Universidade de São Paulo
Escola Politécnica
Departamento de Engenharia Química
Curso de Especialização em Controle e Otimização de Processos
Professores: Dr. Darci Odloak; Dr. Reinaldo Giudici

Noções Básicas sobre a utilização do MATLAB

Marco Antônio Rodrigues - Doutor em Engenharia pela EPUSP

Referências

Hanselman, Duane e Bruce Littlefield (2002). MATLAB 6 Curso Completo. *Prentice Hall Brasil*, Primeira Edição, ISBN: 8587918567

Tutorial do Departamento de Engenharia Química do MIT, Numerical Methods Applied to Chemical Engineering, por Kenneth Beers. (Disponível em PDF).

Websites sobre MATLAB

1. http://www.owlnet.rice.edu/~ceng303/Matlab/MatCont.html
2. http://techteach.no/publications/control_system_"toolbox"/
3. http://www.engin.umich.edu/group/ctm/home.text.html
4. http://www.glue.umd.edu/~nsw/ench250/matlab.htm
5. http://www.eng.fsu.edu/~cockburn/matlab/matlab_"help".html

1.0 Introdução

Nestas notas são descritos comandos básicos e algumas ferramentas numéricas que serão usadas ao longo do curso. Compreende-se que esta pretende ser apenas a apresentação do MATLAB aos alunos e, então, que não deve ser encarada como única fonte de consulta. As referências citadas anteriormente são razoáveis para o nível introdutório. No aprendizado de qualquer linguagem de programação, somente a prática e muitas vezes a tentativa-e-erro, é que levam ao amadurecimento. Nesse sentido, o MATLAB apresenta uma facilidade por possuir um "help" bem claro da maioria dos comandos. A partir da versão 6.0, o "help" foi remodelado na forma de hipertexto e traz uma série de exemplos que são bem úteis para a compreensão dos comandos e rotinas.

1.1 O que é o MATLAB?

O MATLAB é um "software" integrado e interativo que possui um conjunto bem amplo de programas ("toolboxes") para a solução de diversos problemas nas áreas de engenharia, ciências fundamentais, economia, dentre outras. Ao mesmo tempo o MATLAB é uma linguagem de programação que permite ao usuário construir seus próprios códigos, usando ou não as subrotinas disponíveis em seu sistema e "toolboxes".

Quais são as principais diferenças entre o MATLAB e outra linguagem de programação, tal como o FORTRAN? O MATLAB é uma linguagem *interpretada* ao passo que o FORTRAN é uma linguagem *compilada*.

Numa linguagem de programação compilada, os comandos são convertidos diretamente em instruções para máquina que são armazenadas num arquivo. A execução dos comandos não é feita até que o processo de compilação seja executado. Numa linguagem compilada, é preciso empregar comandos para a entrada e saída de dados e para designação de variáveis e alocação de espaço de memória para estas (p.e. o comando `INTEGER` no FORTRAN). Importante mencionar que linguagens compiladas são idealizadas para ter uma quantidade mínima de comandos e sintaxe, de tal modo que qualquer tarefa que possa ser efetuada por uma seqüência mais básica de instruções não é incorporada à definição da linguagem, mas deixada para uma subrotina. Deste modo, bibliotecas de subrotinas foram escritas por especialistas em matemática aplicada e computação para realizar cálculos numéricos usuais. Entretanto, acessar tais bibliotecas querer que o código em desenvolvimento seja ligado ("linked") às mesmas. Apesar de tal operação não ser conceitualmente complicada, a mão-de-obra para projetos simples não é desprezível.

Numa linguagem interpretada, o programa principal que interpreta os comandos do usuário durante o uso

já foi escrito e compilado. Deste modo, no MATLAB os comandos são interpretados linha-a-linha. Logo, uma linguagem interpretada demanda maior esforço computacional do que uma linguagem compilada. Assim, para aplicações que demandam elevado esforço computacional, ou para aplicações no ambiente industrial, o uso de linguagens compiladas é recomendado. Por outro lado, para aplicações de pequeno a médio porte, existem vantagens na utilização de linguagens interpretadas. Nestas, as etapas de declaração e dimensionamento de estruturas (matrizes, vetores etc.), a compilação e a ligação com as subrotinas não são necessárias. Uma das grandes vantagens do MATLAB consiste na facilidade de visualização de resultados. Existem diversas rotinas gráficas, de fácil manuseio, que permitem gerar gráficos completos e bem acabados diretamente, sem a necessidade de importação/exportação de dados e utilização de pacotes específicos. A confecção de figuras em formato final de impressão, tais como `.eps` e `.tiff` tornou-se uma tarefa mais simples a partir Versão 6 do MATLAB. Por tais razões, pode-se programar de modo mais eficiente em MATLAB do que em linguagens compiladas como o FORTRAN ou o C/C++.

O MATLAB não é mais um aplicativo simples mas tornou-se uma ferramenta robusta e completa para o desenvolvimento de tarefas de cálculo em pesquisa e aplicação tecnológica/científica. Existem rotinas prontas e de acesso fácil para a maioria dos procedimentos nas diversas áreas. Além disso, o MATLAB vem com um compilador adicional que permite converter o código MATALB num código C ou C++ que pode ser compilado para gerar um executável "stand-alone". Outra possibilidade é que usando as ferramentas do compilador, pode-se combinar o código MATLAB com subrotinas em FORTRAN ou C/C++, o que abre um amplo espectro de possibilidades ao usuário. Por todas essas razões, o MATLAB é uma escolha adequada para um curso introdutório em cálculos científicos.

1.2 O espaço de trabalho do MATLAB.

O MATLAB pode ser executado (aberto), no Windows, a partir do menu de programas do INICIAR, ou por um atalho na tela principal. Uma vez iniciado, o usuário estará dentro do espaço-de-trabalho ("workspace") do MATLAB, normalmente uma janela de fundo branco com um "prompt" do tipo "≫". Nesta ambiente o usuário pode executar os comando diretamente ou através de um arquivo ".m", conforme será visto na seqüência.

