Introdução ao GNU Octave

Valdenir de Souza Junior

Faculdades Integradas de Caratinga – FIC

Caratinga **Set. 2006**

Sumário

1 Introdução

Esse guia apresenta os conceitos básicos do GNU Octave, uma importante ferramenta de cálculo científico que tem a vantagem de ser software livre. O guia mostra como usar o Octave para fazer cálculos com escalares, vetores e matrizes, traçar gráficos e escrever programas simples.

Maiores informações sobre o Octave podem ser encontradas no manual preparado por Eaton (1997), disponível impresso ou em versão online. Guias do Matlab também podem ser úteis, embora haja variações entre os dois programas. Para maiores detalhes sobre o traçado de gráficos no Octave, a documentação do GNUPLOT (Crawford, 1998) pode ser uma boa referência.

1.1 O Octave e outros programas de cálculo científico

Os programas de cálculo científico podem ser subdivididos em duas categorias, os de *cálculo simbólico* e os de *cálculo numérico*. Os programas de cálculo simbólico trabalham com grandes números inteiros, procurando fornecer respostas exatas para os problemas. Os programas de cálculo numérico trabalham com números de ponto flutuante, fornecendo resultados dentro de uma aproximação.

Embora os programas de cálculo simbólico também possam ser usados para resolver problemas matemáticos numericamente, para os casos em que a manipulação simbólica de expressões algébricas não é requerida, os programas de cálculo numérico são mais eficientes, fornecendo resultados mais rapidamente. Um resumo, com alguns programas de cálculo científico encontrados no mercado é mostrado na Tabela 1.

Tabela 1: Algumas ferramentas de cálculo científico.

Cálculo numérico		
Matlab	Software proprietário	http://www.mathworks.com/
Octave	Software livre	http://www.octave.org/
Scilab	Software livre	http://scilabsoft.inria.fr/
Cálculo simbólico		
Derive	Software proprietário	http://education.ti.com/us/product/software.html/
Maple	Software proprietário	http://www.maplesoft.com/
Mathematica	Software proprietário	http://www.wolfram.com/
Maxima	Software livre	http://maxima.sourceforge.net/
Mupad	Software proprietário	http://research.mupad.de/

O programa Octave se insere na categoria dos programas de cálculo numérico. O Ocatave é um software livre, escrito por Eaton (1997) e vários outros colaboradores, estando disponível sob os termos da Licença Pública Geral do GNU (GPL) (Free Software Foundation, 1991).

O programa provê uma interface por linha de comandos para a solução numérica de problemas lineares e/ou não lineares e para implementar outros

experimentos numéricos usando uma linguagem que é compatível com o programa comercial Matlab. O programa pode ser utilizado também em modo *script* (textos de programação) e permite incorporar módulos escritos nas liguagens C++, C, Fortran e outras.

O Octave tem ferramentas amplas para soluções numéricas de problemas comuns de álgebra linear, para a determinação de raízes de equações, manipulações polinomiais e integração de equações diferenciais ordinárias e equações diferencias algébricas.

Programas como o Octave são usados freqüentemente no lugar de linguagens de programação científica como o C ou Fortran, por já trazerem embutidas muitas ferramentas numéricas e permitirem a visualização gráfica dos resultados de forma mais fácil.

1.2 Iniciando o Octave

Nos sistemas Unix o Octave é iniciado digitando o comando `octave`. O programa então mostra uma mensagem inicial e um sinal de prontidão (`>>`), indicando que está pronto para aceitar comandos:

```
GNU Octave, version 2.1.50 (i686-pc-cygwin).
Copyright (C) 1996, 1997, 1998, 1999, 2000, 2001, 2002, 2003 John W. Eaton.
This is free software; see the source code for copying conditions.
There is ABSOLUTELY NO WARRANTY; not even for MERCHANTIBILITY or
FITNESS FOR A PARTICULAR PURPOSE.  For details, type 'warranty'.

Please contribute if you find this software useful.
For more information, visit http://www.octave.org/help-wanted.html

Report bugs to <bug-octave@bevo.che.wisc.edu>.
>>
```

Para finalizar o programa, digita-se `quit` ou `exit`.

1.3 Operações básicas com o Octave

A forma mais simples de trabalhar com o Octave é digitar os comandos matemáticos, como em uma calculadora normal. Por exemplo, digitando

```
>> 2+3
```

e teclando Enter, tem-se o resultado

```
ans = 5
```

O valor calculado é exibido e guardado na variável `ans`, do inglês *answer* (resposta), e pode ser usado no cálculo seguinte

```
>> ans+10
ans = 15
```

Tabela 2: Operadores aritméticos.

Operador	Operação
`+`	Adição
`-`	Subtração
`*`	Multiplicação
`/`	Divisão
`^`	Potenciação. Ex.: `2^3` resulta: `ans = 8`
`\`	Divisão à esquerda. Ex.: `5\25` tem o mesmo efeito que `25/5` e resulta: `ans = 5`

As operações aritméticas básicas são obtidas pelos operadores indicados na Tabela 2.

A ordem com que as operações são feitas é a usual, ou seja, parênteses são calculados primeiramente, seguidos de potência, multiplicação e divisão e, finalmente, adição e subtração.

O Octave possui uma série de funções matemáticas, sendo algumas delas apresentadas na Tabela 3.

Tabela 3: Algumas funções matemáticas.

Função	Descrição
`abs(x)`	Módulo de x
`acos(x)`	Arco cujo coseno é x
`cos(x)`	Coseno de x (x em radianos)
`cosh(x)`	Cosseno hiperbólico de x
`exp(x)`	Função exponencial e^x
`imag(x)`	Parte imaginária de um complexo
`log(x)`	Logaritmo natural de x (base e)
`log10(x)`	Logaritmo de x na base 10
`real(x)`	Parte real de x
`round(x)`	Arredonda o valor de x
`sign(x)`	Sinal do número (-1 ou $+1$)
`sin(x)`	Seno de x (x em radianos)
`sinh(x)`	Seno hiperbólico de x
`sqrt(x)`	Raiz quadrada de x
`tan(x)`	Tangente de x

Por exemplo, para calcular $1.2\sin(40^\circ + \ln(2.4^2))$, digita-se

```
>> 1.2 * sin(40*pi/180 + log(2.4^2))
ans = 0.76618
```

Conforme pode-se notar, os ângulos usados como argumentos nas funções trigonométricas devem estar em radianos. Para fazer a transformação de graus para radianos, pode-se multiplicar o valor por $\pi/180$. A operação de multiplicação deve ser sinalizada de forma explícita pelo uso do operador `*`, como indicado entre `1.2` e `sin`.

2 A área de trabalho

Conforme visto, o Octave tem uma interface baseada em linha de comando, onde os comandos são digitados, seguidos pela digitação da tecla Enter. O Octave é uma linguagem interpretada, o que significa que cada comando é convertido em código de máquina e executado, por vez. No caso de linguagens compiladas, o programa inteiro é convertido em código de máquina previamente, para depois ser executado. De forma geral os programas compilados são executados de forma mais rápida que os interpretados.

