3; -2],[1-3i; 1+3i],1)

num =

     1    -1    -6


den =

     1    -2    10
```

[A B C D] = zp2ss(z,p,G) - Transforma a representação por pólos e zeros onde z e p são vetores contendo as raízes do numerador de denominador respectivamente e G o ganho do sistema, para a representação em espaço de estados (Matrizes A, B, C e D). Ex.:

```
» [A B C D] = zp2ss([3; -2],[1-3i; 1+3i],1)

A =

    2.0000   -3.1623
    3.1623         0


B =

     1
     0


C =

    1.0000   -5.0596


D =

     1
```

12 – Tópicos avançados

12.1 – Controles e janelas do Windows

No MatLab 4.0 o ambiente Windows possibilitou o uso de componentes Windows nos programas em MatLab. O resultado são programas muito atraentes de iterativos, que facilitam a compreensão e uso por parte do usuário. Alguns comandos são implementados de modo a podermos manipular janelas, botões, caixa de textos, labels, menus, imagens, etc...

12.2 – Criando janelas e controles no MatLab

O controle mais básico existente no Windows é uma janela. A Janela normalmente é o controle mais utilizado dentro do Windows, fazendo com que esta seja a base de toda uma interface que venha a ser criada. Quando se cria uma janela ela serve de “mural” para outros controles. É na janela que colocamos os demais controles como botões, caixas de texto, etc...

Para criar uma janela no MatLab é simples; usamos a função *figure* como mostra abaixo:

```
» H = figure

H =

     1
```

Ao executar este comando, uma janela vazia é aberta e a variável H fica sendo o *identificador* da janela. O valor 1 que temos em H não é importante e não deve ser levado em conta seu valor numérico, o importante na variável H é o que ela *aponta* para uma janela.

Para fecha uma janela usamos o comando *close*. Este comando irá fechar a janela *corrente* (veremos o significado do termo corrente abaixo)

Os demais controles então são criados dentro da janela que acaba de ser criada. O comando para criar controles é o comando *uicontrol* e é usado como se segue:

```
HndCtrl = uicontrol(‘propriedade1’,
‘valor1’,’propriedade2’,’valor2’,...)
```

Com os parâmetros ‘propriedade’ e ‘valor’ podemos escolher o tipo, posição, titulo, etc; do controle que queremos colocar. A variável *HndCtrl* fica sendo o *identificador* do controle criado.

No MatLab, as ações dos programas são sempre executadas seqüencialmente, e os comandos vão atuando com os resultados dos anteriores. Podemos definir um termo *controle corrente, ou janela corrente* (por vezes usaremos a palavra *ativo* para designar janelas ou controles correntes) para designar o controle ou janelas que sofrerão os efeitos dos demais comandos. Quando se cria uma janela a mesma fica sendo a *janela corrente*. Existem comandos para mudar a janela ou controle que estão ativos, com isso podemos manipular com todos os controles e janelas existentes. Portanto, os passos para se criar um controle é; fazer com que a janela onde se queira criar o controle e usar o comando uicontrol para cria-lo. De maneira semelhante procedemos para modificar os atributos de uma janela ou controle;

fazemos com que o controle ou janela que queiramos alterar esteja ativo(a) e depois usamos os comandos de mudança de atributos (comando *SET* como veremos).

Uma janela completa com controles é mostrada a seguir:



Exemplo de controles do Windows no MatLab

Podemos criar uma janela e definir suas propriedades diretamente na função figure. Usando o mesmo esquema de parâmetros usados em uicontrol. Os parâmetros mais importantes são:

Color – Representa a cor de fundo da janela. Pode ser uma string de cor ou um vetor com as componentes RGB.

Colormap – Mapa de cores usado para plots 3D. Deve ser um vetor de 3 colunas com as componentes RGB de cada índice de cor.

CurrentAxes – Faz com que o axes presente no valor dessa propriedade seja o axes corrente.

CurrentObject – Faz com que o controle presente no valor dessa propriedade seja o controle corrente.

MenuBar – Se o valor dessa propriedade for ‘none’ nenhum menu é mostrado na janela. Se for ‘figure’ a janela terá o menu padrão de figuras.

Name – Nome da janela. O valor desta propriedade deve ser uma string.

NumberTitle – Se o valor desta propriedade for ‘on’ aparecerá o nome e o número da janela. Se ‘off’ a barra de titulo aparece em branco.

Pointer – Tipo de ponteiro que estará aparecendo quando o mouse estiver dentro da área da janela. Os valores possíveis são: crosshair, arrow, watch, topl, topr, botl, botr, circle, cross e fleur.

Resize – Se estiver em ‘on’ a janela pode Ter seu tamanho alterado. Se tiver em ‘off’ o tamanho da janela não pode ser alterado.

Units - Unidade usada para posicionar o controle. A posição e tamanho de um controle dentro da janela pode ser feita através de números que representam sua localização. Os possíveis tipos de “coordenadas” são:

Units	**Valor**
Inches	Polegadas conforme o tamanho da janela
Centimeters	Centímetros conforme o tamanho da janela
Normalized	Máximo e mínimo da janela correspondendo a 0 e 1
Points	Pontos da janela
Pixels	Pixeis gráficos

Visible - Visibilidade do controle. Se *on* o controle é visível ao usuário, se *off* o controle é invisível.

UserData - Espaço para o usuário. Nesta propriedade o usuário pode colocar algum valor de referência ou de uso qualquer já que este não interfere no funcionamento do controle.

12.3 – Os comandos *SET* e *GET*

Como vimos acima, podemos modificar os parâmetros dos controles ou janelas já criadas. O comando para modificar as propriedades é o comando *SET*. Através do comando *SET* podemos modificar os mesmos parâmetros que usamos para criar o controle ou (no caso de janelas) modificar parâmetros que não foram citados na hora da criação do controle ou janela e que assim assumiram um valor *defaut*. O comando *SET* é usado como mostrado abaixo:

```
SET(H,’parametro1’,’valor1’,’parametro2’,’valor2’,...)
```

‘parametro’ e ‘valor’ referem-se as mesmas propriedades do controle que foram usados no comando uicontrol. No caso de janelas serão parâmetros como cor, tamanho etc... que estão em seus valores *defaut*. H é o *identificador* do controle a ser mudado.

Se quisermos examinar os parâmetros que estão sendo usados por um controle, usamos o comando *GET*. O comando *GET* mostra os parâmetros e os seus respectivos valores de um controle. Para usa-lo basta digitar: GET(H,‘parametro’) onde H é o identificador do controle e ‘parametro’ é o nome do parâmetro que se quer ver o valor (*GET(H)* faz com que todos os parâmetros sejam mostrados).

12.4 – Controlando os controles

Como sabemos, controles não servem só para enfeitar uma janela, eles geralmente “fazem alguma coisa”. No MatLab existe uma maneira muito prática de se programar a resposta de um controle ao usuário. Por exemplo; ao apertarmos um botão queremos que seja plotado um gráfico, ou que se feche a janela que estamos operando. Porem, nem sempre os controles servem para fazer alguma coisa e sim para retornar algum valor dependendo de seu estado. Para obtermos o estado de um controle, usamos o comando *GET(H,’value’)* (H é o identificador do controle) A propriedade ‘value’ contem o estado do controle no momento

e seu valor varia com o controle, por exemplo, ‘value’ em uma caixa de texto contem o texto que está escrito na caixa naquele momento, já em uma barra de rolagem, contem um número que representa a posição do cursor de rolagem naquele momento.

Controles que realizam alguma ação, normalmente botões, precisão ter um parâmetro que contem comandos a serem executados caso o mesmo seja acionado. Este parâmetro é o parâmetro ‘CallBack’ seu valor é uma seqüência de comandos em forma de string que serão executados caso o controle seja acionado. Por exemplo: Se H é o identificador de um botão, o comando; SET(H,’CallBack’,’t=0:0.01:10; f=sin(x); plot(f,t);’)
Irá definir as variáveis t e f (f-sin(t))e irá plotar o gráfico de f.

12.5 – uicontrol

Como foi visto, o comando *uicontrol* cria um controle na janela que estiver ativa. Para criar os controles, devemos usar os parâmetros do comando *uicontrol* de forma adequada. Dessa forma o comando *uicontrol* age como uma função, retornado um identificador para o controle criado.

12.5.1 – parâmetros do uicontrol

Os parâmetros de *uicontrol* indicam que controle deverá ser criado e como e onde o mesmo irá aparecer na janela corrente. Essas informações são passadas ao comando através de seus parâmetros, como são muitos servem para diversos controles, mostraremos um por um em detalhes.

Style - Estilo de controle que será criado. Esta propriedade diz que tipo de controle será criado, se é botão, caixa de texto, etc... Os valores para esta propriedade são:

Controle	Descrição	Figura
Pushbutton	Botão comum	
Radiobutton	Botão de “rádio”	
Checkbox	Botão de checagem	
Edit	Caixa de texto simples	
Text	Texto simples	
Slider	Barra de rolagem	
Frame	Quadro simples	
Popupmenu	Caixa de texto composta	

Units - Unidade usada para posicionar o controle. A posição e tamanho de um controle dentro da janela pode ser feita através de números que representam sua localização. Os possíveis tipos de “coordenadas” são:

Units	**Valor**
Inches	Polegadas conforme o tamanho da janela
Centimeters	Centímetros conforme o tamanho da janela
Normalized	Máximo e mínimo da janela correspondendo a 0 e 1
Points	Pontos da janela
Pixels	Pixeis gráficos

Position - Posição do controle na janela seguindo a unidade usada. O valor é um vetor linha com as coordenadas X e Y do canto superior esquerdo do controle, sua largura e sua altura. Ex: [X Y tamanho altura]

String - Texto mostrado no controle. Alguns controles não possuem nenhum texto, como barras de rolagem, caixas de texto, etc... O controle *popupmenu* possue string multipla.

Enable - Habilitação do controle. Se seu valor for *on* o controle pode ser utilizado, se for *off* o mesmo aparece desabilitado, não podendo assim ser modificado pelo usiário.

CallBack - String de ações para o controle. Armazena uma string com comandos que serão executados ao acionamento do controle.

Visible - Visibilidade do controle. Se *on* o controle é visível ao usuário, se *off* o controle é invisível.

BackgroundColor - Cor de fundo. Pode ser uma cor definida ou um vetor linha com as componentes RGB. Ex: [0.5 0.5 0.5]

ForegroundColor - Cor de fora . Pode ser uma cor definida ou um vetor linha com as componentes RGB. Ex: [0.5 0.5 0.5]

HorizontalAlignment - Alinhamento do controle. Os valores são: left, center e right representando respectivamente alinhamento à direita, centro e esquerda respectivamente.

Max - Valor máximo da posição do cursor em uma barra de rolagem. Representa o valor que ficará em ‘value’ quando o cursor estiver todo à direita da barra.

Min - Valor mínimo da posição do cursor em uma barra de rolagem. Representa o valor que ficará em ‘value’ quando o cursor estiver todo à esquerda da barra.

Value - Valor de estado do controle. Para cada controles existe uma interpretação dessa propriedade, como mostra a tabela abaixo.

Controle	**Valor**
Pushbutton	“não se aplica”
Radiobutton	0 se desmarcado e 1 se marcado
Checkbox	0 se desmarcado e 1 se marcado
Edit	Contém o texto digitado
Text	“não se aplica”
Slider	Contém a posição do cursor
Frame	“não se aplica”
Popupmenu	Contem o texto atualmente selecionado

UserData - Espaço para o usuário. Nesta propriedade o usuário pode colocar algum valor de referência ou de uso qualquer já que este não interfere no funcionamento do controle.

12.6 – uimenu

Um componente também muito usado no Windows é o *menu*. Praticamente todas as janelas no Windows possue um menu. No MatLab podemos colocar menus nas janelas e fazer com que o mesmo responda aos comandos do usuário e realize execute determinados comandos.

Um menu é composto por *submenus* que podem ser compostos também de mais *submenus*, obtendo-se assim uma espécie de "hierarquia" de menus. Neste tipo de componente um submenu deve ser referenciado ao submenu imediatamente superior. Para criar menus obedecendo essa regra, utilizamos o comando *uimenu* de forma que sua seqüência será a seqüência de aparecimento na janela onde o mesmo está colocado e o menu ou submenu superior será um parâmetro. Ex.:

```
sH1 = uimenu(H1, 'parametro1', 'valor1', 'parametro2', 'valor2', ...);
```

sH1 é o identificador do menu (ou submenu) gerado. H1 é o menu (ou submenu) imediatamente superior. Neste caso o menu sH1 só será visível se acionarmos o menu (ou submenu H1)

Quando se cria um menu ou submenu, o mesmo a adicionado a barra de menu ou menu imediatamente superior. Portanto, ao criar uma janela, se desejarmos que apenas nosso menu seja mostrado, devemos cria-la com o parâmetro *MenuBar* igual a *'none'* fazendo com que a janela seja criada sem o menu padrão.

12.6.1 – Parâmetros do uimenu

O uso dos parâmetros de uimenu bem como a possibilidade de alteração dos mesmos com os comandos GET e SET são idênticas aos uicontrols. Sendo que o uimenu possue parâmetros um pouco diferentes, que serão explicados abaixo.

Label – O valor dessa propriedade será o titulo mostrado pelo menu. Valor é uma string.

Enabled – Se 'on' o menu está habilitado e o usuário pode clica-lo. Se 'off' o menu está "cinza" e não responde aos cliques do usuário.

Visible – se 'on' o menu é visível, se 'off' o mesmo e invisível e não pode ser utilizado.

Accelerator – Tecla de atalho para acionar o menu. O valor desta propriedade é uma string representando a tecla de atalho que aciona o menu.

Checked – Se 'on' o menu está marcado, se 'off' esta desmarcado (porem pode ser marcado). Se não for definido valor para esta propriedade o menu não é de marcação.

CallBack – String com os comandos a serem executados caso se clique no menu.

Separator – Se 'on' o menu é apenas uma bara separadora, se 'off' será um menu normal.

UserData – Parâmetro reservado para o usuário. Seu valor não influi em nada no menu.

13 – Exemplo de aplicação

Para finalizar o conteúdo desta apostila damos de exemplo uma aplicação completa no MatLab. O conjunto de programas a seguir formam uma aplicação destinada a observar a resposta de um sistema à uma entrada periódica que pode ser dada ou pela expressão dividida em intervalos os quais se repetiram formando uma função periódica, ou através do mouse simplesmente “desenhando” a função.

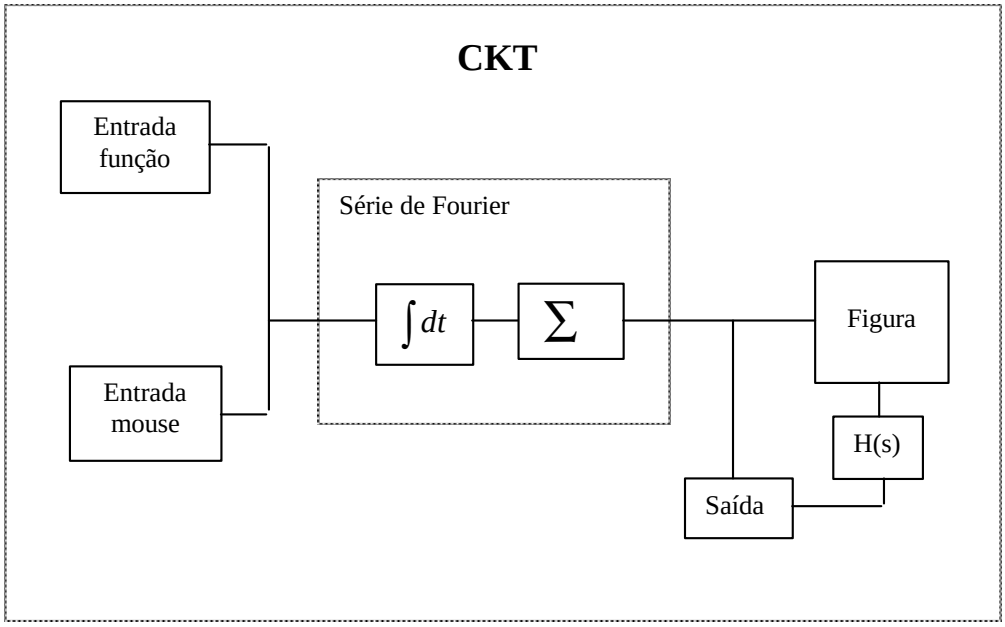