O "workspace" possui uma memória ("history") que acumula os comandos utilizados. Portanto, basta pressionar a tecla ↑ para repetir os comandos mais recentes e ↓ para os mais anteriores. Uma dica interessante é digitar a primeira letra do comando e a tecla ↑. Assim, o usuário obterá os comandos que iniciam pela letra especificada. Isto faz uma boa diferença depois de algum tempo de trabalho.

Uma ultima observação quanto ao "workspace" do MATLAB é sobre como controlar a exibição dos resultados dos comandos na tela. A regra é a seguinte: "," (virgula) ou nada após o comando ou variável irá exibir o resultado. ";" (ponto-e-vírgula) irá suprimir a exibição do resultado.

Ex: ≫`teste = 1`

aparecerá em seguida

```
≫teste =
     1
```

e

```
≫teste = 1;
```

não exibirá nada a seguir.

1.3 Matemática Elementar

Quanto às operações aritméticas elementares, vale a pena comentar somente a divisão e a potenciação.

A divisão no MATLAB é entendida como divisão-a-direita e divisão-a-esquerda, ou seja:

/ ou \

Ex: `33/11 = 11\33`

Na verdade, a divisão-a-direita sintetiza um conjunto de cálculos complexos que generaliza a operação inversão de matrizes. Para uma melhor compreensão deste comando, recomenda-se a leitura do Capítulo 9 do tutorial do primeiro link da lista da página 1.

Já a potenciação é efetuada pelo til (chapéu) número ou variável e a potência

Ex: `33^4`

1.4 Variáveis

O MATLAB admite variáveis com comprimento de até 31 caracteres. Também é sensível a maiúsculas e minúsculas. Símbolos de pontuação não são permitidos por terem uma sintaxe específica e as variáveis não podem ser acentuadas, nem com cedilha. No nome das variáveis podem estar contidos números, desde que não sejam o primeiro caracter.

Ex: ≫`Ab = 333.33`
≫`X11 = 0.001`
≫`a_b_1_11 = 3.e5`

No último item do exemplo anterior foi mostrada a

forma como o MATLAB representa potências ou a notação científica do MATLAB.

As seguintes expressões são reservadas para uso exclusivo do sistema do MATLAB:

`ans`	variável padrão usada para resultados
`pi`	o número π
`eps`	menor número de ponto flutuante
`flops`	contador do núm. de operações de ponto flutuante
`inf`	infinito
`NaN`	não numérico
`i (e) j`	unidade imaginária
`nargin`	número de argumentos de entrada de uma função
`nargout`	número de argumentos de saída de uma função
`realmin`	menor número real positivo utilizável
`realmax`	maior número real positivo utilizável

A lista de todas as variáveis acumuladas no "workspace" é obtida pelo comando `who`. Na Versão 5 e superiores do MATLAB, o comando `who` indica, também, a dimensão de cada variável.

Para fechar essa seção vale a pena comentar como se "limpa" uma variável ou todo o "workspace" e um último cuidado quanto ao comprimento máximo de uma variável.

O MATLAB considera as variáveis utilizadas no "workspace" (e dentro dos arquivos .m, desde que não sejam funções) como globais. Assim, executando-se o comando `clear` no "prompt" de comando, todo o conteúdo do "workspace" será eliminado. Nesse comando não há confirmação para a execução. Já para limpar somente uma variável ou um grupo de variáveis, basta executar o comando `clear` seguido da `lista de variáveis`.

2.0 Manipulação de Matrizes e Vetores

2.1 Preliminares

Conforme pode-se ver no livro de Hanselman e Littlefield (2002), o nome MATLAB vem da sigla "MATRIX LABORATORY". Dessa forma, é fundamental compreender como trabalhar essas estruturas no MATLAB.

O MATLAB considera *todas* as variáveis como matrizes. Assim, no MATLAB um escalar (um número ou variável simples) é uma matriz de uma linha e uma coluna.

Na utilização do MATLAB no contexto do controle, otimização e simulação de processos é extremamente comum termos matrizes e vetores de dimensões elevadas. Logo, tais variáveis não podem ser completamente visualizadas na tela do computador. Dessa forma, o comando `size` é fundamental para saber as dimensões das variáveis dos problemas que estão sendo resolvidos. Para tanto, considere o seguinte exemplo:

Ex: Suponha que A seja uma matriz de 10 linhas por 10 colunas. Então:

```
≫ size(A)
ans =
    10    10
```

O comando `size` sempre retorna uma matriz de uma linha e duas colunas. O primeiro elemento dessa matriz é o número de linhas da variável e o segundo o número de colunas. Uma vez que em nossos problemas freqüentemente manipulamos com o número de linhas e colunas das matrizes, é interessante aplicarmos a seguinte estratégia na utilização do `size`.

Ex: `≫ [nl,nc]=size(A);`

ou seja, `nl` acumula o número de linhas de `A` e `nc` o número de colunas.

No exemplo anterior foi introduzida uma forma de se criar matrizes e vetores no MATLAB. O exemplo seguinte ajudará a compreender melhor esta sintaxe.

Ex: `≫ A=[10, 22, 1; -1, 8, 4; 0, 0, 1]` ou
`≫ A=[10 22 1; -1 8 4; 0 0 1]` ou
`≫ A=[ 10 22 1`
`-1 8 4`
`0 0 1].`
irá resultar em

```
≫ A =
    10    22     1
    -1     8     4
     0     0     1
```

Portanto, é direto deduzir que na definição de uma matriz o MATLAB usa a vírgula para separar as colunas e o ponto-e-vírgula para separar as linhas ou o espaços para separar as colunas e novas linhas para separar as linhas.

A criação de vetores é feita de forma semelhante, com a diferença dos vetores poderem ser matrizes de uma linha (vetor linha) ou de uma coluna (vetor coluna).

Ex: `≫ x=[10 22 1]`
irá resultar em

```
≫ x =
    10    22     1
```

e

`≫ x=[10; 22; 1]` ou
`≫ x=[10`
`22`
`1]` ou ainda
`≫ x=[10 22 1]'`
irá resultar em

```
≫ x =
    10
    22
     1
```

Na última linha do exemplo anterior foi introduzido o comando "'" (apóstrofo) que é usado para a transposição no MATLAB, ou seja, transformar linhas em colunas. Esta é uma operação muito freqüente. Deve-se tomar cuidado quando se manipula números complexos. Neste caso, o comando "'" resultará no complexo conjugado do número. Para a transposição simples do número complexo deve-se usar o comando ".'" para obter o resultado esperado.