2.1 Definição de variáveis

É possível definir variáveis a serem usadas na sessão de trabalho, como segue:

```
>> a=3
a = 3
```

Pressionando Enter Octave confirma na tela o valor atribuído, a menos que seja colocado um caractere de ponto-e-vírgula (`;`) no final do comando

```
>> b=2.5;
>>
```

Pode-se digitar vários comandos em uma mesma linha, separando-os por vírgula ou por ponto e vírgula:

```
>> a=3, b=2.5; c=7.5
a = 3
c = 7.5000
```

Na definição dos nomes das variáveis, Octave diferencia letras maiúscula de minúsculas. Por exemplo, `a` é diferente de `A`. Uma vez definidas as variáveis, pode-se efetuar operações com as mesmas

```
>> a+b
ans = 5.5000
```

É possível definir números complexos especificando sua parte imaginária por meio da variável pré-definida `i`:

```
>> c=3+2i
c = 3 + 2i
```

Além da variável `i`, que como é de se esperar vale $\sqrt{-1}$, e da variável `ans`, que toma o resultado do último cálculo realizado, outras variáveis pré-definidas no Octave são `pi`, que vale 3.14159... e `j`, que assim como `i` vale $\sqrt{-1}$. Essa variáveis podem ser redefinidas pelo usuário, mas para evitar enganos é melhor não fazê-lo. Da mesma forma, não é recomendável dar às variáveis nomes de funções, como `sin` e `cos`.

Para obter uma relação das variáveis nomeadas na seção de trabalho, digita-se

```
>> who

*** dynamically linked functions:

dispatch

*** local user variables:

a  b  c
```

Para remover da área de trabalho uma variável já atribuída, usa-se o comando `clear`, seguido do nome da variável, como em

```
>> clear c
```

Para apagar todas as variáveis, digita-se

```
>> clear all
```

2.2 Formatação e precisão numérica

O Octave normalmente exibe os números com seis algarismos significativos. Apesar de exibir os números dessa forma, o Octave trabalha internamente com uma precisão bem maior. Por isso, freqüentemente é conveniente guardar os resultados em variáveis, no lugar de digitar novamente os valores mostrados, como forma de evitar a introdução de erros nos resultados.

O comando `format` permite selecionar a forma com que os algarismos são mostrados. Digitando `format long` o Octave passa a exibir os valores com pelos menos 16 algarismos significativos nas respostas dos cálculos efetuados.

```
>> c=1/3
c = 0.33333
>> format long
>> c
c = 0.333333333333333
```

O comando `format` sem parâmetros faz o programa retornar ao seu modo normal de exibição

```
>> format
>> c=1/3
c = 0.33333
```

Além dos números reais e complexos mostrados, outros números são reconhecidos e calculados pelo Octave:

Infinito (Inf) Resultado da divisão de um número por zero. Esta é uma resposta válida e pode ser usada nos cálculos e atribuída a uma variável, assim como outro número qualquer.

Not a Number (NaN) Resultado da divisão de zero por zero e outras operações que geram resultados indefinidos. Novamente, os resultados podem ser tratados como qualquer outro número, apesar dos resultados dos cálculos com seu uso gerarem sempre a resposta `NaN`.

Considere o cálculo

```
>> 1 - 0.2 - 0.2 - 0.2 - 0.2 - 0.2
ans =  5.5511e-17
```

O resultado fornecido é bem pequeno, mas não chega a ser exatamente zero, que é a resposta correta. A razão para isto é que 0.2 não pode ser representado em binário usando um número finito de dígitos (em binário fica 0.0011001100...). Os programas de computador procuram representar esses números da forma mais próxima possível, mas o uso repetido dessas aproximações pode causar problemas.

2.3 Repetindo comandos anteriores

Octave mantém um registro de todos os comandos que digitados durante a sessão, e pode-se usar as teclas de direção $\boxed{\uparrow}$ e $\boxed{\downarrow}$ para rever os comandos precedentes (com o primeiro sendo o mais recente). Para repetir um desses comandos, basta achá-los com essas teclas e pressionar $\boxed{\text{Enter}}$.

Ao procurar um comando precedente particular, sabendo as primeiras as primeiras letras da linha digitada, pode-se pressionar essas letras e então pressionar $\boxed{\uparrow}$, de forma a encontrar todas as linhas precedentes começando com tais letras.

Uma vez que um comando foi recuperado, pode-se editá-lo antes de executálo outra vez. Pode-se pode usar $\boxed{\leftarrow}$ e $\boxed{\rightarrow}$ para mover o cursor através da linha, e digitar outros caracteres ou usar a tecla $\boxed{\text{Delete}}$ para apagar os caracteres.

2.4 Outros comandos úteis

O Octave possui um sistema de ajuda integrado, que pode ser bastante útil para obter maiores informações sobre um comando ou encontrar uma função em particular. A forma básica de usar a ajuda é digitar

```
>> help nome_do_comando
```

Em algumas situações pode ser desejável interromper a execução de comandos que porventura estejam levando muito tempo para serem concluídos. O comando corrente pode ser interrompido pressionando simultaneamente as teclas

$\boxed{\text{Ctrl}}$ $\boxed{\text{C}}$

3 Vetores e matrizes

O Octave permite trabalhar com vários tipos de dados. Números reais ou complexos, variáveis booleanas (lógicas), strings (seqüências de caracteres) são exemplos de dados considerados como *escalares*. Objetos escalares podem ser agrupados em listas, constituindo os vetores e matrizes.

3.1 Vetores

No Octave os vetores são criados colocando-se seus componentes entre colchetes (`[ ]`). Os elementos de um vetor-coluna são separados por ponto-e-vírgula (`;`). Assim, para definir o vetor

$$\mathbf{v}_1 = \left\{ \begin{array}{c} 1 \\ 2 \\ 3 \end{array} \right\}$$

digita-se

```
>> v1=[1; 2; 3]
v1 =

        1
        2
        3
```

Outra forma possível é separar os elementos em cada linha, digitando Enter após cada elemento, com o último seguido de um fechamento de colchetes (`]`):

```
>> v1=[1
2
3]
v1 =

        1
        2
        3
```

Um vetor-linha é definido separando seus elementos por espaço ou por vírgula. Por exemplo, para definir o vetor-linha

$$\mathbf{v}_2 = \left\{ \begin{array}{ccc} 4 & 5 & 6 \end{array} \right\}$$

pode-se digitar

```
>> v2=[4,5,6]
v2 =

        4        5        6
```

ou

```
>> v2=[4 5 6]
v2 =

        4        5        6
```

Os vetores $\mathbf{v}_1$ e $\mathbf{v}_2$ tiveram seus elementos digitados um por um. Uma outra forma de definir vetores é especificar os valores da forma

```
nome_do_vetor = valor_inicial : valor_final
```

