

Cálculo Numérico

Material básico e de apoio para a disciplina Cálculo Numérico

Salvador 2018-1

DR. MARCOS FIGUEREDO

Dr. Marcos Figueredo
Escola de Arquitetura,
Engenharia e Tecnologia da
Informação
Engenharia
`marcos.figueredo@unifacs.br`

1

Introdução

O que são métodos numéricos e por que devemos estudá-los? Métodos numéricos são técnicas pelas quais os problemas matemáticos são formulados de modo que possam ser resolvidos com operações lógicas e aritméticas. Como os computadores digitais se sobressaem na execução de tais operações, os métodos numéricos são, as vezes, referidos de forma mais ampla como matemática computacional. Além das contribuições para a educação geral, existem diversas razões adicionais pelas quais os métodos numéricos devem ser estudados:

1. Os métodos numéricos são ferramentas extremamente poderosas na resolução de problemas.
2. Durante a carreira, o profissional de engenharia frequentemente terá oportunidade de usar softwares disponíveis no mercado. O uso inteligente desses programas depende do conhecimento da teoria básica fundamental dos métodos;
3. A familiarização com os métodos permite que o engenheiro produza seu próprio conteúdo, permitindo superar dificuldades encontradas em softwares prontos.
4. Os métodos numéricos representam um meio eficiente para a fixação das ideias de programação que representam um diferencial no mundo do trabalho.
5. Auxiliam o entendimento e a aplicação dos diversos conceitos de cálculo aprendidos ao longo do curso.

2

Detalhes da disciplina

Nossa disciplina discute as associações entre os métodos numéricos e problemas de engenharia, utilizando linguagem computacional ou software numérico. São apresentadas situações-problemas que requerem a adoção de soluções empregando-se estudos e análises de métodos numéricos e computacionais. São enfatizados os aspectos de interpretação dos resultados numéricos obtidos.

2.1 Carga horária

Nome da Disciplina	Carga horária	Dia	horário
Cálculo Numérico	33 horas	dia	17:30 as 18:45

2.2 Ementa

Discute as associações entre os métodos numéricos e problemas de engenharia, utilizando linguagem computacional ou software numérico. São apresentadas situações-problemas que requerem a adoção de soluções empregando-se estudos e análises de métodos numéricos e computacionais. São enfatizados os aspectos de interpretação dos resultados numéricos obtidos.

2.3 Conteúdo Programático

1. Erros Numéricos e Aritmética de Ponto Flutuante
2. Zero de Funções
3. Interpolação
4. Integração Numérica

2.4 Metodologia

Aula expositivo-dialogada, juntamente com resolução de problemas no intuito de fixação do conteúdo. Como a disciplina esta intimamente relacionada com o desenvolvimento da lógica, muitas vezes o desenvolvimento dos temas será instigado mediante ao estudo de uma situação problema. No intuito de dar dinamismo às aulas, em muitos momentos serão oferecidos aos alunos resumos teóricos dos conteúdos.

2.5 Avaliação

A avaliação da disciplina será dividida em duas etapas sendo duas provas escritas.

Data	Avaliação	Nota
25/04/2018	1ª Avaliação - AV1	10 pontos
06/06/2018	2ª Avaliação- AV2	10 pontos
13/06/2018	2ª Chamada - AV1 e AV2	10 pontos
27/06/2018	3ª Avaliação	10 pontos

2.6 Referências

- 1) DORNELLES FILHO, Adalberto Ayjara. Fundamentos de Cálculo Numérico. 1. ed. Porto Alegre: Editora Bookman, 2016 (Disponível na Biblioteca Virtual 3.0)
- 2) PIRES, Augusto de Abreu. Cálculo Numérico: Prática com Algoritmos e Planilhas. 1. ed. São Paulo: Editora Atlas, 2014 (Disponível na Biblioteca Virtual 3.0)
- 3) VARGAS, José Viriato Coelho e ARAKI, Luciano Kiyoshi. Cálculo Numérico Aplicado. 1. ed. Barueri: Editora Manole, 2017 (Disponível na Biblioteca Virtual 3.0)

Material Complementar

- 1) ARENALES, Selma e DAREZZO, Artur. Cálculo Numérico: aprendizagem com Apoio de Software. 2. ed. São Paulo: Editora Cengage Learning, 2015 (Disponível na Biblioteca Virtual 3.0)
- 2) ÁVILA, Geraldo e ARAÚJO, Luís Claudio Lopes de. Cálculo - Ilustrado, Prático e Descomplicado. 1. ed. Rio de Janeiro: Editora LTC, 2012 (Disponível na Biblioteca Virtual 3.0)
- 3) CHAPRA, Steven C. e CANALE, Raymond P. Métodos Numéricos para Engenharia. 7. ed. Porto Alegre: Editora Bookman, 2016 (Disponível na Biblioteca Virtual 3.0)

- 4) CHAPRA, Steven C. Métodos Numéricos com MATLAB® para Engenharia. 3. ed. Porto Alegre: Editora Bookman, 2016 (Disponível na Biblioteca Virtual 3.0)
- 5) SILVA, Paulo Sérgio Dias da. Cálculo Diferencial e Integral. 1. ed. Rio de Janeiro: Editora LTC, 2017 (Disponível na Biblioteca Virtual 3.0)

3

Conceitos iniciais

O Cálculo Numérico corresponde a um conjunto de ferramentas ou métodos usados para se obter a solução de problemas matemáticos de forma aproximada. Esses métodos se aplicam principalmente a problemas que não apresentam uma solução exata, portanto precisam ser resolvidos numericamente.

Assim podemos afirmar que estudamos um conjunto de métodos (metodologia) para resolver problemas matemáticos por meio de uma máquina calculadora ou um computador, sendo de grande importância pois, embora os métodos analíticos usualmente nos forneçam a resposta em termos de funções matemáticas, existem problemas que não possuem solução analítica.

Exemplo

Calcule a integral de

$$\int e^{x^2} dx$$

Mas, mesmo nestes casos podemos obter uma solução numérica para o problema. Uma solução via Cálculo Numérico é um conjunto de dados numéricos que fornecem uma aproximação para a solução exata do problema, aproximação esta que pode ser obtida em grau crescente de exatidão.

3.1 Modelagem Matemática e Soluções de problemas

Conhecimento e compreensão são pré-requisitos para a implementação efetiva de qualquer ferramenta. Não importa quão impressionante seja a sua caixa de ferramentas, você terá dificuldade em reparar um carro se você não entender como funciona.

Isto é particularmente verdadeiro ao usar computadores para resolver problemas de engenharia. Embora tenham uma grande utilidade potencial, os computadores são praticamente inúteis sem uma compreensão fundamental de como funcionam os sistemas de engenharia.

Esse entendimento é inicialmente adquirido por meios empíricos, isto é, por observação e experiência. No entanto, embora tal informação derivada empiricamente seja essencial, é apenas metade da história. Ao longo de anos de observação e experiência, engenheiros e cientistas notaram que certos aspectos de seus estudos empíricos ocorrem repetidamente.