Ex: Seja o número complexo:

```
>> y = 5 + 0.35i
```

Então:

```
>> y' =
   5.0000 - 0.3500i
```

e

```
>> y.'=
   5.0000 + 0.3500i
```

Tendo em vista essa advertência, é preciso analisar qual das operações de transposição se deseja realizar.

Para visualizar (ou extrair) uma linha ou coluna de uma matriz, basta executar o comando

```
>> a=A(:,1)

a =

    10
    -1
     0
```

e

```
>> b=A(1,:)

b =

    10    22     1
```

ou seja, ":" na posição da linha ou coluna da matriz, considera toda essa dimensão.

Outra manipulação muito usual, é extrair partes das matrizes.

Ex: Suponha que seja necessário extrair apenas as duas primeiras colunas e a primeira e última linha da matriz A dos exemplos anteriores. Para tanto, basta executar:

```
>> B=[A(1,1:2);A(3,1:2)]

B =

    10    22
     0     0
```

No exemplo anterior, na verdade foram extraídos e empilhados dois vetores, criando uma nova matriz, B.

2.2 Multiplicação de matrizes e vetores

Mais uma vez é importante lembrar que o MATLAB trabalha com a estrutura de matricial. Desta forma, para uma melhor utilização do MATLAB devemos estar seguros (pelo menos conhecer o mínimo) da álgebra matricial. Desta forma, as multiplicações de matrizes e vetores obedecem a essas regras.

Ex: Considere a matriz `A` e o vetor coluna `x` dos exemplos anteriores. Então

```
≫A*x
```

irá resultar em

```
≫ans =
   585
   170
     1
```

mas se fizermos

```
≫x*A
```

obteremos

```
≫??? Error using ==> *
  Inner matrix dimensions must agree.
```

pois o número de colunas de `x` (1) é diferente do número de linhas de `A` (3). Agora, se fizermos:

```
≫x'*A
```

obteremos

```
≫ans =
      78   396    99
```

Tarefa. Buscar num livro de álgebra matricial as definições de produto escalar e produto cruzado. No "workspace" do MATLAB, chamar o "help" dos comandos `dot` e `cross`. Definir duas matrizes quaisquer, de dimensões compatíveis, e realizar o produto cruzado e o produto escalar entre estas. Quando um é igual ao outro?

2.3 Operações e comandos especiais com matrizes e vetores

No dia-a-dia dos estudos nas áreas de controle e otimização de processos é muito comum realizarmos algumas operações com matrizes e utilizarmos algumas matrizes especiais. Assim, operações como a transposição (descrita no item 2.1), inversão (desde que exista a inversa!), potenciação, cálculo de autovalores, cálculo de valores singulares, dentre diversas outras, são fácil e rapidamente executadas pelo MATLAB. Além disso, a criação de matrizes nulas retangulares ou não e de matrizes identidades de dimensões quaisquer, também são trivialmente realizadas neste pacote.

Ex: Considere, novamente, a matriz `A` dos exemplos anteriores.

Seu determinante é calculado através do comando:

```
>> det(A)
```

resultando em

```
ans =
   102
```

Os autovalores de A são calculados através da solução do sistema de equações obtido pelo determinante da matriz $(\lambda I - A)$. No MATLAB, os autovalores de A são obtidos por:

```
>> eig(A)
```

que leva a

```
ans =
   9.0000 + 4.5826i
   9.0000 - 4.5826i
   1.0000
```

Os valores singulares de A são obtidos por:

```
>> svd(A)
```

gerando

```
ans =
   25.2386
    5.4275
    0.7446
```

O resultado do cálculo tanto dos autovalores quanto dos valores singulares no MATLAB, é fornecido por ordem decrescente de valor absoluto.

O posto, ou rank, da matriz A é determinado por:

```
>> rank(A)
```

e obteremos

```
ans =
     3
```

A inversa da matriz A é obtida através do comando

```
≫inv(A)
```

que irá resultar em

```
≫ans =
    0.0784   -0.2157    0.7843
    0.0098    0.0980   -0.4020
         0         0    1.0000
```

Se quisermos obter a inversa da transposta de A, então devemos executar o seguinte comando

```
≫inv(A')
```

e obteremos

```
≫ans =
    0.0784    0.0098         0
   -0.2157    0.0980         0
    0.7843   -0.4020    1.0000
```

Uma matriz nula com 2 linhas e 3 colunas é gerada por:

```
≫nula_23 = zeros(2,3)
nula_23 =
     0     0     0
     0     0     0
```

E uma matriz nula quadrada, 2 por 2, por exemplo

```
≫nula=zeros(2)
nula =
     0     0
     0     0
```

ou seja, basta especificar a ordem da matriz.

De modo semelhante, uma matriz identidade de ordem 3 (3 linhas e 3 colunas) é gerada através do seguinte comando:

```
≫Id=eye(3)
Id =
     1     0     0
     0     1     0
     0     0     1
```

Tarefa. Ler o Apêndice D do livro de *Ogunnaike e Ray (1994). Process Dynamics, Modeling and Control. Oxford University Press Inc.* Usar o "help" do MATLAB e executar os comandos dos exemplos do apêndice da referência acima.

Um recurso também útil e bem utilizado é a criação de identidades e matrizes nulas ou de uns (com todos os elementos iguais a 1) com a mesma dimensão que uma matriz já existente.

Ex: Considere a seguinte matriz

```
≫M=[0.1 2.34 0.22; 1 0 0.15];
```

e para criarmos uma identidade da dimensão do número de linhas de `M`, então devemos executar:

```
≫[nl,nc]=size(M);Id=eye(nl);
```

ou da seguinte forma

```
≫Id=eye(size(M,1));
```

Neste último comando, tomamos o primeiro elemento da matriz resultante de `size`.