O funcionamento do programa é o seguinte; O usuário irá entrar com dados referentes a uma função periódica em t. O programa a seguir calcula N termos da série de Fourier correspondente à função. De posse dos dados da série de Fourier, o mesmo faz a computação dos termos correspondentes à saída aplicando a função de transferência dada.

A entrada da função requer que o usuário informe (usando os controles mostrados): o tempo inicial “t_0”, a função de transferência H(s), o número de termos da série de Fourier, e um conjunto de funções f(t) com seus devidos “passos” ou seja intervalos de tempo t onde são definidas. O modo como o usuário informa as partes da função de entrada pode ser através do botão “add”, da caixa de texto “f(t)=”, e da caixa de texto “passo”. Cada vez que o usuário pressiona o botão “add” uma parte da função é definida como o valor da caixa “f(t)” e com intervalo de tamanho “passo”, pressionando novamente, uma nova parte é definida como o valor da caixa “f(t)” só que desta vez definida começando no final da primeira e com tamanho “passo”.

Uma maneira alternativa de entrar com a função de entrada é através do mouse, simplesmente “desenhando” a função. Um programa captura pontos da janela e os liga (começando do 0) com uma linha amarela sendo que cada linha representa uma reta e o intervalo fica definido automaticamente pelo programa, assim quando o último ponto for escolhido (pressionando o botão direito do mouse), o programa gera as partes da função de acordo com a retas colocadas e os intervalos correspondentes.

Daí os botões “entrada” e “saída” plotam a soma dos termos calculados para a serie e a soma dos termos processados com a função de transferência dada.

O programa está dividido em pequenos blocos como é mostrado a seguir:



Cada bloco (ou conjunto de blocos) é composto de um programa ou função armazenado em um arquivo .m e suas listagens estão no apêndice A e a explicação da função de cada bloco será dada a seguir.

CKT – É o programa principal (ckt.m). Este não realiza cálculo algum, o mesmo é apenas responsável por desenhar os controles na janela de trabalho do aplicativo, e associar suas respectivas ***strings*** de ***callback*** e também pela montagem da função de entrada usando-se o botão add e as caixas de texto de "f(t)" e "passo". Ckt coloca os botões "entrada", "saida", que plotam a soma da série de entrada e saída, o botão "desenha", que chama o programa para obter a função de entrada através do mouse, o botão "reset" que limpa todas as variáveis e deixa o programa pronto para ser usado novamente, a caixa de t_0 e H(s), as caixas para entrada da função manualmente, e um axes onde serão plotadas as funções.