Esse comportamento geral pode então ser expresso como leis fundamentais que essencialmente incorporam a sabedoria cumulativa da experiência passada. Assim, a maioria das resoluções de problemas de engenharia emprega a abordagem em duas vertentes do empirismo e análise teórica, Figura 1. Um *Modelo matemático simples* pode ser definido, de forma geral, como uma formulação ou equação que expressa as características essenciais de um sistema ou processo físico em termos matemáticos. Em um sentido muito geral, ele pode ser representado com uma relação funcional da forma:

$$\text{Variável dependente} = f(\text{variáveis independentes, parâmetros, funções forçantes}) \quad (1)$$

em que:

1. Variável dependente= é uma característica que normalmente reflete o comportamento ou estado do sistema;

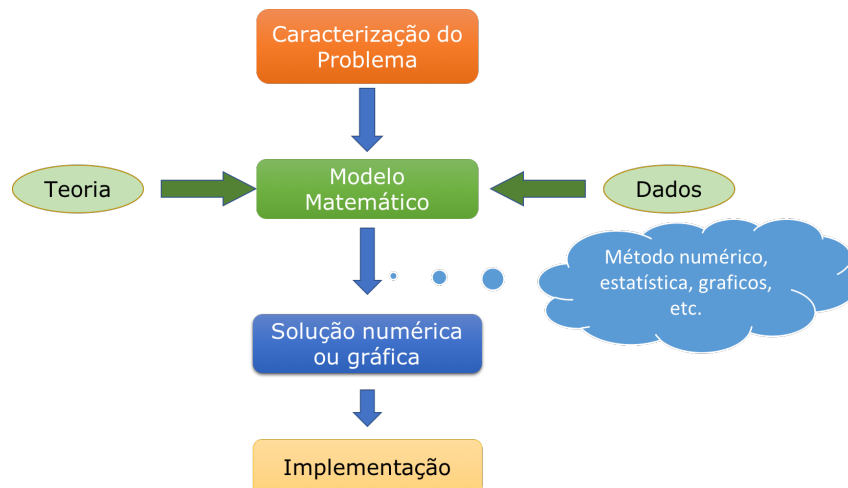


Figura 1: sintetização da resolução de problemas

2. variáveis independentes= normalmente são dimensões, como tempo e espaço, ao longo das quais o comportamento do sistema é determinado;
3. parâmetros= refletem propriedades ou composição do sistema;
4. funções forçantes = são as influências externas agindo sobre o sistema;

A expressão real da equação 1 pode variar de uma simples relação algébrica a um conjunto grande e complicado de equações diferenciais.

Duas ideias são frequentes em cálculo numérico, a de **iteração ou aproximação sucessiva** e a de **aproximação local**.

Iteração: Em um sentido amplo, iteração significa a repetição sucessiva de um processo. Um método iterativo se caracteriza por envolver os seguintes elementos:

- a) **Aproximação inicial:** consiste em uma primeira aproximação para a solução do problema numérico.
- b) **Teste de parada:** é o instrumento por meio do qual o procedimento iterativo é finalizado

Aproximação local: Aqui a ideia é aproximar uma função por outra que seja de manuseio mais simples. Por exemplo, aproximar uma função não linear por uma função linear em um determinado intervalo do domínio das funções.

4

Princípios básicos do MATLAB

MATLAB (R), acrônimo de MATrix LABoratory, é um sistema interativo e linguagem de programação para computação numérica e visualização para as áreas científicas e técnicas. Seu elemento básico de dados é uma matriz.

O MATLAB permite a solução de muitos problemas numéricos em uma fração do tempo que seria necessário para escrever um programa em uma linguagem como java, C ou Pascal. Além do mais, em MATLAB as soluções dos problemas são expressas de um modo bem próximo do que elas são escritas matematicamente.

O objetivo principal desta seção é fornecer uma introdução e uma visão geral do programa. Além de analisar seu ambiente e suas regras de prioridade numérica.

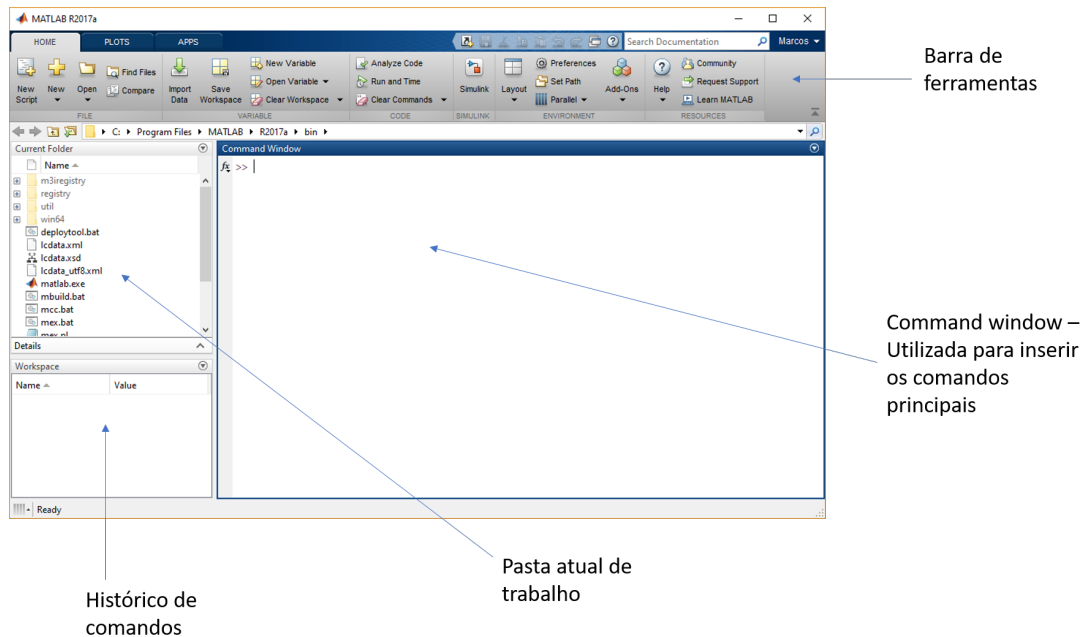


Figura 2: Janela Principal do MATLAB

A interação entre o MATLAB e o usuário é feita por intermédio de uma janela, na qual um comando é fornecido e o resultado exibido. No caso de dúvida sobre a sintaxe ou mesmo a existência de um comando, o MATLAB possui meios de auxiliar o usuário durante a sessão.

4.1 Janela de Comando

O MATLAB é um software que fornece ao usuário um ambiente adequado à realização de diversos tipo de cálculos; além disso, contém ferramentas bastante úteis para implementação de métodos numéricos.

Pode-se operar o MATLAB inserindo um comando de cada vez no *prompt* de comando (command window) Figura 2. Nesta seção, usaremos esse modo interativo ou modo calculadora para introduzir operações comuns, como realização de cálculos simples e criação de gráficos. Em seguida mostraremos como tais comandos podem ser usados para desenvolver programas no MATLAB.

Após inicializar o MATLAB a janela *command window* abrirá com o prompt de comando a seguir

Por simplicidade utilizaremos neste material uma janela recortada:

Listing 1: janela comando Matlab recortada

```
>>
```

O modo calculadora do MATLAB opera de um modo sequencial à medida que os comandos são digitados linha por linha. Para cada comando aparece um resultado, portanto, pode-se pensar nesse modo de operação do MATLAB com uma calculadora sofisticada. figura 4

Um comando é finalizado acionando-se a tecla Enter ou Return.