Outro recurso largamente utilizado com o MATLAB é a concatenação de matrizes ou vetores. Este recurso é útil quando se deseja montar uma estrutura particular ou empilhar os resultados numa única matriz ao longo de uma seqüência de cálculo.

Ex: Suponha que, por uma necessidade que ficará mais clara nos estudos futuros, seja necessário acrescentar uma matriz nula de dimensão 2x1 à direita da matriz M do exemplo anterior. Isto pode ser feito da seguinte forma:

```
≫Ma=[M zeros(2,1)]
Ma =
    0.1000    2.3400    0.2200         0
    1.0000         0    0.1500         0
```

Logo, fica claro que se pode manipular matrizes e vetores de modo bem conveniente usando o MATLAB.

Para encerrar esta seção, apresenta-se uma forma interessante e útil de se criar vetores. Ao se construir um gráfico, conforme será visto a seguir, é necessário especificar as variáveis dependentes e as independentes. No contexto deste curso, a variável independente que aparece com maior freqüência é o tempo. Então para criarmos um vetor que contenha os valores do tempo, para um dado incremento e valor final, basta criar uma seqüência da seguinte forma:

Ex: ≫`t=0:0.1:1`

```
t =
  Columns 1 through 7
         0    0.1000    0.2000    0.3000
0.4000    0.5000    0.6000

  Columns 8 through 11
    0.7000    0.8000    0.9000    1.0000
```

No exemplo acima, o primeiro valor após ao igual é o ponto inicial, o valor entre dois pontos é o tamanho do incremento e o valor final é o fim da seqüência. Podemos, também, usar variáveis nesse tipo de construção.

Ex: ≫ `t0=0;T=0.1;tfim=10;`
≫ `t=t0:T:tfim*T`

que resultará na mesma seqüência do exemplo anterior. Vale lembrarmos que o valor de `T` pode ser positivo ou negativo, ou seja, pode ser um incremento ou um decremento.

3.0 A utilização de arquivos .m

3.1 Idéia Geral dos arquivos .m

Até este ponto considerou-se que os comandos estivessem sendo executados diretamente no "workspace". Entretanto, esta não é a maneira mais conveniente de se trabalhar com o MATLAB, pois assim tem-se de executar uma instrução por vez. Além disso, ao encerrar o MATLAB tudo o que foi digitado e as variáveis que estavam na memória são perdidos.

De forma semelhante a qualquer linguagem de programação, pode-se construir um arquivo que contenha o código para a implementação da tarefa desejada. No FORTRAN, por exemplo, os arquivos-fonte são identificados pela extensão ".for" que, após a compilação e o link, geram um executável identificado pela extensão ".exe". No MATLAB, os arquivos-fonte são identificados pela extensão ".m". Ao contrário do FORTRAN, C/C++, Pascal etc., o MATLAB não gera um código executável e os arquivos ".m" são executados a partir da linha-de-comando do "workspace" simplesmente digitando o nome do mesmo.

Os arquivos ".m" contém o conjunto de comandos que definem variáveis, executam funções do MATLAB ou aquelas criadas pelo usuário, conforme será visto a seguir. De agora em diante, todos os exemplos serão considerados como arquivos ".m' e será indicado o nome do arquivo respectivo.

Nas versões do MATLAB anteriores a 5, não existe um editor de programas incorporado ao "workspace". Dessa forma, é necessário que se faça uso de editores que não gerem caracteres gráficos, tais como o NOTEPAD do Windows, dentre outros.

Pode-se dividir o código em diversos arquivos ".m", podendo um ser chamado dentro de outro. Por exemplo: o arquivo `entrada.m` define os dados de entrada e dentro dele é chamado o arquivo `principal.m`, que por sua vez executa uma série de instruções e chama o arquivo `grafico.m`.

Para executar um arquivo ".m" na linha-de-comando ou para chamar um dentro de outro, basta digitar o nome do arquivo sem a extensão .m. Os nomes dos arquivos ".m" não devem exceder a oito (8) caracteres (nas versões anteriores à 5.0), sem acentuação ou cedilha. Devem estar armazenados no sub-diretório de trabalho corrente. A última seção do texto traz informações como indicar um caminho na linha de comando do MATLAB.

Uma última informação geral sobre os arquivos ".m' é quanto ao "status" das variáveis. Estas são sempre globais, não importando se foram definidas dentro de um outro arquivo ".m". As variáveis, no MATLAB, somente deixam de ser globais e passam a ser locais quando criamos "functions", conforme será visto na próxima seção. Uma variável local não fica disponível fora da "function", a menos que ela seja um parâmetro de saída ou entrada.

3.2 Construindo `functions`

Não se deve entender os diversos arquivos ".m" citados na seção anterior como o equivalente a subrotinas do FORTRAN. Esta eqüivalência, no MATLAB, corresponde às `functions`. Tais sub-programas também devem possuir a extensão ".m", mas com algumas peculiaridades. Os comandos e funções que constituem uma nova `function` devem obrigatoriamente estar contidos num arquivo com o mesmo nome que a `function` e extensão ".m". A primeira linha desse arquivo deve conter a sintaxe para uma nova `function`. Por exemplo, um arquivo denominado "stat.m", que implementa o cálculo da média e desvio-padrão de um vetor `x` pode ser escrito como:

```
function [mean,stdev] = stat(x)
n = length(x);
mean = sum(x)/n;
stdev = sqrt(sum((x - mean).^2)/n);
```

O cabeçalho do arquivo function.m deve sempre conter `function` seguido da lista de argumentos de saída, que devem ter o mesmo nome das variáveis correspondentes no corpo da `function` e do lado direito da igualdade o nome da `function` e a lista de parâmetros de entrada da mesma.

A fim de não tornar estas notas muito extensas, foi incluído no final do texto um exemplo mais claro da utilização de `functions`, onde é mostrada a implementação de um código para a solução de um conjunto de três equações diferenciais ordinárias não lineares, oriundo da modelagem de um reator químico.

4.0 Programando no MATLAB.

Mesmo que se utilize os diversos programas disponíveis no MATLAB, sempre é necessário que elaborar um programa, por mais simples que seja, para a resolução dos problemas em questão. Assim, é importante conhecer o básico de alguns comandos de

programação no MATLAB. Antes de continuar com a descrição, vale comentar que o MATLAB foi concebido em "C" e, portanto, guarda uma semelhança com a sintaxe desta linguagem.