Dessa forma são dados incrementos unitários entre o valor inicial e valor final. Por exemplo

```
>> a=1:5
a =

  1  2  3  4  5
```

Outra maneira é especificar os valores da forma

```
nome_do_vetor = valor_inicial : incremento : valor_final
```

Por exemplo, para definir o vetor **b**, com valores de 1 a 9, variando de dois em dois, digita-se

```
>> b=1:2:9
b =

  1  3  5  7  9
```

O comando `linspace(i,s,n)` permite criar um vetor com uma quantidade de elementos iguais a `n`, com valores entre entre o limite inferior `i` e o limite superior `s`. Por exemplo, o vetor **a** poderia ser definido por

```
>> a=linspace(1,5,5)
a =

        1        2        3        4        5
```

O comando linspace sem o último argumento fornece um vetor de 100 elementos como padrão. O tamanho de um vetor pode ser verificado com o comando `length`, como no exemplo abaixo:

```
>> b=linspace(1,5);
>> length(b)
ans = 100
```

De maneira similar o comando `logspace(i,s,n)` fornece `n` elementos entre 10^i e 10^s. A ausência do terceiro argumento, `n`, fornece 50 elementos entre 10^i e 10^s.

Um vetor pode ser transposto usando-se o apóstrofo (') como operador de transposição.

```
>> a=[1 2 3], c=a'
a =

        1        2        3

c =

        1
        2
        3
```

3.2 Operações com vetores

Para multiplicar todos os números em um vetor por um número, basta usar o operador `*`, como a seguir:

```
>> a=[1 3 5]; b=2*a
b =

        2        6       10
```

O mesmo também é válido para a divisão, com o uso do operador `/`. Também é possível adicionar um mesmo número a cada elemento usando os operadores `+` ou `-`, embora esta não seja uma convenção matemática padrão.

A multiplicação de dois vetores entre si segue as regras da multiplicação matricial. Para que seja possível multiplicar uma matriz $\mathbf{A}$, de ordem $m_1 \times n_1$, por uma matriz $\mathbf{B}$, de ordem $m_2 \times n_2$, é preciso que as dimensões n_1 e m_2 tenham o mesmo valor. Por exemplo, uma matriz $\mathbf{A}$ de 3 colunas só pode ser multiplicada por uma outra matriz $\mathbf{B}$, se essa tiver 3 linhas.

Dados dois vetores $\mathbf{a}$ e $\mathbf{b}$, de n linhas, o *produto escalar* entre esses dois vetores é definido como

$$\mathbf{a} \bullet \mathbf{b} = \mathbf{a}^T\mathbf{b} = \left\{ \begin{array}{c} a_1 \\ a_2 \\ \vdots \\ a_n \end{array} \right\}^T \left\{ \begin{array}{c} b_1 \\ b_2 \\ \vdots \\ b_n \end{array} \right\} = a_1b_1 + a_2b_2 + \ldots + a_nb_n$$

Assim, como exemplo, é definido a seguir os vetores $\mathbf{a}$ e $\mathbf{b}$ e realizado o produto escalar entre os mesmos

```
>> a=[1; 3; 5]; b=[1; 2; 3]; a'*b
ans = 22
```

Uma forma mais elegante e simples de realizar o produto escalar entre dois vetores é usar a função `dot`:

```
>> a=[1; 3; 5]; b=[1; 2; 3]; dot(a,b)
ans = 22
```

Um outro tipo de operação, definida entre vetores de três elementos, é o *produto vetorial*. Essa operação, que resulta em um terceiro vetor, é definida como

$$\begin{aligned}\mathbf{a}\times\mathbf{b} &= \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{vmatrix} \\ &= (a_2b_3 - a_3b_2)\mathbf{i} + (a_3b_1 - a_1b_3)\mathbf{j} + (a_1b_2 - a_2b_1)\mathbf{k} = \begin{Bmatrix} a_2b_3 - a_3b_2 \\ a_3b_1 - a_1b_3 \\ a_1b_2 - a_2b_1 \end{Bmatrix}\end{aligned}$$

A função `cross` permite realizar o produto vetorial entre dois vetores. Por exemplo

```
>> a=[1 1 0]; b=[0 1 1]; cross(a,b)
ans =

     1    -1     1
```

A multiplicação *elemento por elemento* de dois vetores **a** e **b**, com três elementos cada é feita com o operador `.*`. Nesse caso, a operação é dada por

$$\begin{Bmatrix} a_1 \\ a_2 \\ a_3 \end{Bmatrix} .* \begin{Bmatrix} b_1 \\ b_2 \\ b_3 \end{Bmatrix} = \begin{Bmatrix} a_1b_1 \\ a_2b_2 \\ a_3b_3 \end{Bmatrix}$$

É importante observar o ponto (`.`) na frente do operador de multiplicação, que diferencia o produto convencional da operação elemento por elemento.

A divisão elemento por elemento é feita, de forma idêntica, com o operador `./`.

Igualmente, o operador `.^` efetua a potenciação elemento por elemento, entre dois vetores. Por exemplo

```
>> a=[2 2 2]; b=[1 2 3]; a.^b
ans =

     2     4     8
```

As operações de soma (`+`) e subtração (`-`) entre dois vetores são realizadas também elemento por elemento, como na álgebra vetorial convencional.

Todas as operações vetoriais elemento por elemento (`+ - ./ .* .^`) podem ser realizadas entre dois vetores, contanto que eles tenham a mesma dimensão.

Algumas funções normalmente usadas com escalares também podem ser aplicadas a vetores. Por exemplo, para criar uma tabela de valores de senos de ângulos entre 0 e 2π, com incrementos de 60 graus, primeiramente define-se o vetor dos ângulos

```
>> angulos=[0: pi/3: 2*pi]
angulos =

  0.00000  1.04720  2.09440  3.14159  4.18879  5.23599  6.28319
```

Então basta aplicar a função `sin` ao vetor de ângulos criado

```
>> y=sin(angulos)
y =

   0.00000   0.86603   0.86603   0.00000  -0.86603  -0.86603  -0.00000
```

Esse procedimento é usado para traçar gráficos, conforme mostrado em seção adiante.

3.3 Matrizes

Matrizes são definidas de forma semelhante à definição dos vetores: usam-se vírgulas ou espaços para separar os elementos de uma linha e pontos-e-vírgulas ou a tecla Enter para separar as linhas:

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

       1       2       3
       4       5       6
       7       8       9
```

Os elementos de uma matriz podem ser acessados fazendo referência à posição dos mesmos dentro da matriz, ou seja, pelos índices que definem a linha e a coluna do elemento. Assim, o elemento da segunda linha na terceira coluna pode ser acessado da seguinte maneira:

```
>> M(2,3)
ans = 6
```

Para a seleção de linhas ou colunas inteiras usa-se o operador dois pontos (`:`). O comando `M(:,2)` faz referência a todas as linhas da coluna 2; `M(3,:)` retorna a linha 3, com todos os seus elementos.