Todo texto após o sinal de % é considerado comentário sendo usado para a documentação de um programa. As teclas ↑ e ↓ servem para listar os comandos previamente usados

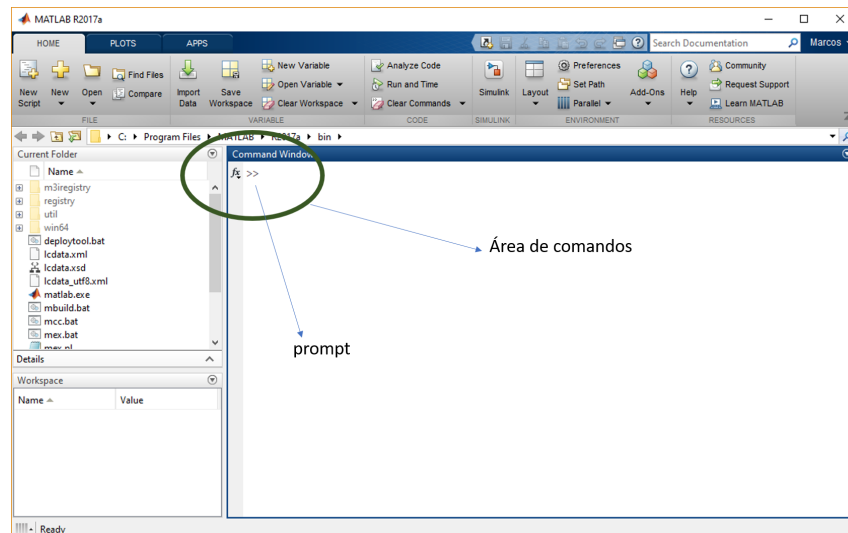


Figura 3: Local para digitação dos comandos

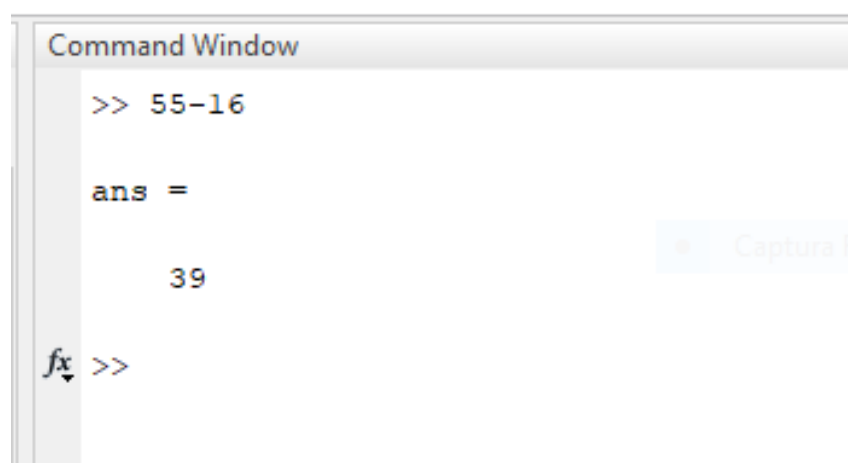


Figura 4: calculadora simples matlab

e as teclas \rightarrow e \leftarrow movem o cursor na linha de comando possibilitando a sua modificação.

Observe que o MATLAB atribui automaticamente a resposta a uma variável, *ans* no cálculo seguinte:

Listing 2: janela comando Matlab recortada

```
>> ans+11
```

como resultado:

Listing 3: janela comando Matlab recortada

```
>> ans =  
      39
```

O MATLAB sempre atribuirá os resultados à variável padrão *ans*, se os cálculos dos valores forem explicitamente atribuídos a uma variável definida pelo usuário.

A execução de um comando pode ser interrompida a qualquer instante bastando para isto pressionar as teclas *Control + C*, simultaneamente. O comando *clc* é usado para limpar a janela de comandos e *home* posiciona o cursor no canto superior esquerdo da janela de comando. O término de execução do MATLAB é feito pelos comandos *quit* ou *exit*.

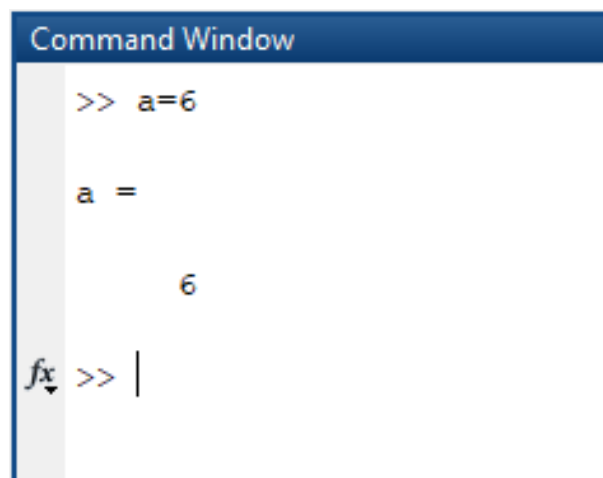
A abrangência e potencialidade do MATLAB está muito além do que será mostrado neste texto, por isso é aconselhável executar o comando *demo* para visualizar uma demonstração e se ter uma ideia dessa potencialidade.

4.2 Atribuição

A ação de atribuir valores a uma variável cham-se atribuição, o que resulta no armazenamento dos valores na posição de memória correspondente à variável.

4.2.1 Escalares

A atribuição de valores escalares é similar a outras linguagens de programação. Digite:



e observe que, após o comando de atribuição, o nome e o valor da variável são impressos na tela para confirmar o que foi feito.

essa é uma característica do MATLAB, que pode ser suprimida finalizando a linha de comando com um ponto e virgula(;). Digite

```
Command Window
>> a=7;
fx >> |
```

É possível digitar vários comandos na mesma linha separando-os com virgulas ou pontos e virgulas. Os comandos terminados por virgulas serão exibidos, enquanto aqueles terminados por ponto e vírgula não serão exibidos. Por exemplo:

```
Command Window
>> a=5,A=8;x=2;

a =

    5

fx >> |
```

O MATLAB é sensível a maiúsculas e minúsculas (*case sensitive*), isto é, a variável a não é o mesmo que A .

É possível atribuir valores complexos às variáveis, uma vez que o MATLAB efetua manipulações aritméticas complexas de forma automática. Por exemplo, número imaginário unitário $\sqrt{-1}$ é pré-atribuído à variável i ; com consequência, um valor complexo pode ser atribuído com em:

```
Command Window
>> x=2+3*i

x =

    2.0000 + 3.0000i

fx >> |
```

Observe que o MATLAB permite que o símbolo j seja utilizado para representar o número unitário de entrada, ainda que ele sempre use um i para exibir o número.

Existem diversas variáveis predefinidas, conforme vemos na tabela 1.