4.1 O comando `for`

A manipulação de laços não condições no MATLAB é feita pelo comando `for`, que pode ser melhor entendido pelo seguinte exemplo.

Ex: Soma dos números naturais de 1 a 10.

```
numeros = []; % criando um vetor vazio
soma=0; % zerando a variável soma
for in=1:10
  soma = soma + in;
% acumuando a soma no vetor numeros
  numeros = [numeros;soma];
end
```

Então, após digitar o código acima e salvá-lo com o nome "soman.m" no subdiretório de trabalho, basta executar, na linha-de-comando

```
≫ soma
soma =
    55
```

e

```
≫ numeros
numeros =
     1
     3
     6
    10
    15
    21
    28
    36
    45
    55
```

No comando `for` precisamos usar um contador, no caso `in`. O primeiro valor indica o princípio e o último o fim. Existe, também, a possibilidade de usarmos três argumentos, onde o central é o incremento. A seguir vão os comandos a serem executados. Um `for` deve ser sempre fechado por um `end`. Pode-se incluir um `for` dentro de outro.

4.2 O comando `if`

O `if`, ou o condicional do MATLAB, é bem semelhante aos das demais linguagens. É preciso que se especifique, apenas, os operadores relacionais usados neste comando.

<	Menor que	==	igual a
<=	Menor que ou igual a	~=	diferente de
>	Maior que		
>=	Maior que ou igual a		

Existe, também, o laço condicional `while`, muito semelhante ao do "C" e FORTRAN. Para este comando sugere-se ao usuário que acione o "help" na linha de comando através de:

```
≫ "help" while
```

Os operadores lógicos que podem ser combinados tanto nos `if` quanto nos `while` são:

&	and
|	or
~	not

6.0 Usando as funções `ode23` e `ode45`

Nesta seção, será descrita a utilização das rotinas para solução de equações diferenciais ordinárias, `ode23` e `ode45`. Tais códigos implementam as fórmulas de Runge-Kutta. O primeiro resolve o sistema de EDO por um método de segunda ordem fazendo a extrapolação para o cálculo do erro relativo com um método de terceira ordem. Já o `ode45` implementa um algoritmo de quarta ordem e faz a extrapolação com um método de quinta ordem. `ode45` gera resultados mais precisos que `ode23`.

Sintaxe:

```
[T,Y] = ode45(F, [T0, Tfinal], Y0)
```

`Entrada:`

`F`	nome da `function` onde está definido os sistema de EDO. Chamada: `yprime = fun(t,y)` onde `F = 'fun'`.
`t`	tempo (escalar).
`y`	solução (vetor coluna).
`yprime`	retorna um vetor coluna com o lado direito da EDO, ou seja: `yprime(i) = dy(i)/dt`.
`t0`	valor inicial de t.
`tfinal`	valor final de t.
`y0`	vetor coluna com as condições iniciais do sistema.

`OUTPUT:`

`T`	vetor coluna. Retorna cada ponto (passo) de integração no intervalo `[t0,tfinal]`.
`Y`	retorna uma matriz com o número de linhas igual ao número de passos de integração no intervalo `[t0,tfinal]` e o número de colunas igual ao número de variáveis dependentes do problema.

7.0 Construção de Gráficos

A construção de gráficos no MATLAB é fácil e um dos atrativos para a utilização deste pacote. Pode-se construir gráficos de uma única curva ou de múltiplas, colocar mais de um gráfico na mesma figura, escolher o tipo de símbolo e as cores, formatar eixos e títulos de eixos e gráficos.

O comando mais simples é o `plot`. Por exemplo, para construir o gráfico de um vetor `x` de n posições basta executar:

```
plot(x)
```

O gráfico resultante terá o eixo das abscissas como uma seqüência de 0 a n espaçados de 1 em 1. Caso seja necessário especificar o eixo das abscissas deve-se construir um vetor com essas informações e este deve ter a mesma dimensão que o das ordenadas.

Ex:

```
x=0:0.1:1;
y=x.^(1/3);
plot(x,y)
```

Nesse caso o MATLAB irá colocar a cor padrão para a curva. Caso se queira escolher a cor e o símbolo, deve-se acrescentar essa informação dentro do `plot`, após `x,y`, ou seja:

```
plot(x,y,'r-')
```

onde "'" é o apóstrofo e o gráfico será construído com uma linha contínua vermelha. O comando "`help`" `plot` trás a lista dos símbolos e cores possíveis.

Para especificar o título dos eixos, deve-se acrescentar o seguinte comando, logo abaixo do `plot`:

```
xlabel('nome_do_eixo_x')
ylabel('nome_do_eixo_y')
```

Também é possível fixar o comprimento dos eixos. Assim, suponha que `x` varie de 0 a 9.5 e `y` de -1.35 a 2.45. Para uma boa visualização pode-se fixar os eixos nos seguintes limites:

```
eixo_x = [0, 10] e
eixo_y = [-1.5, 2.5]
```

e, então definir o comando que fixa os eixos:

```
axis = ([eixo_x eixo_y])
```
ou simplesmente
```
axis = ([0,10,-1.5,2.5])
```

Logo, `axis` é um vetor linha com os limites do eixo x nas duas primeiras posições e os do eixo y nas duas últimas.

Para se construir múltiplas curvas na mesma figura deve-se usar o comando `subplot(???)`, onde `???` é uma seqüência de números que especificam:

primeiro `?`: o número de gráficos na vertical da mesma figura. Se queremos dois gráficos, então ?=2. Se três, então ?=3.

segundo `?`: o número de gráficos na horizontal da mesma figura.

terceiro `?`: numera seqüencialmente os gráficos.

Ex.:

`subplot(211)`
`subplot(212)`

para plotar na mesma figura, um gráfico abaixo do outro.

`subplot(221)`	`subplot(222)`
`subplot(223)`	`subplot(224)`

para plotar quatro gráficos na mesma figura.

`subplot(211)`	
`subplot(223)`	`subplot(224)`

para plotar três gráficos na mesma figura.