Entrada função – É onde se entra com a função de entrada do sistema manualmente, programa para montar a mesma está embutido em ckt.m.

Entrada mouse – É o programa que monta a função de entrada com o mouse. O mesmo possue um ***loop*** que espera até que o botão direito do mouse seja pressionado, enquanto isso, o mesmo vai coletando os pontos onde o usuário clica e vai calculando as equações das retas correspondentes.

Série de Fourier – É o bloco responsável pelo cálculo dos coeficientes da série de Fourier. O mesmo usa um programa para fazer a integral com variáveis diferentes da padrão.

$\int dx$ - É o bloco que está no arquivo intm.m . Este bloco faz integral com variáveis período e N, que não estão definidas no corpo da função.

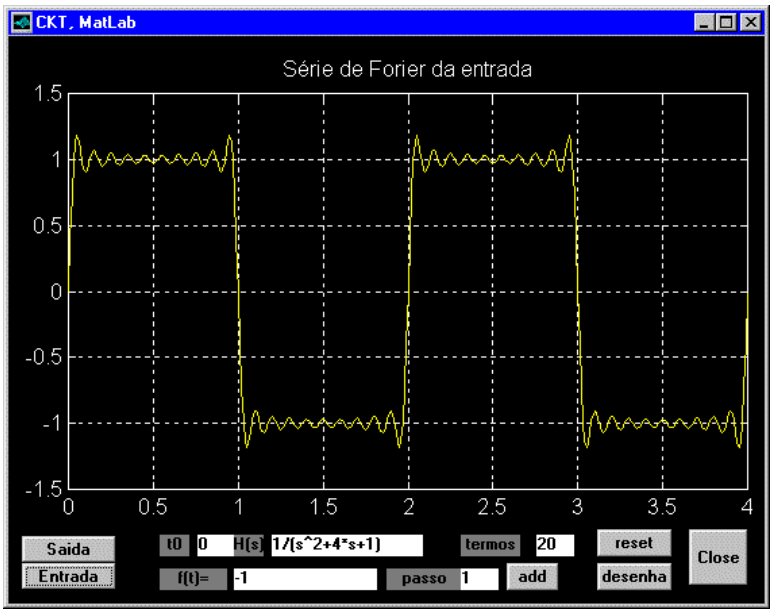
$\sum$ - Este bloco soma os termos já integrados da série de Fourier e calcula os termos a_n e b_n da série. A chama ao integrador e o programa de soma estão no arquivo sfourier.m .

Saída – Este bloco serve apenas de representação do botão que plota o gráfico da saída, não possuindo tarefa relacionada ao mesmo.

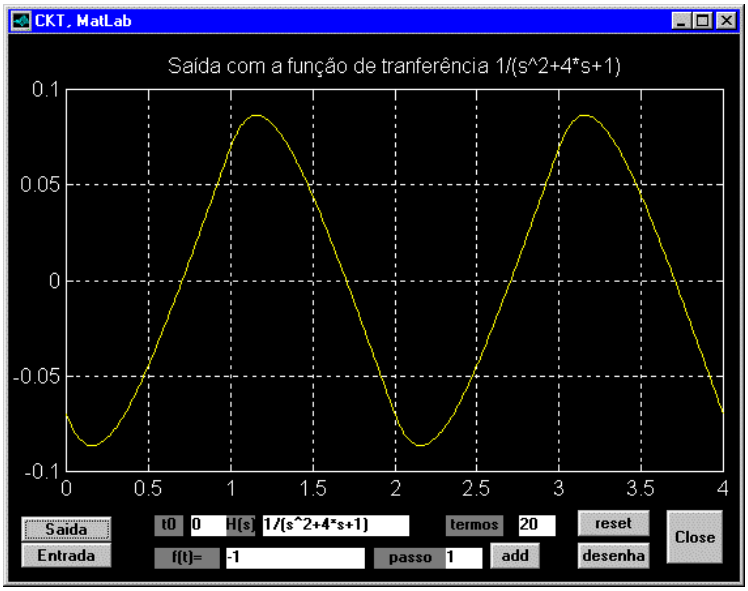
H(s) – Este bloco encontra-se no arquivo tfourier.m e é o bloco que processa os termos a_n e b_n com a função de transferência dada.

Figura – Este bloco apenas representa o axes de saída, onde as funções são plotadas.

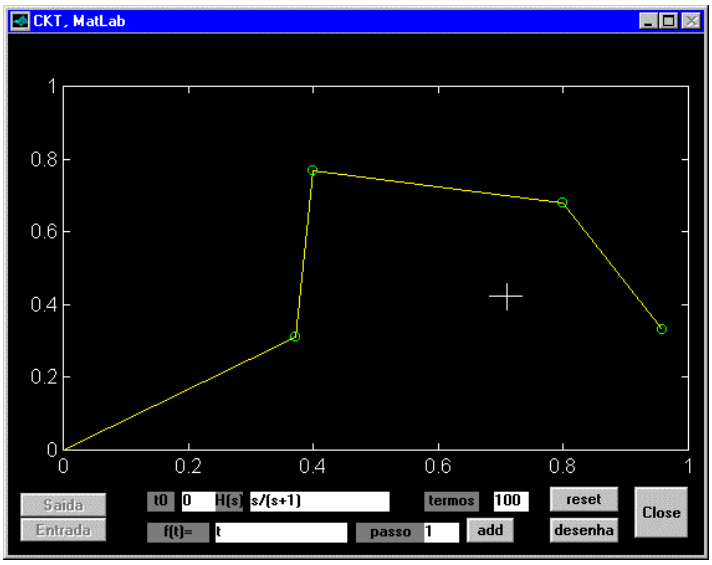
A seguir serão mostrados alguns exemplos do funcionamento do programa.



Entrada de uma onda quadrada de período 2 representada com 20 termos da série de Fourier



Resposta à entrada quadrada por um circuito com função de transferência $H(s) = \frac{1}{s^2 + 4s + 1}$



Exemplo de entrada de uma função com o mouse.

Bibliografia

[1] – Student Eedition of MatLab 4 HELP
Copyright© 1984-1994 the MathWorks, Inc

[2] – Apostilha de MatLab, prof. Ednilson Pereira de Freitas
Engenharia Elétrica, USP

[3] – MathWorks Home Page, MathLab's
http://www.mathworks.com

Apêndice

Listagem do arquivo CKT.M