```
>> M(:,2)
ans =

       2
       5
       8

>> M(3,:)
ans =

       7       8       9
```

A partir da matriz **M** é possível definir outra:

```
>> N=M(1:2,2:3)
N =

        2        3
        5        6
```

O operador dois pontos foi usado duas vezes: uma para delimitar o intervalo de linhas desejado (da linha 1 a 2) e outra para especificar o intervalo de colunas (da coluna 2 a 3). Quando é necessário remover alguma linha ou coluna de uma matriz atribui-se a esta linha ou coluna a matriz nula (`[ ]`). Dessa forma a matriz remanescente será composta apenas das linhas (ou colunas) remanescentes. Por exemplo, para remover a terceira coluna da matriz **M**, usa-se a instrução:

```
>> M(:,3)=[ ]
M =

        1        2
        4        5
        7        8
```

É possível também montar matrizes maiores através de outras menores, como:

```
>> a=[1 1; 1 1]; b=[2 2; 2 2]; c=[a b; b a]
c =

        1        1        2        2
        1        1        2        2
        2        2        1        1
        2        2        1        1
```

Outras funções, listadas na Tabela 4, também podem ser usadas para a definição de matrizes.

Tabela 4: Comandos para manipulação de matrizes.

Comando	Descrição
`eye(N)`	Cria uma matriz identidade de dimensão N
`ones(M,N)`	Cria uma matriz de M linhas e N colunas, com todos os elementos iguais a 1
`zeros(M,N)`	Cria uma matriz nula, de M linhas e N colunas
`rows(A)`	Retorna o número de linhas da matriz A
`columns(A)`	Retorna o número de colunas da matriz A

3.4 Operações com matrizes

A adição, subtração e multiplicação de matrizes é feita com os operadores `+`, `-` e `*`, respectivamente e seguem as regras da álgebra matricial normal.

A multiplicação, a divisão e potenciação elemento por elemento, mostrada para vetores, também pode ser feita para matrizes, com os respectivos operadores `.*`, `./` e `.^`. Tomando duas matrizes **M** e **N**, de tamanho 2×2, o produto elemento por elemento entre as mesmas fica

$$\begin{bmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{bmatrix} .* \begin{bmatrix} n_{11} & n_{12} \\ n_{21} & n_{22} \end{bmatrix} = \begin{bmatrix} m_{11}n_{11} & m_{12}n_{12} \\ m_{21}n_{21} & m_{22}n_{22} \end{bmatrix}$$

Por exemplo

```
>> M=[1 2; 3 4]; N=[4 5; 6 7]; M.*N
ans =

    4   10
   18   28
```

4 Sistemas de equações lineares

4.1 Matriz dos coeficientes não-singular

Um sistema de equações lineares é dado da forma

$$\mathbf{Ax} = \mathbf{b},$$

onde **A** é a matriz dos coeficientes, **b** é o vetor de termos independentes e **x** é o vetor das incógnitas.

Resolver um sistema de equações é obter o valor de **x**. Se a matriz **A** é não-singular (admite inversa), a solução do sistema de equações pode ser dada por

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}.$$

Considere-se o sistema

$$\begin{aligned} x + y &= 3 \\ 2x - 3y &= 5 \end{aligned}$$

A solução deste sistema no Octave, usando a inversa de **A** é dada por

```
>> A=[1 1; 2 -3]; b=[3; 5]; x=inv(A)*b
x =

  2.80000
  0.20000
```

Observe-se que **b** é um vetor-coluna. O resultado pode ser conferindo fazendo o produto

```
>> A*x
ans =

  3.00000
  5.00000
```

que é equivalente ao vetor **b**, confirmando a solução encontrada.

Embora o procedimento de usar a inversa da matriz **A** possa dar respostas rápidas no Octave, do ponto de vista computacional ele exige um esforço maior. Uma forma mais eficiente é usar o *método da eliminação de Gauss* para resolver o sistema de equações. O Octave fornece como atalho para esse método o operador da barra invertida (\). É importante notar a diferença entre esse operador e o operador de divisão normal (/). Assim, o sistema anterior é resolvido usando o método de Gauss, da forma

```
>> A=[1 1; 2 -3]; b=[3; 5]; x=A\b
x =

  2.80000
  0.20000
```

4.2 Matriz dos coeficientes singular

Uma vez acionado, o procedimento dado pelo operador da barra invertida \ tenta achar a solução do sistema, quaisquer que sejam a matriz e o vetor passados, mesmo se a solução não for possível, ou indeterminada. Considere o exemplo

$$\begin{aligned} x_1 + x_2 + x_3 &= 2 \\ 2x_1 + 3x_3 &= 5 \\ 3x_1 + x_2 + 4x_3 &= 6 \end{aligned}$$

Resolvendo esse sistema usando o operador \

```
>> A=[1 1 1; 2 0 3; 3 1 4]; b=[2; 5; 6]; x=A\b
warning: matrix singular to machine precision, rcond = 1.15648e-17
warning: attempting to find minimum norm solution
x =

   0.69048
  -0.11905
   1.09524
```

Embora o vetor **x** tenha sido calculado, um aviso é emitido, dizendo que a matriz **A** é *singular*. Isso significa que a matriz é não-invertível e acontece porque pelo menos uma das linha de **A** é dada pela combinação linear das outras linhas. Nesse caso seu determinante é nulo. Isso pode ser confirmado por

```
>> det(A)
ans =  5.5511e-16
```

que dá um valor suficientemente próximo de zero.

Observando a matriz **A**, pode-se verificar que a terceira linha é dada pela soma das duas outras. O número de linhas linearmente independentes numa matriz é dada pelo seu *posto*. O posto de uma matriz pode ser calculado com a função `rank`.