O comando `subplot` deve sempre vir antes do `plot`. O exemplo final vai ajudar a compreensão dessa seqüência de comandos.

Para encerrar esta seção, mostra-se como construir múltiplas curvas dentro de um mesmo gráfico. Para tanto, basta colocar a nova seqüência de variáveis logo após o final da primeira dentro do mesmo `plot`, ou seja:

```
plot(x1,y1,'r-', x2,y2,'b:')
```

onde a curva relativa a (x1,y1) é vermelha e contínua e a de (x2,y2) é azul e tracejada.

8.0 Exemplo de um CSTR não-isotérmico

Ref: LUYBEN, W.L. (1990). Process Modeling Simulation and Control for Chemical Engineers, Second Ed., McGraw-Hill, New York.

Uma reação exotérmica irreversível:

$$A \xrightarrow{k} B$$

é realizada num CSTR, conforme mostra a Fig. 1. A reação é de primeira ordem em relação ao reagente A e tem um calor de reação λ_r. Assume-se as hipóteses de perda de calor desprezáveis, densidades constantes e mistura perfeita. Uma camisa de resfriamento envolve o reator para retirar o calor. Água resfriada circula na camisa a uma vazão F_J e com a temperatura de entrada T_{JO}. O reator tem um volume V e a camisa de arrefecimento um volume de água V_J considerado constante. Considera-se que a água na camisa tenha mistura perfeita. Admite-se regime permanente para as vazões de entrada e saída.

A constante da taxa de reação é uma função da temperatura, de acordo com a expressão (Arrhenius):

$$k = C_1 e^{-E/RT} \qquad (1)$$

Realizando os balanços dinâmicos de massa e energia para o reator e camisa de arrefecimento, chega-se ao seguinte sistema de EDO:

$$\frac{dC_A}{dt} = \tau(C_{Ao} - C_A) - k\ C_A \qquad (2)$$

$$\frac{dT}{dt} = \tau(T_o - T) - \omega\, k\, C_A - \gamma\,(T - T_J) \qquad (3)$$

$$\frac{dT_J}{dt} = \tau_J(T_{Jo} - T_J) + \gamma_J(T - T_J) \qquad (4)$$

com:

$$\omega = \frac{\lambda_r}{\rho\, Cp} \qquad (5)$$

$$\gamma = \frac{U\, A_H}{\rho\, Cp} \qquad (6)$$

$$\gamma_J = \frac{U\, A_H}{\rho_J\, C_J\, V_J} \qquad (7)$$

$$\tau = \frac{F_0}{V} \qquad (8)$$

$$\tau_J = \frac{F_J}{V_J} \qquad (9)$$

As Tabelas 1 e 2 trazem os valores dos parâmetros para este exemplo.

Tabela 1: Condições de regime permanente para o modelo do CSTR não isotérmico

	T(°R)	T_J(°R)	C_A(lbmol/ft^3)	C_A(mol/l)
Inferior	537,16	536,62	0,4739	7,5913
Intermediário	600,00	594,63	0,2451	3,9262
Superior	651,06	641,79	0,0591	0,9467

Tabela 2: Parâmetros para o CSTR não-isotérmico

V_J = 0,109 m^3	C_1 = 7,08x10^{10} h^{-1}
E = 6,9778x10^4 J/mol	λ_r = - 6,9778x10^4 J/mol
U = 3065580 J/h m^3 K	A_H = 7,25 m^2
T_{jo} = 294,26 K (21,1 °C)	V = 48 ft^3
C_p = 3,14 J/g K	C_J = 4,1867 J/g K
ρ = 800950 g/m^3	ρ_J = 997984 g/m^3

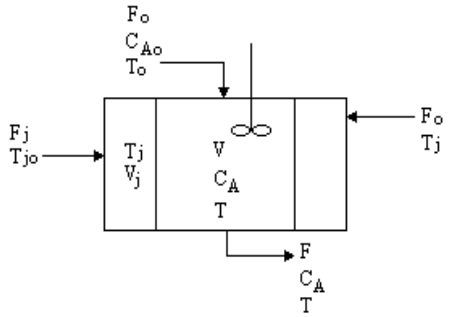

Fig. 1: Esquema do CSTR Não Isotérmico.

Arquivo `entrada.m`

```
% Clear the "workspace", funtions and globals.
Clear; clear globals; clear functions

global CAO TO TJO omega gama gamaJ tau tauJ C1 E

% Simulation label.
sinton = '012608';

% Setup of the initial time, the sampling period and the end time of simulation.
t0     =           0 ; % Start time.
tamost =         0.1 ;  % Sampling period [min].
tout   =      tamost ; % Final time of integration.
tfim   =         6.0 ; % Final time of simulation = (tfinal*tamost) [min].
tfinal = tfim/tamost ;

% Start vectors.
temp   = [];
conc   = [];
vett   = [];

% Set parameters. See tab. 5 pag. 125 of Luyben (1990).
C1     = 7.08e10;
Rg     = 1.99;
E      = 30000/Rg;
CAO    = 0.5;
FO     = 40;
TO     = 530;
TJO    = 530;
V      = 48;
```

```
VJ     = 3.85;
FJ     = 49.9;
U      = 150;
AH     = 250;
LAMBDA = - 30000;
RO     = 50;
CP     = 0.75;
ROJ    = 62.3;
CJ     = 1.0;
gamaJ  = U*AH/(ROJ*CJ*VJ);
gama   = U*AH/(RO*CP*V);
omega  = LAMBDA/(RO*CP);
tau    = FO/V;
tauJ   = FJ/VJ;

% Setting initial conditions of the states.
 CA = 0.4739; T  = 537.16; TJ = 536.62; %  Lower steady state.
% CA = 0.2451; T  = 600.00; TJ = 594.63; % Intermediary steady state.
% CA = 0.0591; T  = 651.06; TJ = 641.79; % Upper steady state.

x(1,1) = CA; x(2,1) = T; x(3,1) = TJ;

% Store the state variables values in to the initial conditions.
conc = [conc  ; x(1,1)];
temp = [temp  ; x(2,1)];
tref = [tref  ; x(3,1)];
vett = [vett  ;   0   ];

xold = x;

% Call the main program.
main

% Save the "workspace" with label "sinton".
eval(['save ', sinton])
```