Por padrão o MATLAB exibe apenas quatro casas decimais. Caso deseje ser mais preciso, digite o seguinte comando:

Listing 4: janela comando Matlab recortada

```
>> format long
```

Agora, quando π é digitado, o resultado é exibido com 15 algarismos significativos

Variáveis especiais Significado	
<i>ans</i>	Variável usada para exibir os resultados
<i>pi</i>	Número 3.14159
<i>eps</i>	Menor número tal que, quando adicionado a 1, cria um número maior que 1 no computador.
<i>flops</i>	Armazena o número de operações em ponto flutuante realizadas.
<i>inf</i>	Significa infinito
<i>NAN</i> ou <i>nan</i>	Significa não é um número, por exemplo, 0/0.
<i>i</i> e <i>j</i>	Unidade imaginária $\sqrt{-1}$
<i>nargin</i>	Número de argumentos de entrada de uma função
<i>nargout</i>	Número de argumentos de saída de uma função
<i>realmin</i>	Menor número que o computador pode armazenar
<i>realmax</i>	Maior número que o computador pode armazenar

Tabela 1: tabela de variáveis pré-definidas**Listing 5:** janela comando Matlab recortada

```
>> pi
ans      =
      3.14159265358979
```

Para retornar ao formato com quatro casas decimais, digite

Listing 6: janela comando Matlab recortada

```
>> format short
```

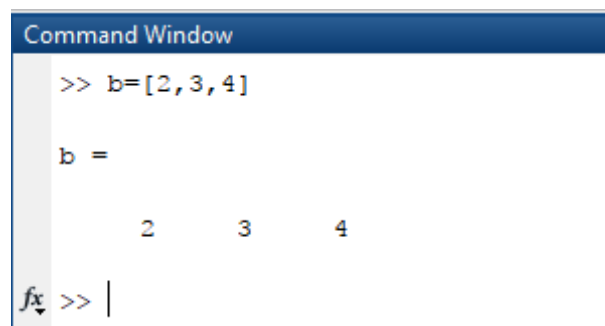
A tabela 2 apresenta um resumo dos comando de formato de exibição (de números) rotineiramente empregados em cálculos científicos e de engenharia. Todos eles têm a sintaxe *format tipo*

4.2.2 Arranjos, vetores e matrizes

Um arranjo ou vetor é um conjunto de variáveis homogêneas (conteúdo de mesmo tipo) identificadas por um mesmo nome e individualizadas por meio de índices. O arranjo pode ser definido elemento por elemento

$$a = [a_1, a_2, \dots, a_n]$$

, separados por espaço em branco ou virgula



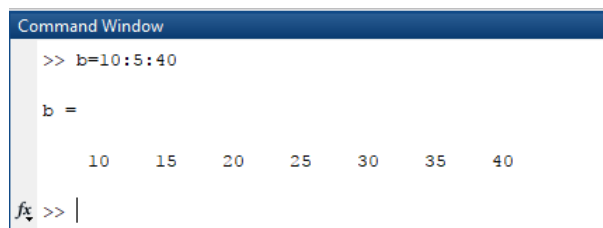
Formatos numéricos		
Tipo	Resultado	Exemplo
short	Formato de ponto fixo com 5 dígitos	3.1416
long	Formato de ponto fixo com 15 dígitos para precisão dupla e 7 dígitos para precisão simples	3.14159265358979
short e	Formato de ponto flutuante com 5 dígitos	3.1416e + 000
long e	Formato de ponto flutuante com 15 dígitos para precisão dupla e 7 dígitos para precisão simples	3.14159265358979e + 000
short g	Melhor entre os formatos de ponto fixo e ponto flutuante com 5 dígitos	3.1416
long g	Melhor entre os formatos de ponto fixo e ponto flutuante com 15 dígitos para precisão dupla e 7 dígitos para precisão simples	3.14159265358979
short eng	Formato de engenharia com pelo menos 5 dígitos e uma potência que é um múltiplo de 3	3.1416e + 000
long eng	Formato de engenharia com exatamente 16 dígitos significativos e uma potência que é um múltiplo de 3	3.14159265358979e + 000
bank	2 dígitos decimais	3.14

Tabela 2: tabela de formatos aceitos pelo MATLAB

O vetor a possui os elementos $a(1) = 2$, $a(2) = 3$ e $a(3) = 4$. Se os valores do arranjo forem igualmente espaçados ele pode ser definido pela expressão

$$b = < \text{valor inicial} > : \text{incremento} : < \text{valor final} >$$

Para gerar um vetor com o primeiro elemento igual a 10, o segundo igual a 15, o terceiro igual a 20 e assim, sucessivamente, até o último igual a 40, basta definir



```

Command Window
>> b=10:5:40

b =

    10    15    20    25    30    35    40
  
```

Se o incremento desejado for igual a 1, então ele poderá ser omitido.

Em vez de se usar incremento, um arranjo também pode ser construído definindo o número desejado de elementos na função **linspace**, cuja sintaxe é:

$$\text{linspace}(< \text{valor inicial} >, < \text{valor final} >, < \text{número de elementos} >)$$

Assim, para criar um arranjo com o primeiro elemento igual a 10, o último igual a 40 e possuindo 7 elementos,

```
Command Window
>> c=linspace(10,40,7)

c =

    10    15    20    25    30    35    40

fx >> |
```

Foi gerado um vetor idêntico a $b = 10 : 5 : 40$ definido acima. Se no comando `linspace` o parâmetro <número de elementos> for omitido, então serão gerados 100 pontos.

Os elementos também podem ser acessados individualmente,

```
Command Window
>> c(2) %segundo elemento de c

ans =

    15

fx >> |
```

```
Command Window
>> c(3:5) %terceiro ao quinto elemento

ans =

    20    25    30

fx >> |
```

```
Command Window
>> c(4:end) %quarto ao último elemento

ans =

    25    30    35    40

fx >> |
```

O endereçamento indireto também é possível, permitindo referenciar os elementos em qualquer ordem,

```
Command Window
>> c([4 1]) %quarto e primeiro elementos

ans =

    25    10

fx >> |
```

Nos exemplos acima os arranjos possuem uma linha e várias colunas, por isso são também chamados vetores linha. Do mesmo modo, podem existir vetores coluna, ou seja, arranjos com várias linhas e uma única coluna. Para criar um vetor coluna elemento por elemento estes devem estar separados por `(;)`

$$v = [v_1, v_2; \dots; v_n]$$

Deste modo, para gerar um vetor coluna com os elementos 2,3,4 fazemos:

```
Command Window

>> v=[2;3;4]

v =

     2
     3
     4

fx >> |
```

Separando os elementos de um arranjo por brancos ou vírgulas são especificados os elementos em diferentes colunas (vetor linha). Por outro lado, separando os elementos por ponto-e-vírgula especifica-se os elementos em diferentes linhas (vetor coluna). Para transformar um vetor linha em um vetor coluna e vice-versa, é usado o operador de transposição (').

```
Command Window

>> c

c =

    10    15    20    25    30    35    40

>> cT=c'

cT =

    10
    15
    20
    25
    30
    35
    40

fx >> |
```

A função ***length*** é usada para se saber o comprimento de um vetor

As matrizes são arranjos bidimensionais ou conjunto de vetores e constituem as estruturas fundamentais do MATLAB e por isso existem várias maneiras de manipulá-las. Uma vez definidas, elas podem ser modificadas de várias formas, como por inserção, extração e rearranjo.