```
clear;
periodo=0;
i=1;
axes('position',[0.08 0.2 0.9 0.7]);
plotsaidastr = 'tfourier;';
plotentradastr = 'sfourier; set(btsaida,''Enable'',''on'');';
stemp ='t1=0; dt=0.01; t2=2*periodo;
termos=eval(get(cntl_termos,''string'')); H=get(cntl_hs,''string'');';
addfunctionstr = ['set(btentrada,''Enable'',''on'');
x(1)=eval(get(cntl_x0,''string'')); tm(1)=x(1); fti=[''(''
get(cntl_ft, ''string'') '')''];  eval([''F'' int2str(i) '' =
fti;'']); i=i+1; x(i)=eval(get(cntl_passo,''string'')); tm(i)=tm(i-
1)+x(i); partes=i-1;  periodo=periodo+x(i); eval(stemp);'];
%res = ['x=0; y=0; an=0; bn=0; t=0; f=0; periodo=0; i=1; x=0; tm=0;
set(btsaida,''Enable'',''off''); set(btentrada,''Enable'',''off'');'];
res='clear; close; ckt;';
desenhastr = 'termos=eval(get(cntl_termos,''string''));
H=get(cntl_hs,''string''); mouse; set(btentrada,''Enable'',''on'');';

btentrada=uicontrol( ...
    'Style','pushbutton', ...
    'Units','pixels', ...
    'Position',[10 10 70 20], ...
    'String','Entrada', ...
    'Enable','off', ...
    'Callback', plotentradastr);