```
>> rank(A)
ans = 2
```

Examinadno o sistema como um todo, pode-se notar que somando as duas primeiras equações do sistema, chega-se a $3x_1+x_2+4x_3 = 7$, o que é claramente *inconsistente* com a terceira equação. Em outras palavras não há solução para esse sistema de equações.

4.3 Mal condicionamento

Uma definição para um sistema de equações mal condicionado é se pequenas mudanças nos dados produz grandes mudanças nos resultados. Considere o sistema

$$\begin{aligned} x_1 + x_2 &= 2 \\ x_1 + 1.01x_2 &= 2.01 \end{aligned}$$

para o qual o Octave fornece a solução correta

```
>> A=[1 1; 1 1.01]; b=[2; 2.01]; x=A\b
x =

  1.00000
  1.00000
```

Agora considere-se uma pequena mudança de 0.5% em um dos elementos de **A**:

```
>> A(1,2)=1.005; x=A\b
x =

  -0.01000
   2.00000
```

Uma mudança de 0.5% em um elemento de **A** provocou um decréscimo em x_1 de 101% e um aumento de x_2 em 100%!

A sensibilidade de uma matriz é estimada pelo seu *número de condição*, cuja definição foge ao escopo desse tutorial. Contudo, quanto maior o número de condição, mais sensível a solução se torna. No Octave o número de condição é calculado com o uso da função `cond`:

```
>> A=[1 1; 1 1.01]; cond(A)
ans = 402.01
```

Para a matriz singular do item anterior, o cálculo do número de condição leva a

```
>> A=[1 1 1; 2 0 3; 3 1 4]; cond(A)
ans =  1.7309e+16
```

que, como é de se esperar, tem um valor bem alto.

5 Gráficos

5.1 A janela gráfica

Além de um ambiente de trabalho baseado em linha de comando, quando usado em um ambiente XWindow, o Octave cria automaticamente uma janela separada para a apresentação de gráficos (Figura 1). Mesmo em ambiente texto é possível direcionar a saída gráfica para um arquivo, de forma a visualizá-lo posteriormente em outro aplicativo. A interface gráfica do Octave é baseada no GNUPLOT (Crawford, 1998), um utilitário de distribuição livre para traçado de funções .

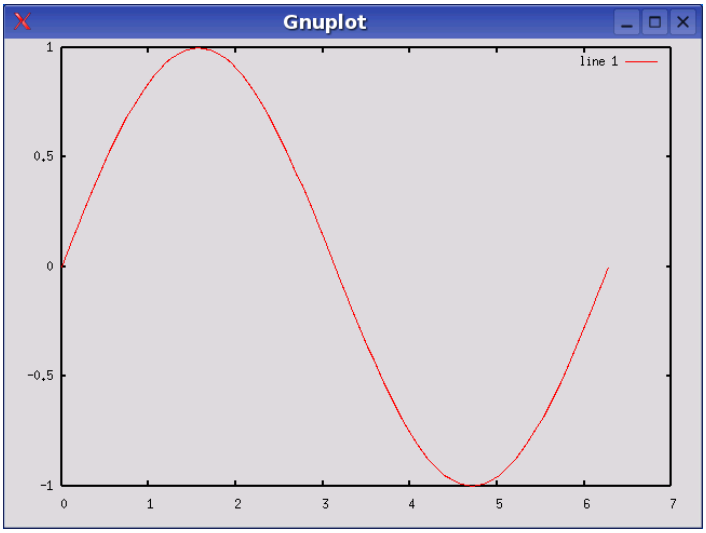

Figura 1: Saída gráfica do Octave

5.2 Gráficos bidimensionais

O comando básico para o traçado de gráficos bidimensionais é o `plot(x,y)`. Os parâmetros `x` e `y` são as coordenadas a serem traçadas. Se `x` e `y` forem um par de escalares, somente um ponto é traçado. Usando vetores como parâmetros, o programa irá traçar todos os pontos correspondentes a esses valores uní-los por linhas retas.

O primeiro passo para o traçado de gráficos 2D é formar uma tabela de coordenadas (x, y) da função ser plotada. O procedimento para criar essa tabela usando vetores foi mostrado em seção anterior. Tome-se, como exemplo, a função $cos(x)$, a ser traçada no intervalo entre 0 e 2π. Para construir um vetor **x**, com 100 valores dentro desse intervalo, pode-se usar o comando

```
>> x=[0: 2*pi/100: 2*pi];
```

Outra opção é o uso do comando

```
>> x=linspace(0, 2*pi);
```

Para completar a tabela de coordenadas, determina-se o vetor **y** correspondente aos valores em **x**. Isso é feito simplesmente invocando a função com o simples comando

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

Uma vez construída a tabela com as coordenadas (x, y), pode-se usar a função `plot`:

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

A Figura 2 mostra a curva obtida

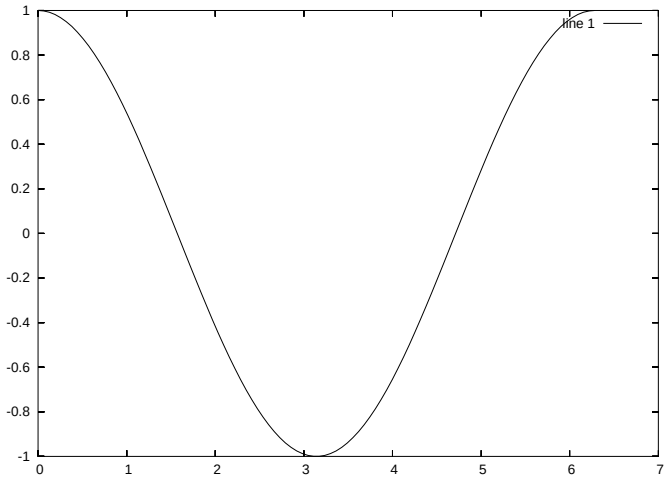

Figura 2: Gráfico de $y = cos(x)$

Linhas de grade podem ser adicionadas ao gráfico com o comando

```
>> grid on
```

O resultado é mostrado na Figura 3. O comando `grid off` retorna ao modo sem linhas de grade.

A aparência do gráfico pode ser modificada com os parâmetros adionais, mostrados na Tabela 5, colocados entre aspas. Por exemplo o comando

```
>> plot(x,y,"-kx")
```

traça o gráfico com linha cheia, na cor preta, usando um "x" como marcador das coordenadas.

Os comandos `title`, `xlabel` e `ylabel` permitem escrever um título para o gráfico e um rótulo em cada um dos eixos. Deve-se passar como parâmetro para esses comandos uma seqüência de caracteres entre aspas.

Para salvar o gráfico produzido em um arquivo de imagem do tipo PostScript Encapsulado (EPS), a seqüência de comandos é a seguinte

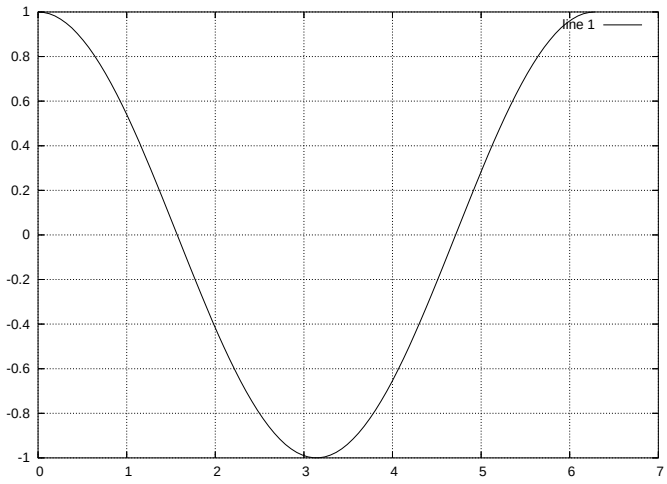

Figura 3: Gráfico de $y = cos(x)$, com linhas de grade

Tabela 5: Parâmetros para traçado de gráficos.

Símbolo	Cor	Símbolo	Marcador	Símbolo	Tipo de linha
b	azul	+	sinal positivo	-	linha cheia
w	branco	*	asterisco	.	pontilhado
c	ciano	o	diamante		
m	magenta	x	letra x		
k	preto				
g	verde				
r	vermelho				