Similarmente aos vetores, para construir uma matriz os elementos de uma mesma linha devem estar separados por branco ou vírgula e as linhas separadas por ponto-e-vírgula ou Enter (ou Return),

$$A = [a_{11}a_{12} \dots a_{1n}; a_{21}a_{22} \dots a_{2n}; \dots; a_{m1}a_{m2} \dots, a_{mn}]$$

Para criar uma matriz A com 2 linhas e 3 colunas,

```
Command Window

>> A=[3 2 1;5 7 8]

A =

     3     2     1
     5     7     8

fx >> |
```

Para modificar um elemento basta atribuir-lhe um novo valor,

```
Command Window

>> A(1,2)=-5

A =

     3    -5     1
     5     7     8

fx >> |
```

Se for atribuído um valor a um elemento não existente, ou seja, além dos elementos da matriz, então o MATLAB aumenta esta matriz automaticamente, sem aviso, preenchendo-a com valores nulos de forma a matriz permanecer retangular,

```
Command Window

>> A(3,5)=10

A =

     3    -5     1     0     0
     5     7     8     0     0
     0     0     0     0    10

fx >> |
```

Seja agora a matriz quadrada B de ordem 3, $B = [123; 456; 789]$

De modo similar aos arranjos, os elementos de uma matriz podem ser referenciados individualmente, tal como, elemento da linha 2 e coluna 3, ou em conjuntos, neste caso usando a notação de arranjo. Por exemplo, os elementos das linhas 1 e 3 e coluna 2,

```
Command Window

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

B =

     1     2     3
     4     5     6
     7     8     9

>> B([1 3],2) %elementos da linha 1 e 3 e coluna 2

ans =

     2
     8

fx >> |
```

A notação de vetor, < valor inicial >:< incremento >:< valor final >, também pode ser usada ou até mesmo *linspace*. Os elementos das linhas 1 e 2 e última coluna são

```

Command Window
>> B

B =

     1     2     3
     4     5     6
     7     8     9

>> B(1:2,end) %elementos da 1 e 2 linha e última coluna

ans =

     3
     6

fx >> |

```

Para remover uma linha ou coluna de uma matriz usa-se a matriz vazia []. Por exemplo para remover a 3 coluna da Matriz A

```

Command Window
>> A

A =

     3    -5     1     0     0
     5     7     8     0     0
     0     0     0     0    10

>> A(:,3)=[]

A =

     3    -5     0     0
     5     7     0     0
     0     0     0    10

fx >> |

```

E posteriormente para remover a linha 3


```
Command Window

>> A

A =

     3     -5     0     0
     5     7     0     0
     0     0     0    10

>> A(3,:)=[]

A =

     3     -5     0     0
     5     7     0     0

fx >> |
```

A função *size* é usada para fornecer o número de linhas e colunas de uma matriz. Ela pode ser usada de duas formas:

```
Command Window

>> B

B =

     1     2     3
     4     5     6
     7     8     9

>> dim =size(B)

dim =

     3     3

fx >> |
```

onde a variável *dim* é um vetor linha com duas posições, contendo o número de linhas e colunas de *B*, respectivamente. A outra forma é

```

Command Window

>> B

B =

     1     2     3
     4     5     6
     7     8     9

>> [nlin ncol]=size(B)

nlin =

     3

ncol =

     3

fx >> |

```

onde as variáveis simples `nlin` e `ncol` contém o número de linhas e colunas de `E`, respectivamente. Se a função `length` for usada em uma matriz, então ela fornecerá o maior valor entre número de linhas e colunas,

O MATLAB tem funções que se aplicam individualmente à cada coluna da matriz produzindo um vetor linha com elementos correspondentes ao resultado de cada coluna. Se a função for aplicada à transposta da matriz ter-se-ão resultados relativos à cada linha da matriz. Se o argumento da função for um vetor em vez de uma matriz, então o resultado será um escalar. Algumas destas funções são mostradas na Tabela 3.

Comandos de MATLAB	
Função	Descrição
<code>sum</code>	soma dos elementos
<code>prod</code>	produto dos elementos
<code>mean</code>	média aritmética
<code>std</code>	desvio padrão
<code>max</code>	maior elemento
<code>min</code>	menor elemento
<code>sort</code>	ordenar os elementos
<code>magic</code>	cria uma matriz aleatória

Tabela 3: Algumas funções para operações de matrizes no MATLAB

Existem também várias funções para manipulação de matrizes dentre as quais destacam-

se:

Função	Descrição
diag	se o argumento for um vetor, então cria uma matriz diagonal com os elementos do vetor; se o argumento for uma matriz, então produz um vetor coluna contendo os elementos da diagonal.
tril	obtem a parte triangular inferior de uma matriz
triu	obtem a parte triangular superior de uma matriz.

Tabela 4: Algumas funções para operações de matrizes no MATLAB

vemos a seguir (figura) alguns exemplos:

```

Command Window
>> A=magic(4)

A =

    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1

>> sum(A)

ans =

    34    34    34    34

>> prod(A)

ans =

    2880    2156    2700    1248

fx >> |

```

Também são disponíveis várias matrizes elementares de grande utilidade, como as mostradas na Tabela 5. Se um único parâmetro for provido, então a matriz será quadrada de ordem igual ao valor do parâmetro. Se forem dois parâmetros, então ela será retangular com as dimensões iguais aos valores desses parâmetros. Por exemplo,

```

Command Window
>> Z=zeros(4)

Z =

     0     0     0     0
     0     0     0     0
     0     0     0     0
     0     0     0     0

fx >> |

```

Função	Descrição
zeros	matriz com todos os elementos iguais a zero.
ones	matriz com todos os elementos iguais a um.
eye	matriz identidade ou parte dela
rand	elementos aleatórios uniformemente distribuídos entre 0 e 1
randn	elementos aleatórios com distribuição normal com média 0 e desvio padrão 1.

Tabela 5: Algumas funções para operações de matrizes no MATLAB

```

Command Window
>> O=ones(3,4)

O =

     1     1     1     1
     1     1     1     1
     1     1     1     1

fx >> |

Command Window
>> rand(3)

ans =

    0.8147    0.9134    0.2785
    0.9058    0.6324    0.5469
    0.1270    0.0975    0.9575

fx >> |

```

As funções `rand` e `randn` produzem números pseudoaleatórios e a sequência gerada é determinada pelo estado do gerador. Quando o MATLAB for ativado, é atribuído um mesmo valor inicial ao estado, e portanto, a sequência de números gerados será a mesma. Depois de gerada uma sequência o estado do gerador é mudado. Para colocar o gerador no estado inicial usa-se o comando:

```
reset(RandStream.getDefaultStream);
```

4.3 Variáveis literais

Uma variável pode conter uma cadeia de caracteres em vez de um número. Estes caracteres são manipulados iguais aos vetores linha. A cadeia de caracteres deve estar delimitada por apóstrofes para ser atribuída a uma variável literal. Assim,

```
Command Window

>> s='MatLAB'

s =

    'MatLAB'

>> s(1)

ans =

    'M'

fx >> |
```

4.4 Variáveis especiais

O MATLAB tem diversas variáveis especiais, as quais estão listadas na Tabela 1. Com excessão de *ans*, *i* e *j* estas variáveis não devem ser redefinidas. Para mais informações sobre matrizes e outras variáveis use `help elmat`.

4.5 Expressões aritméticas

São disponíveis as operações aritméticas básicas mostradas na Tabela 6. As expressões são avaliadas da esquerda para a direita, tendo a potenciação maior ordem de precedência, seguida pela multiplicação e divisão (ambas tendo igual precedência) e seguida pela adição e subtração (com igual precedência). Os parênteses podem ser usados para alterar a ordem de precedência.