Arquivo `main.m`

```
                                        % Main program.

%%%%%%%%%%%%%%%%%% Main Loop %%%%%%%%%%%%%%%%%%%%
for ii = 1: tfinal,

  % Print the sampling instant.
  ts1 = sprintf('t = %5.2f h', ii*tamost);
  disp(ts1)

  % Integrating the system.
  Y0      = xold;
  [t,Y]   = ode45('model',[t0 tout],Y0);
  [nl,nc] = size(Y);
  x       = Y(nl,:)'; % Get the last line of Y, which contains
                      ; % the value of state variable at tout.

  % Store the states.
  conc   = [ conc   ; x(1,1)    ];
  temp   = [ temp   ; x(2,1)    ];
  tref   = [ tref   ; x(3,1)    ];
  vett   = [ vett   ; ii*tamost ];

  % Build the graphics.
  figure(1)
  grafico
  drawnow

  % Reset the time and go to the next iteration.
  t0   = tout;
  tout = tout + tamost;
  xold = x;
end

% Building the last graphics.
figure(1)
grafico
drawnow
```

Arquivo `model.m`

```
function dY = model(t,Y)

global CAO TO TJO omega gama gamaJ tau tauJ C1 E

CA = Y(1,:);
T  = Y(2,:);
TJ = Y(3,:);

k = C1*exp( -E/T );

f1 =  tau*(CAO - CA) - k*CA;
f2 =  tau*(TO  - T ) - omega*k*CA - gama*(T - TJ);
f3 =  tauJ*(TJO - TJ) + gamaJ*( T - TJ );

dY  = [f1;
       f2;
       f3];
```

Arquivo `grafico.m`

```
% Build the graphics of controlled and manipulated variables.
clf
%
subplot(311)
plot(vett,temp,'r-')
ylabel('T (oR)')
title(' Reactor Temperature')
%
subplot(312)
plot(vett,conc,'r-')
ylabel('C (lbmol/ft3)')
title('Concentration in the reactor')
%
subplot(313)
plot(vett,tref,'r-')
ylabel('TJ (oR)')
xlabel('Time [h]')
title('Coolant Temperature')
```

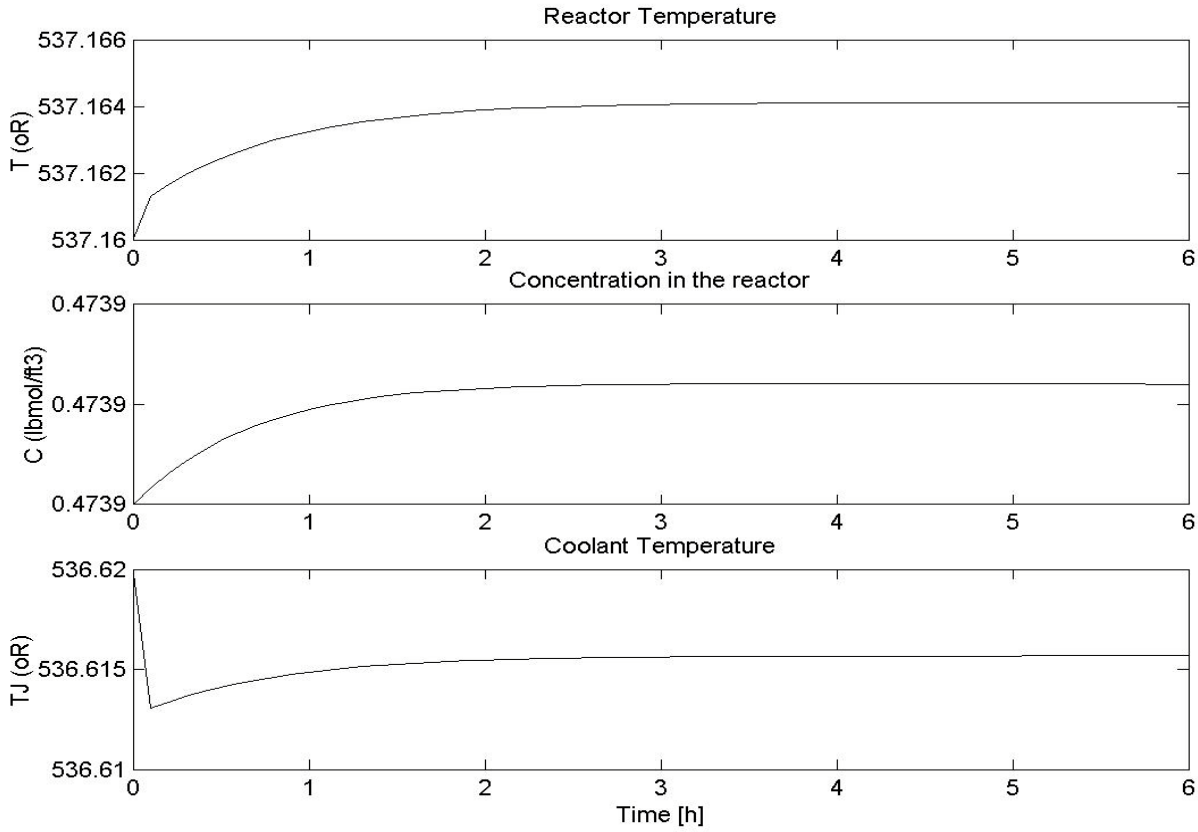

Figura 2: Resultados para a condição inicial na condição de regime permanente inferior
Obs: atenção à escala do gráfico.