btsaida=uicontrol( ...
    'Style','pushbutton', ...
    'Units','pixels', ...
    'Position',[10 30 70 20], ...
    'String','Saida', ...
    'Enable','off', ...
    'Callback', plotsaidastr);

btclose=uicontrol( ...
    'Style','pushbutton', ...
    'Units','normalized', ...
    'Position',[0.9 0.03 0.08 0.1], ...
    'String','Close', ...
    'Enable','on', ...
    'Callback' , 'close');

h = uicontrol('Style','text','Units','normalized', ...
                'String','t0','Position',[0.2 0.08 .04 .04]);
cntl_x0 =
uicontrol('Style','edit','Units','normalized','BackgroundColor',
'white', ...
                'String','0','Position',[0.25 0.08 .05 .04]);

h = uicontrol('Style','text','Units','normalized', ...
                'String','H(s)','Position',[0.3 0.08 .04 .04]);
```

```
cntl_hs =
uicontrol('Style','edit','Units','normalized','BackgroundColor',
'white', ...
                  'String','s/(s+1)','Position',[0.35 0.08 .2 .04]);

h = uicontrol('Style','text','Units','normalized', ...
                  'String','termos','Position',[0.6 0.08 .08 .04]);
cntl_termos =
uicontrol('Style','edit','Units','normalized','BackgroundColor',
'white', ...
                  'String','100','Position',[0.7 0.08 .05 .04]);


h = uicontrol('Style','text','Units','normalized', ...
                  'String','f(t)=','Position',[0.2 0.02 .09 .04]);
cntl_ft = uicontrol('Style','edit','Units','normalized',
'BackgroundColor', 'white',...
                  'String','t','Position',[0.3 0.02 .19 .04]);