Operação	Expressão	Operador	Exemplo
adição	$a + b$	+	$1 + 2$
subtração	$a - b$	-	$5.1 - 4.7$
multiplicação	$a \times b$	*	$6 * 7$
divisão	$a \div b$	/ ou \	$7/4$
potenciação	a^b	^	2^10

Tabela 6: Alguns operadores aritméticos no MATLAB

As operações básicas entre vetores só são definidas quando estes tiverem o mesmo tamanho e orientação (linha ou coluna). Estas operações básicas são apresentadas na Tabela 7. As operações de multiplicação, divisão e potenciação envolvendo vetores antecidas pelo caractere `(.)`, significa que estas operações são efetuadas elemento a elemento.

Sejam $a = [a_1 a_2 \dots a_n]$ e $b = [b_1 b_2 \dots b_n]$ e c um escalar.

De modo similar às operações vetoriais, existem as operações matriciais básicas, as quais estão compiladas na Tabela . O operador `\` envolvendo matrizes e vetores está relacionado com solução de sistemas lineares.

Operação	Expressão	Resultado
adição escalar	$a + c$	$[a_1 + ca_2 + c \dots a_n + c]$
adição vetorial	$a + b$	$[a_1 + b_1a_2 + b_2 \dots a_n + b_n]$
multiplicação escalar	$a * c$	$[a_1 * ca_2 * c \dots a_n * c]$
multiplicação vetorial	$a. * b$	$[a_1 * b_1a_2 * b_2 \dots a_n * b_n]$
divisão à direita	$a./b$	$[a_1/b_1a_2/b_2 \dots a_n/b_n]$
divisão à esquerda	$a.\backslash b$	$[b_1/a_1b_2/a_2 \dots b_n/a_n]$
potenciação	$a.^c$	$[a_1^ca_2^c \dots a_n^c]$
	$c.^a$	$[c^a_1c^a_2 \dots c^a_n]$
	$a.^b$	$[a_1^b_1a_2^b_2 \dots a_n^b_n]$

Tabela 7: Algumas operações vetoriais no MATLAB

Seja c um escalar e

$$A = [a_{11}, a_{12}, \dots, a_{1n}; a_{21}, a_{22}, \dots, a_{2n}; a_{m1}, a_{m2} \dots a_{mn}]$$

e

$$B = [b_{11}, b_{12}, \dots, b_{1n}; b_{21}, b_{22}, \dots, b_{2n}; \dots; b_{m1}, b_{m2}, \dots, b_{mn}]$$

Operação	Expressão	Resultado
adição escalar	$A + c$	$a_{ij} + c$
adição matricial	$A + B$	$a_{ij} + b_{ij}$
multiplicação escalar	$A * c$	$a_{ij} * c$
multiplicação por elemento	$A. * B$	$a_{ij} * b_{ij}$
multiplicação matricial	$A * B$	AB
divisão por elemento	$A./B$	a_{ij}/b_{ij}
divisão a esquerda	$A.\backslash B$	b_{ij}/a_{ij}
potenciação	$A.^c$	a_{ij}^c
	A^c	A^c
	$c.^A$	$c^{a_{ij}}$
	c^A	c^A
	$A.^B$	$a_{ij}^{b_{ij}}$

Tabela 8: Algumas operações vetoriais no MATLAB

Como pode ser esperado de uma linguagem para aplicações nas áreas científicas e técnicas, o MATLAB oferece várias funções importantes para Matemática, Ciências e Engenharia. A Tabela 9 apresenta algumas funções matemáticas elementares, e uma lista mais completa pode ser obtida usando o comando `help elfun`. As funções matemáticas especializadas podem ser listadas usando `help specfun`.

4.6 Expressões lógicas

Uma expressão se diz lógica quando os operadores são lógicos e os operandos são relações e/ou variáveis do tipo lógico. Uma relação é uma comparação realizada entre valores do mesmo tipo. A natureza da comparação é indicada por um operador relacional conforme a Tabela .

Função	Descrição	Função	Descrição
<code>acos</code>	arco cosseno	<code>cos</code>	cosseno
<code>acosh</code>	arco co-seno hiperbólico	<code>cosh</code>	cosseno hiperbólico
<code>acot</code>	arco co-tangente	<code>cot</code>	cotangente
<code>acoth</code>	arco co-tangente hiperbólica	<code>coth</code>	cotangente hiperbólica
<code>acsc</code>	arco co-secante	<code>csc</code>	cossecante
<code>acsch</code>	arco co-secante hiperbólica	<code>csch</code>	cossecante hiperbólica
<code>asec</code>	arco secante	<code>sec</code>	secante
<code>asech</code>	arco secante hiperbólica	<code>sech</code>	secante hiperbólica
<code>asin</code>	arco seno	<code>sin</code>	seno
<code>asinh</code>	arco seno hiperbólico	<code>sinh</code>	seno hiperbólico
<code>atan</code>	arco tangente	<code>tan</code>	tangente
<code>atanh</code>	arco tangente hiperbólico	<code>tanh</code>	tangente hiperbólica
<code>atan2</code>	arco tangente de 4 quadrantes		

Tabela 9: Algumas operações vetoriais no MATLAB

Função	Descrição	Função	Descrição
<code>exp</code>	exponencial	<code>abs</code>	valor absoluto
<code>log</code>	logaritmo natural	<code>angle</code>	ângulo de fase
<code>log10</code>	logaritmo decimal	<code>conj</code>	conjugado do complexo
<code>sqrt</code>	raiz quadrada	<code>imag</code>	parte imaginária
<code>nthroot</code>	raiz qualquer índice	<code>real</code>	parte real
<code>ceil</code>	arredonda em direção a $+\infty$	<code>lcm</code>	mínimo múltiplo comum
<code>fix</code>	arredonda em direção a 0	<code>rem</code>	resto da divisão
<code>floor</code>	arredonda em direção a $-\infty$	<code>round</code>	arredonda em direção ao inteiro mais próximo
<code>gcd</code>	máximo divisor comum	<code>sign</code>	sinal

Tabela 10: Algumas operações vetoriais no MATLAB

4.7 Gráficos

Uma das grandes habilidades do MATLAB é a facilidade para produzir gráficos de ótima qualidade. Nesta seção serão vistos como gerar gráficos bi e tridimensionais e os modos de gravá-los em arquivos para que possam ser incluídos em textos.

Para gerar gráficos bidimensionais podem ser usados as versáteis funções *plot* e *fplot*.

4.7.1 Função plot

sintaxe:

$$\text{plot}(x_1, y_1, ' < \text{tipoDeLinha} >', x_2, y_2, ' < \text{tipoDeLinha} >', \dots)$$

onde x e y são vetores contendo as abscissas e ordenadas dos pontos a serem exibidos, respectivamente e $<\text{tipoDeLinha}>$ é uma cadeia de 1 a 4 caracteres que especifica a cor e o estilo da linha, os quais são mostrados na Tabela .