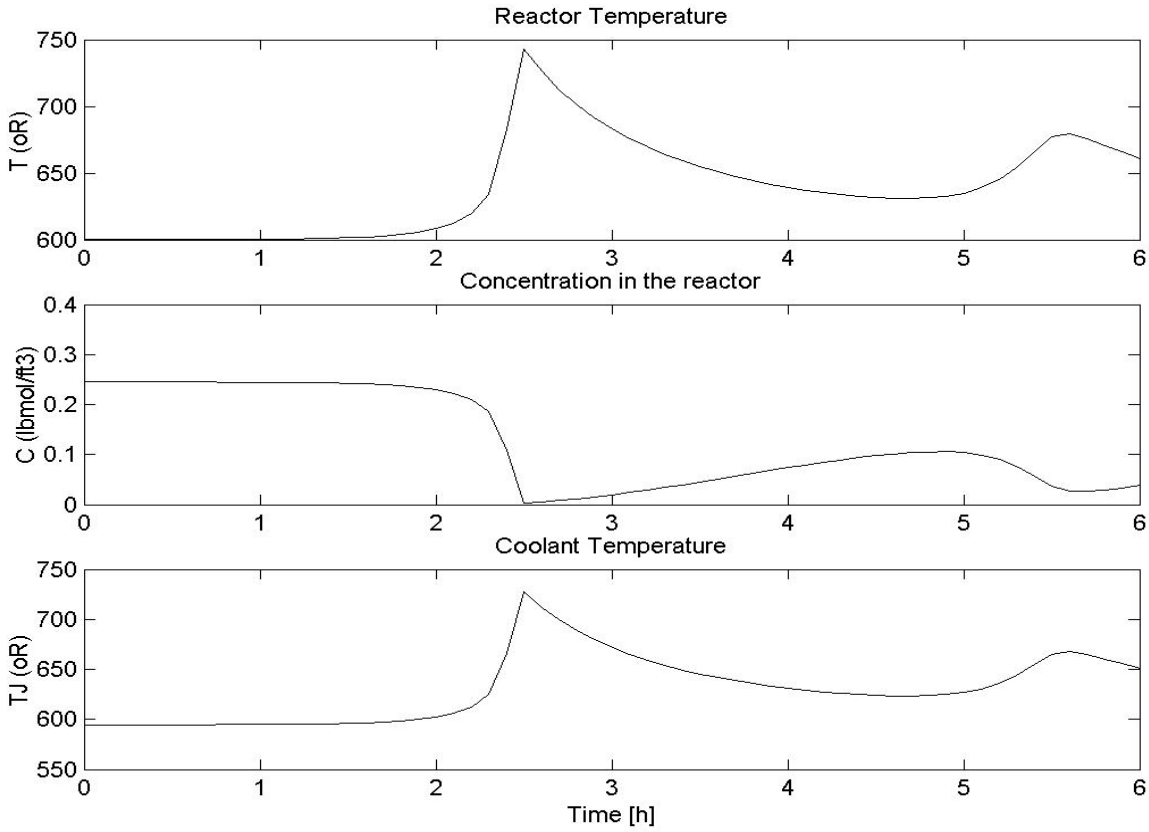

Figura 3: Resultados para a condição inicial na condição de regime permanente intermediário.

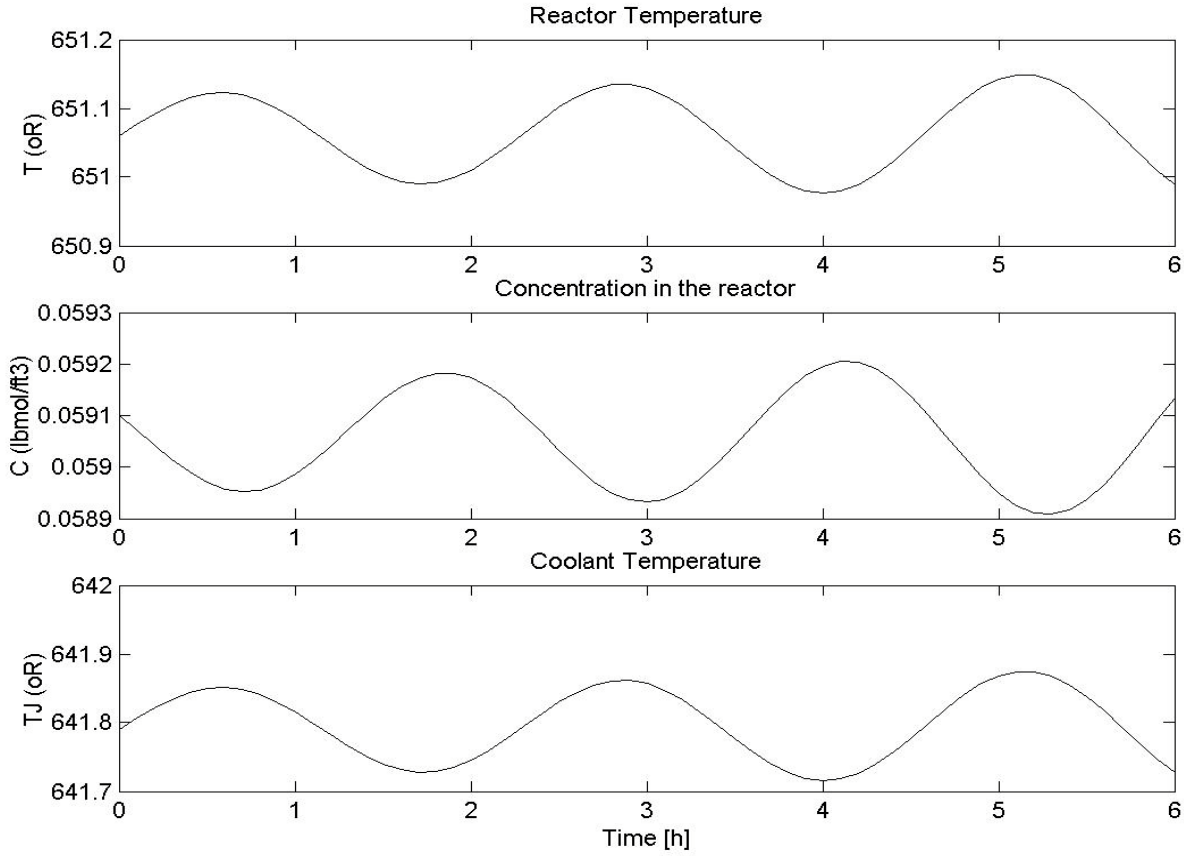

Figura 4: Resultados para a condição inicial na condição de regime permanente superior.