```
>> gset term postscript eps
>> gset output "nome_grafico.eps"
>> replot
```

Para fazer voltar a saída gráfica para a tela, pode-se sair e entrar novamente no Octave ou executar o comando:

```
>> gset term x11
```

Em ambiente Windows, o parâmetro "`x11`" no comando acima deve ser substituído por "`windows`". O comando `close` fecha a tela gráfica.

Para traçar gráficos de funções paramétricas, deve-se usar o comando `gset parametric`. Por exemplo, para traçar o gráfico da equação $\begin{cases} x = \text{sen}(3t) \\ y = \cos(5t) \end{cases}$, com o argumento t variando no intervalo $[-\pi, \pi]$, pode-se usar a seguinte seqüência de comandos:

```
>> gset parametric
>> t=-pi: pi/100: pi;
>> plot(sin(3*t),cos(5*t))
>> gset noparametric
```

Note que o comando `gset` é usado novamente, ao final, para desativar o traçado de funções paramétricas.

5.3 Gráficos tridimensionais

O comando `mesh(x,y,z)` permite traçar a malha para gráficos tridimensionais, da forma

$$z = f(x, y)$$

Nesse caso as variáveis independentes x e y são vetores de dimensões n_x e n_y, respectivamente, e a variável dependente z é uma matriz de dimensão $n_x \times n_y$. Assim, conforme essa definição, o elemento $z(i, j)$ é o valor da superfície no ponto $(x(i), y(j))$.

Considere-se por exemplo a função

$$z = cos(x)sin(y)$$

no intervalo $[0, 2\pi]$ dividido em 50 incrementos. Os comandos

```
>> x=[0: 2*pi/50: 2*pi]';
>> y=x;
>> z=cos(x)*sin(y');
>> mesh(x,y,z)
```

produzem a curva mostrada na Figura 4.

Para mudar o ângulo de visão, pode-se clicar com o botão direito do mouse sobre a figura, e arrastá-la para uma nova posição.

Enquanto o comando `mesh(x,y,z)` representa o gráfico por meio de uma malha, o comando `surf(x,y,z)` representa a função tridimensional como uma

superfície, adicionando à malha efeitos de cores e profundidade. Uma vez que essa função é acionada, as chamadas subsequentes à função `mesh` irão também mostrar uma superfície com os mesmos efeitos de profundidade, a não ser que o comando `clf` seja usado antes para limpar a janela gráfica, ou então o comando `close` seja usado para fechá-la.

A seqüência de comandos usadas para o gráfico da Figura 4 é válida para funções do tipo $z = f(x)g(y)$. Para outros tipos de funções um outro procedimento é necessário. Considere-se, por exemplo, a função $z = (x-3)^2 - (y-2)^2$. Uma seqüência de comandos para traçar o gráficos dessa equação é a seguinte:

```
>> x = 2: 0.2: 4;
>> y = 1: 0.2: 3;
>> [xx,yy] = meshgrid(x,y);
>> z = (xx-3).^2 - (yy-2).^2;
>> surf(x,y,z)
```

Esse comandos produzem a forma de sela, mostrada na Figura 5.

O comando `meshgrid`, usado no exemplo acima, recebe dois vetores de coordenadas x e y e retorna duas matrizes correspondentes às coordenadas da malha. As linhas de xx são cópias de x e as colunas de yy são cópias de y. Note que na potenciação é usada a operação elemento por elemento, com "`.^`" no lugar de "`^`".

6 Programação

As seqüências de comandos do Octave podem ser armazenadas em arquivos e executadas no espaço de trabalho. Esses arquivos são do tipo texto simples, sem formatação, e devem ter a extensão ".m". Os arquivos devem ser salvos no diretório de trabalho corrente. No ambiente de trabalho do Octave o comando

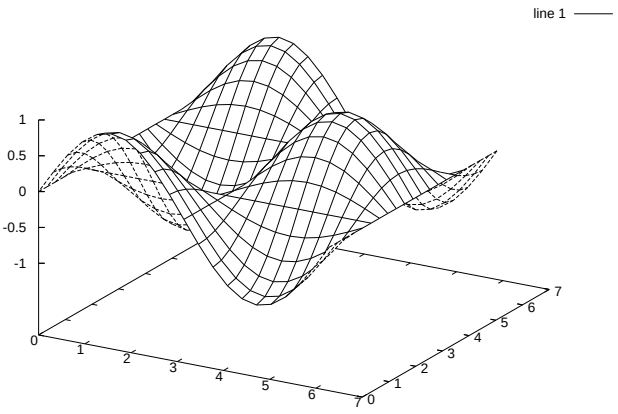

Figura 4: Gráfico de $z = sin(x)cos(x)$

`pwd` mostra qual é o diretório corrente. Os comandos `ls` ou `dir` fornecem uma lista dos arquivos do diretório corrente. O comando `cd` permite mudar o diretório corrente, enquanto o uso do comando "`cd ..`" permite subir um nível na árvore de subdiretórios.

O arquivo de comandos é executado uma vez que o seu nome é invocado na linha de comandos do Octave. Alguns recursos de programação são descritos a seguir.

6.1 Exibição de mensagens na tela

A função `disp` permite mostrar uma mensagem na tela ou mostrar o conteúdo de uma variável, finalizando com uma nova entrada de linha. Por exemplo,