As funções são muito usadas na avaliação de fórmulas para uma série de argumentos. Considere que a velocidade de um saltador de *bungee jumping* em queda livre pode ser

Operador	Descrição	Operador	Descrição
>	maior que	<	menor que
>=	maior igual	<=	menor igual
==	igual	~=	conjugado do complexo
&&	operador e		operador ou
~	negação		

Tabela 11: Algumas operações lógicas no MATLAB

Simbolo	Cor	Simbolo	Estilo de linha
y	amarela	.	ponto
m	lilás	o	círculo
c	turquesa	x	marca x
r	vermelho	+	mais
g	verde	*	asterisco
b	azul	-	linha sólida
w	branco	:	linha pontilhada
k	preto	-.	linha de traço e ponto
		--	linha tracejada

Tabela 12: Tipos de linha e cores adotadas pelo MATLAB

calculada pela expressão:

$$v = \sqrt{\frac{gm}{c_d}} \tanh \left(\sqrt{\frac{gc_d}{m}} t \right) \quad (2)$$

Onde v é a velocidade (m/s), g é a aceleração da gravidade ($9.81m/s^2$), m é a massa em Kg , c_d é o coeficiente de arraste (Kg/m) e t tempo (s). Vamos criar um vetor coluna t que contenha valores de 0 a 20 em passos de 2:


```
Command Window

>> t=[0:2:20] '

t =

    0
    2
    4
    6
    8
   10
   12
   14
   16
   18
   20

fx >> |
```

Observe que o numero de elementos de t , com a função `length` é 11.
Atribua valores aos parâmetros

```
Command Window

>> g=9.81;m=68.1;cd =0.25;
fx >> |
```

O MATLAB permite avaliar uma fórmula tal como $v = f(t)$, onde a fórmula é calculada para cada valor do arranjo t , e o resultado é atribuído para uma posição correspondente no arranjo v . Para o caso analisado:

```
Command Window

>> v=sqrt(g*m/cd)*tanh(sqrt(g*cd/m)*t)

v =

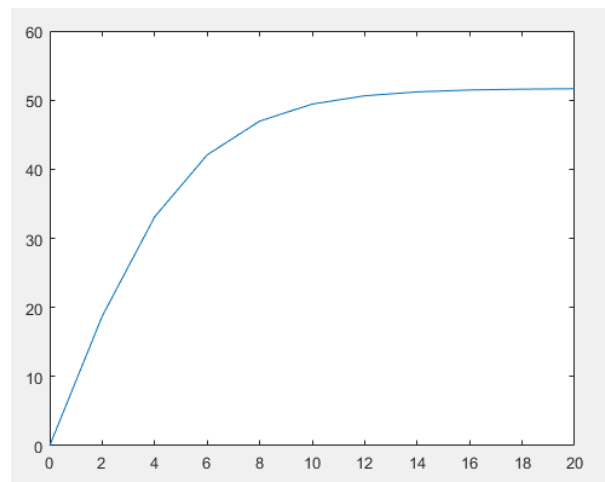
    0
 18.7292
 33.1118
 42.0762
 46.9575
 49.4214
 50.6175
 51.1871
 51.4560
 51.5823
 51.6416

fx >> |
```

Com o MATLAB, é possível criar gráficos de forma rápida e conveniente. Para criar um gráfico dos arranjos t e v a partir dos dados anteriores digite:

Listing 7: janela comando Matlab recortada

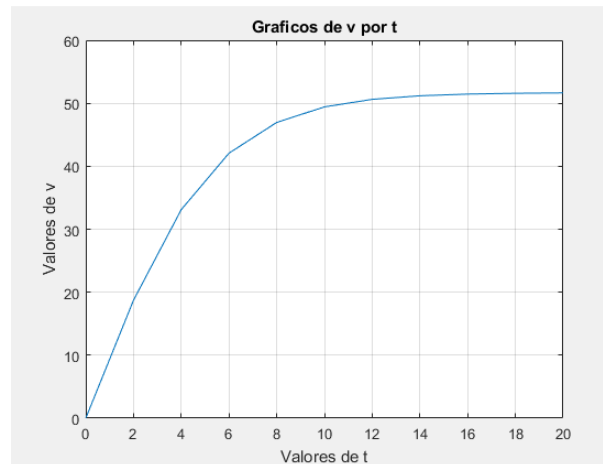
```
>> plot(t,v)
```



É possível personalizar um pouco o gráfico com os comandos a seguir:

Listing 8: janela comando Matlab recortada

```
>> title('Grafico de v por t')
>> xlabel('Valores de t')
>> ylabel('Valores de v')
>> grid
```



O comando *plot* exibe, por padrão (default), uma linha sólida de cor azul. Caso o usuário decida plotar cada ponto com um símbolo (marcador), pode-se incluir um especificador delimitando por aspas simples no comando *plot*. a tabela 13 mostra os especificadores disponíveis. Para um marcador do tipo círculo abertos, por exemplo

Cores		Símbolos		Símbolos	
Azul	b	Ponto	.	Sólida	-
Verde	g	Círculo	o	Pontilhada	:
Vermelho	r	Símbolo X	x	Traço-ponto	-.
Ciano	c	Mais	+	Tracejada	--
Magenta	m	Estrela	*		
Amarelo	y	Quadrado	s		
Preto	k	Diamante	d		
Branco	w	Triângulo (para baixo)	v		
		Triângulo (para cima)	^		
		Triângulo (para esquerda)	<		
		Triângulo (para direita)	>		
		Pentagrama (estrela de cinco pontas)	p		
		Hexagrama (estrela de seis pontos)	h		

Tabela 13: Especificadores de cores, símbolos (ou marcadores) e tipos de linha

Listing 9: janela comando Matlab recortada

```
>> plot(t,v,'o')
```

É possível combinar diversos especificadores: caso se queira utilizar marcadores quadrados verdes, conectados por linhas tracejadas verdes, pode-se inserir o comando

Listing 10: janela comando Matlab recortada

```
>> plot(t,v,'s—g')
```

Também é possível controlar a largura da linha, assim como o tamanho do marcador e as cores das bordas e do preenchimento. O comando a seguir, por exemplo, utiliza uma linha mais larga (largura 2 pontos), tracejada e cor ciano, para conectar marcadores maiores (10 ponto), com forma de diamante, bordas pretas e preenchimento de cor magenta:

Listing 11: janela comando Matlab recortada

```
>> plot(x,y,'—dc','LineWidth',2,'MarkerSize',10,'MarkerEdgeColor','k','  
MarkerFaceColor','m')
```

Observe que a largura de linha padrão é um ponto; para os marcadores, o padrão são seis pontos com borda azul e sem preenchimento (cor branca).

O MATLAB permite exibir mais de um conjunto de dados no mesmo gráfico. Por exemplo, uma maneira alternativa de conectar cada marcador de ponto com uma linha reta seria digitar:

Listing 12: janela comando Matlab recortada

```
>> plot(t,v,t,v,'o')
```

Deve-se mencionar que, inevitavelmente, os gráficos anteriores são apagados toda vez que o comando `plot` é executado. O comando *hold on* mantém o gráfico corrente e todas as propriedades dos eixos, de modo que comandos gráfico adicionais possam ser acrescentados ao gráfico existente; já o comando *hold off* retorna ao modo padrão. Por exemplo, se tivéssemos digitado os comando a seguir, o gráfico final exibiria apenas marcadores:

Listing 13: janela comando Matlab recortada

```
>> plot(t,v)  
>> plot(t,v,'o')
```

Os comando a seguir por sua vez, resultariam em um gráfico que exibiria tanto as linhas quanto os marcadores:

Listing 14: janela comando Matlab recortada

```
>> plot(t,v)  
>> hold on  
>> plot(t,v,'o')  
>> hold off
```

Além do *hold*, outro comando útil é o `subplot`, que permite dividir a janela do gráfico em subjanelas ou painéis, e tem a sintaxe:

Listing 15: janela comando Matlab recortada

```
subplot(m,n,p)
```

Esse comando divide a janela do gráfico em uma matriz m por n , com gráficos menores em cada célula da “matriz” e seleciona o p -éssimo gráfico como o corrente.