h = uicontrol('Style','text','Units','normalized', ...
                  'String','passo','Position',[0.5 0.02 .1 .04]);
cntl_passo = uicontrol('Style','edit','Units','normalized',
'BackgroundColor', 'white',...
                  'String','1','Position',[0.6 0.02 .05 .04]);
h = uicontrol('Style','pushbutton','Units','normalized', ...
                  'String','add','Position',[0.66 0.02 .07
.05],'Enable','on', 'Callback', addfunctionstr );

contHndl=uicontrol( 'Style','pushbutton',
'Units','normalized','Position',[0.78 0.08 0.1 0.05], ...
    'String','reset', 'Enable','on', 'Callback' , res);

contHndl=uicontrol( 'Style','pushbutton',
'Units','normalized','Position',[0.78 0.02 0.1 0.05], ...
    'String','desenha', 'Enable','on', 'Callback' , desenhastr);
```

Listagem do arquivo MOUSE.M

```
plot([]);
hold on;
axis([0 1 0 1]);
but = 1;
n=1;
x=0;
y=0;
while but == 1,
   [xi,yi,but] = ginput(1);
   if (xi>x(n))
     plot(xi,yi,'go','era','back');
     x = [x; xi];
     y = [y; yi];
     n = n + 1;
     line(x,y);
   end
end
```

```
tm(1)=x(2);
for a=2:1:n-1
  inc=(y(a+1)-y(a))/(x(a+1)-x(a));
  ind=y(a)-x(a)*inc;
  ss=['(' num2str(ind) '+t*(' num2str(inc) '))'];
  eval(['F' int2str(a-1) ' = ss;'] );
  tm(a)=x(a+1);
end
partes=n-2;
periodo=x(n)-x(2);
t1=x(2);
dt=0.01;
t2=2*x(n);
hold off;
```

Listagem do arquivo INTM.M

```
function s = intm(F,v,x0,x1,sv1,v1,sv2,v2)

if (sv1~='')
  eval([sv1 ' = ' num2str(v1) ';']);
end
if (sv2~='')
  eval([sv2 ' = ' num2str(v2) ';']);
end

s=eval(int(F,v,x0,x1));
```

Listagem do arquivo SFOURIER.M

```
a0=0;

Ff = '';

for aa=1:1:partes
 a0=a0+1./periodo.*(intm(eval(['F'
int2str(aa)]),'t',tm(aa),tm(aa+1),'periodo',periodo,'',0));
end

for n=1:1:termos
  at=0;
  bt=0;
  for aa=1:1:partes
    an(n)=at+2./periodo.*(intm([eval(['F' int2str(aa)]) '*'
'cos(n*2*pi*1/periodo*t)'
],'t',tm(aa),tm(aa+1),'periodo',periodo,'n',n));
    bn(n)=bt+2./periodo.*(intm([eval(['F' int2str(aa)]) '*'
'sin(n*2*pi*1/periodo*t)'
],'t',tm(aa),tm(aa+1),'periodo',periodo,'n',n));
    at=an(n);
    bt=bn(n);
```

```
  end
end

w=0;
s=0;

count=0;
for t=t1:dt:t2
  count=count+1;
  ft=a0;
  for n=1:1:termos
    w=n.*2.*pi./periodo;
    a=an(n);
    b=bn(n);
    f(count)=ft+a.*cos(w.*t)+b.*sin(w.*t);
    ft=f(count);
  end
end

t=t1:dt:t2;
plot(t,f);
grid on;
title('Série de Forier da entrada');
```

Listagem do arquivo TFOURIER.M

```
count=0;
for t=t1:dt:t2
  count=count+1;
  s=0;
  ft=a0*eval(H);
  for n=1:1:termos
    w=n.*2.*pi./periodo;
    s=sqrt(-1).*w;
    Hr=real(eval(H));
    Hi=imag(eval(H));
    a=an(n).*Hr+bn(n).*Hi;
    b=-an(n).*Hi+bn(n).*Hr;
    f(count)=ft+a.*cos(w.*t)+b.*sin(w.*t);
    ft=f(count);
  end
end

t=t1:dt:t2;
plot(t,f);
grid on;
title(['Saída com a função de tranferência ' H]);
```