```
>> disp("A base do logaritmo neperiano vale:"), disp(e)
A base do logaritmo neperiano vale:
2.7183
```

Esse comando pode ser útil em alguns programas, onde for necessário enviar uma mensagem ao usuário, ou mostrar resultados parciais de cálculo.

Uma exibição de mensagens e variáveis com formatação mais sofisticada pode ser conseguida com a função `fprintf`, também usada em linguagem C. A função tem a seguinte sintaxe:

`printf("`*formatação*`",`*argumentos*`);`

A expressão de controle da formatação pode conter caracteres que serão exibidos na tela e os códigos que indicam o formato em que os argumentos devem ser impressos, mostrados na Tabela 6. Cada argumento deve ser separado por vírgula.

É conveniente colocar um caractere de ponto e vírgula (`;`) ao do final do comando. Caso contrário, a função retornará, além da mensagem produzida, a

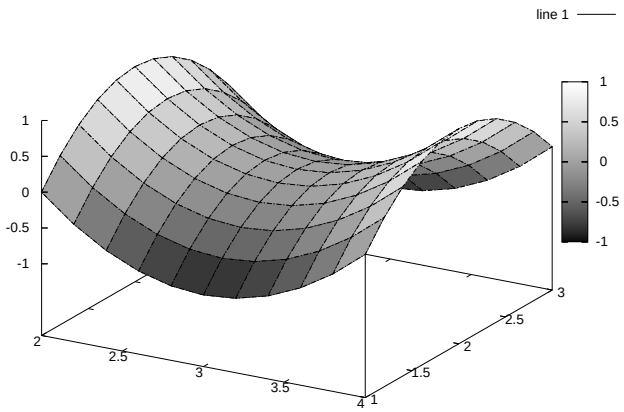

Figura 5: Gráfico de $z = (x-3)^2 - (y-2)^2$

Tabela 6: Códigos de formatação da função printf.

`%c`	caractere simples	`\n`	nova linha
`%d`	decimal	`\t`	tabulação
`%e`	notação científica	`\b`	retrocesso
`%f`	ponto flutuante	`\"`	aspas
`%o`	octal	`\\`	barra invertida
`%s`	cadeia de caracteres	`\f`	salta formulário
`%u`	decimal sem sinal	`\0`	nulo
`%x`	hexadecimal		

quantidade de caracteres gerados. É conveniente, também, indicar uma entrada de linha, ao final da formatação (`\n`), de forma a evitar que o sinal de prontidão do Octave fique posicionado a seguir na mesma linha da mensagem produzida. Um exemplo de uso do comando é dado a seguir:

```
>> it=1; A=1.73; X=sqrt(3);
>> printf("Iteracao: %d Raiz aprox.: %f Raiz exata: %f\n",it,A,X);
Iteracao: 1 Raiz aprox.: 1.730000 Raiz exata: 1.732051
```

6.2 Entrada e saída de dados

Eventualmente pode ser necessário que o programa solicite ao usuário que digite algum dado no teclado. Para isso é usado a função `input`. Por exemplo,

```
>> n = input("Digite um numero: ")
```

mostra na tela a mensagem

```
Digite um numero:
```

e aguarda até que o usuário digite o dado solicitado. A seqüência de caracteres entrada é calculada como uma expressão e o seu valor é atribuído à variável `n`. A seqüência de caracteres pode ser um valor numérico, o nome de uma variável ou qualquer outra expressão válida. Caso se queira obter uma seqüência de caracteres literais, sem que o seu valor seja calculado, deve-se usar como segundo argumento do comando, a expressão '`"s"`'.

Em algumas situações pode ser conveniente salvar os resultados dos cálculos feitos em um arquivo ou mesmo ler os valores a serem atribuídos às variáveis a partir de arquivos em disco. Para isso são usados os comandos `save` e `load`.

O comando `save` permite guardar os valores das variáveis em arquivo e tem a forma

`save` *opções nomearquivo listadevariáveis*

Se a lista de variáveis não for especificada todas as variáveis correntes na área de trabalho são salvas. Para que as variáveis sejam salvas no formato texto (não binário), a opção `-ascii` deve ser usada. Por exemplo, o comando

```
>> A = [1 2; 3 4]; b=33;
>> save -ascii teste.dat A b
```

cria as variáveis `A` e `b` e salva-as no arquivo de nome "teste.dat", no formato texto. Nesse caso, o arquivo teste.dat fica:

```
# Created by Octave 2.1.50, Mon Mar 28 16:18:57 2005     <vsj@SCARPA>
# name: A
# type: matrix
# rows: 2
# columns: 2
 1 2
 3 4
# name: b
# type: scalar
33
```

O comando `load` permite ler variáveis guardadas em um arquivo. As variáveis devem ser escritas no arquivo, no formato usado pelo comando `save`. O comando tem a forma

`load` *opções nomearquivo listadevariáveis*

A lista de variáveis dever vir com os nomes das mesmas, separados por espaço em branco. Se for especificada uma lista de variáveis, o comando irá ler somente as variáveis cujos nomes conferem com os relacionados. O Octave não sobrescreve o valor de variáveis já existentes, a não ser que a opção `-force` seja usada. Por exemplo, o comando

```
>> load -force teste.dat A b
```

irá buscar no arquivo de nome "teste.dat" as variáveis `A` e `b`. A opção `-force` indica que se essas variáveis já existirem na área de trabalho, os novos valores lidos devem ser atribuídos às mesmas.

6.3 Definição de funções

Uma função no Octave tem a forma geral:

`function [`*lista-saída*`] = nome(`*lista-entrada*`)`
  comandos da função
`endfunction`

onde *lista-saída* é uma lista de parâmetros de saída da função, separados por vírgula; *lista-entrada* é uma lista de parâmetros de entrada, separados por vírgula; `nome` é o nome dado à função.

Uma função pode ser criada digitando-a no ambiente de trabalho, ou criando um arquivo com a função e salvando-o no diretório de trabalho. O arquivo deve ter o mesmo nome dado à função e a extensão "`.m`". Com exceção das variáveis definidas como parâmetros de saída, as variáveis definidas na função, chamadas

de variáveis locais, não permanecem no ambiente de trabalho após a execução da função.

Como exemplo, considere-se a criação de uma função, chamada de `somaprod`, que recebe dois valores retorna a soma e o produto entre esses dois valores. A função fica da seguinte forma:

```
function [soma, produto] = somaprod(a,b)
% Funcao de exemplo
% Recebe dois parametros e calcula
% a soma e o produto entre os mesmos
  soma = a+b;
  produto = a*b;
endfunction
```

Como se pode notar, comentários podem ser colocados na função com o uso do caractere `%`. Após salvar a função no arquivo `somaprod.m`, a mesma pode ser usada como se fosse uma função pré-existente no Octave:

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

Para que a função possa ser usada também para realizar as operações de soma e produto em uma lista de valores ao mesmo tempo, o operador de produto (`*`) deve ser substituído pelo operador de multiplicação elemento por elemento (`.*`). Assim, a evocação da função com a passagem de vetores, que nesse caso funcionam como listas de valores, retorna também listas de resultados, para cada variável de saída:

```
>> [s,p]=somaprod([1 2],[3 4])
s =

        4        6

p =

        3        8
```

6.4 Estruturas condicionais

A estrutura condicional `if` tem três formas básicas. A forma mais simples é a seguinte:

```
if (condição)
  seqüência de comandos
endif
```

A *seqüência de comandos* é executada somente se expressão contida na *condição* resultar em verdadeiro. A expressão condicional é construída com o auxílio dos operadores mostrados na Tabela 7.

A segunda forma da estrutura condicional é como segue:

Tabela 7: Operadores para construção de expressões lógicas.

Operadores relacionais		Operadores lógicos	
`==`	igual	`&`	E
`~=`	diferente	`|`	OU
`<`	menor	`~`	NÃO
`<=`	menor ou igual		
`>`	maior		
`<=`	maior ou igual		

`if (`*condição*`)`
  seqüência de comandos A
`else`
  seqüência de comandos B
`endif`

Se a *condição* for verdadeira a *seqüência de comandos A* é executada; caso contrário, a *seqüência de comandos B* é executada.

A terceira forma, mais genérica, permite a combinação de múltiplas decisões numa estrutura única:

`if (`*condição*`)`
  seqüência de comandos A
`elseif (`*condição*`)`
  seqüência de comandos B
  ⋮
`else`
  seqüência de comandos N
`endif`

Não há limites para o número de cláusulas `elseif` na estrutura. Cada condição é testada e se resultar em verdadeiro, a seqüência de comandos correspondente é executada. Somente uma cláusula `else` pode aparecer e deve estar na última parte da estrutura.

Uma outra estrutura condicional útil é a `switch`, que tem a forma geral:

`switch` *expressão*
  `case` *rótulo*
    seqüência de comandos A
  `case` *rótulo*
    seqüência de comandos B
    ⋮
  `otherwise`
    seqüência de comandos N
`endswitch`

Pode-se usar um número ilimitado de cláusulas `case` na estrutura e a cláusula `otherwise` deve aparecer na sua última parte. A estrutura `switch` foi introduzida na versão 2.0.5 do Octave.

6.5 Estruturas de repetição

A estrutura `for` é de uso conveniente quando se deseja implementar um número de repetições previamente sabido. A estrutura tem a forma:

`for` *variável* `=` *vetor-linha*
  seqüência de comandos
`endfor`

Nesse tipo de estrutura, a *variável* assume os valores contidos em cada posição do *vetor-linha*, a cada iteração. A última iteração é feita quando a *variável* assumir o valor da última posição do *vetor-linha*.

A estrutura `while` estrutura tem a forma:

`while (`*condição*`)`
  seqüência de comandos
`endwhile`

Nessa estrutura a seqüência de comandos é executada enquanto a *condição* for atendida.

A estrutura `do-until` é semelhante à estrutura `while`, exceto que nela a seqüência de comandos é repetida até que determinada condição seja atendida. Tem a seguinte forma:

`do`
  seqüência de comandos
`until (`*condição*`)`

7 Tópicos adicionais

7.1 Raízes de equações

Uma função polynomial $f(x)$, de grau n, é uma função da forma

$$f(x) = a_0 + a_1 x + \cdots + a_{n-1}x^{n-1} + a_n x^n.$$

onde $a_0, \ldots, a_n, n > 0$ são constantes numéricas.

Um polinômio é representado no Octave, por meio de um vetor com seus coeficientes, arranjados de forma descendente. Por exemplo, para representar o polinômio

$$p(x) = x^4 - 3x^3 - 15x^2 + 19x + 30$$

pode-se digitar

```
>> p = [ 1 -3 -15 19 30]
p =

    1   -3  -15   19   30
```

Uma raiz r de um polinômio p é um número tal que $p(r) = 0$. Para calcular as raízes do polinômio, após definí-lo, usa-se o comando `roots`:

```
>> roots(p)
ans =

   5.0000
  -3.0000
   2.0000
  -1.0000
```

O teorema fundamental da álgebra estabelece que um polinômio de grau n tem n raízes, entre reais e complexas. No caso do polinômio ter raízes complexas, as mesmas são dadas como no exemplo:

```
>> q=[7 4 1 2]
q =

  7  4  1  2

>> roots(q)
ans =

  -0.82117 + 0.00000i
   0.12487 + 0.57649i
   0.12487 - 0.57649i
```

Para achar raízes de funções não-polinomiais, a função `fsolve` pode ser usada. Essa função recebe como parâmetros o nome da função que se deseja calcular a raiz e a estimativa inicial para a raiz. Por exemplo, para calcular a raiz da função $f(x) = x^2 + \ln x$, com a estimativa inicial para a raiz $x = 0,5$ inicialmente deve-se definir tal função, da seguinte forma:

```
> function y = minhafuncao(x)
> y = x^2 + log(x)
> endfunction
```

Em seguida, invoca-se `fsolve`, da seguinte forma:

```
> [x,informacao]=fsolve('minhafuncao',0.50)
y = -0.44315
y = -0.44315
y = -0.014768
y =  -3.1395e-04
y =  -1.0317e-07
y =  -6.9605e-13
y =  1.6653e-16
x = 0.65292
informacao = 1
```

O resultado `informacao = 1` indica que a solução convergiu. No caso, o valor `x = 0.65292` é a raiz calculada

7.2 Integração numérica

Para a integração de funções de uma variável, a função `quad` é usada. Sua sintaxe é a seguinte:

`[`*v*, *ier*, *nfun*, *err*`] = quad (`*f*, *a*, *b*, *tol*, *sing*`)`

O primeiro argumento é o nome da função a ser integrada. Essa deve ter a forma $y = f(x)$, onde x e y são escalares. O segundo e o terceiro argumentos são os limites de integração. O valor infinito (`Inf`) pode ser atribuído a tais limites.

O argumento opcional *tol* é um vetor que especifica a precisão desejada para o resultado. O primeiro elemento do vetor é a tolerência absoluta desejada e o segundo elemento é a tolerância relativa desejada. Para escolher somente um teste relativo indique o valor 0 para a tolerência absoluta. Para escolher somente um teste absoluto indique o valor 0 para a tolerência relativa.

O argumento opcional *sing* é um vetor que contém um conjunto de valores para os quais se sabe que a integração é singular.

O resultado da integração é devolvido em *v* e um código de erro é retornado em *ier*, com 0 indicando uma integração bem sucedida. O valor de *nfun* indica a quantidade de vezes que a função foi calculada e *err* contém uma estimativa de erro na solução.

Por exemplo, para calcular $\int_{3.0}^{3.6} \frac{dx}{x}$, digita-se:

```
>> function y = minhafuncao2(x)
> y = 1/x
> endfunction
>> [v, ier, nfunc, err] = quad('minhafuncao2', 3.0, 3.6)
y = 0.30303
y = 0.33247
y = 0.27838
y = 0.32890
y = 0.28094
y = 0.32298
y = 0.28540
y = 0.31546
y = 0.29154
y = 0.30719
y = 0.29898
y = 0.33319
y = 0.27788
y = 0.33102
y = 0.27940
y = 0.32618
y = 0.28295
y = 0.31937
y = 0.28828
y = 0.31136
y = 0.29513
v = 0.18232
```

```
ier = 0
nfunc = 21
err =  2.0242e-15
```

Referências

Crawford, D. *Gnuplot online documentation*. http://www.gnuplot.info/docs/gnuplot.html. 1998.

Eaton, J. W. *GNU Octave - A high-level interactive language for numerical computation*. http://www.octave.org. 1997.

Free Software Foundation. *GNU General Public License*. Version 2. http://www.fsf.org/licensing/licenses/index_html. Jun 1991.

Smith, P. *Tutorial Guide to Matlab*. Department of Engineering. University of Cambridge. Sep 2001.