Podemos demonstrar o comando `subplot` por meio de um comando simples para gerar gráficos tridimensionais, o `plot3`, que tem a sintaxe

Listing 16: janela comando Matlab recortada

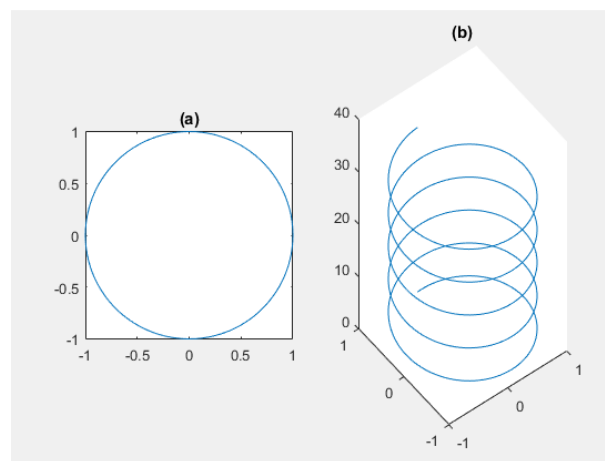
```
plot3(x,y,z)
```

onde x, y, z são três vetores de mesmo comprimento. O resultado é uma linha no espaço tridimensional através dos pontos cujas coordenadas são os elemento de x, y e z .

O gráfico de uma hélice fornece um bom exemplo para ilustrar essa teoria. Primeiro, vamos traçar um círculo com a função bidimensional `plot` utilizando a representação paramétrica $x = \sin(t)$ e $y = \cos(t)$. Empregamos o comando `subplot` para, em seguida, acrescentar o gráfico tridimensional.

Command Window

```
>> t=0:pi/50:10*pi;  
>> subplot(1,2,1);  
>> plot(sin(t),cos(t))  
>> axis square  
>> title(' (a) ')  
>> subplot(1,2,2);  
>> plot3(sin(t),cos(t),t);  
>> title(' (b) ')  
fx >> |
```



5

Exercícios

Exercício 1.

Use a função *linspace* para criar vetores idênticos aos abaixo criando com a notação dois-pontos:

1. $t = 4 : 6 : 35$
2. $x = -4 : 2$

Exercício 2.

Considere circuito elétrico simples consistindo de um resistor, um capacitor e um indutor. a carga no capacitor $q(t)$ como uma função do tempo pode ser calculada como

$$q(t) = q_0 e^{-Rt/(2L)} \cos \left(\sqrt{\frac{1}{LC} - \left(\frac{R}{2L}\right)^2} t \right)$$

onde t é o tempo, q_0 é a carga inicial, R é a resistência, L é a indutância e C é a capacitância. Use o MATLAB para gerar um gráfico dessa função de $t = 0$ a $t = 0.8$ dado que $q_0 = 10$, $R = 60$, $L = 9$ e $C = 0.00005$

Exercício 3.

A função de densidade de probabilidade normal é uma curva em forma de sino que pode ser representada como

$$f(z) = \frac{1}{\sqrt{2\pi}} e^{-z^2/2}$$

Use o MATLAB para gerar um gráfico dessa função de $z = -5$ até 5. Identifique as ordenadas como frequência e as abscissas com z

Exercício 4.

A expansão em série de Maclaurin para o cosseno é:

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \dots$$

Use o MATLAB para criar um gráfico do cosseno com um gráfico da expansão em série até o termo $\frac{x^8}{8!}$. Utilize a função nativa *factorial* no calculo da expansão em série, e para as abscissas utilize o intervalo $x = 0$ até $3\pi/2$

Exercício 5.

A equação da *Borboleta* é dada pelas seguintes equações paramétricas:

$$\begin{cases} x = \sin(t) \left(e^{\cos(t)} - 2 \cos(4t) - \sin^5\left(\frac{t}{12}\right) \right) \\ y = \cos(t) \left(e^{\cos(t)} - 2 \cos(4t) - \sin^5\left(\frac{t}{12}\right) \right) \end{cases}$$

Gere valores de x e y para $t \in [0, 100]$ com $\Delta t = \frac{1}{16}$ e construa gráficos de:

1. x, y, t
2. y, x

Use o comando subplot para organizar esses gráficos verticalmente e faça o gráfico de (b) quadrado. inclua títulos e identificações dos eixos em ambos os gráficos e uma legenda em (a). Para (a) empregue uma linha tracejada para y de modo a distingui-la de x .

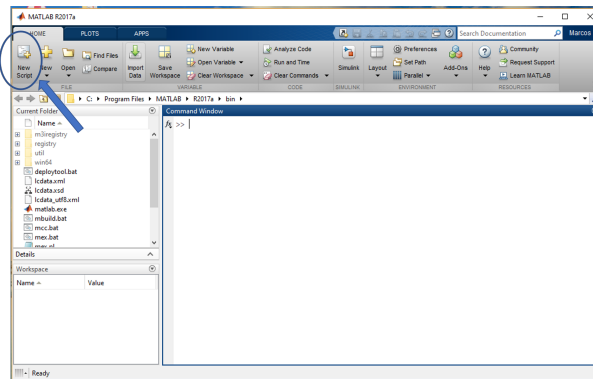
6**Programação básica no MATLAB****6.1 Arquivos-M**

Uma maneira comum de operar o MATLAB é inserir um comando por vez na command window. Os programas de MATLAB ou arquivos M (*M-files*), no entanto, fornecem uma maneira alternativa - e mais dinâmica - de efetuar operações: um programa de MATLAB consiste em uma série de instruções ou comando que podem ser executados todos de uma vez. Observe que a nomenclatura “*arquivo M*” ou “*M-file*” vem do fato de os programas desenvolvidos no MATLAB serem salvos com extensão *.m*. Tais arquivos podem ser de dois tipos: programas comuns (Scripts) ou funções (function files).

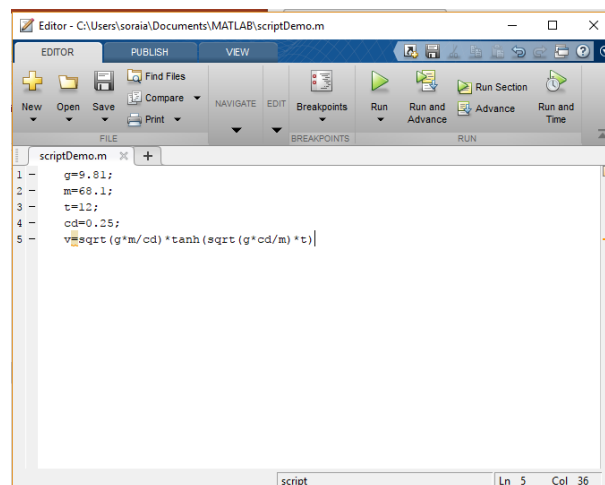
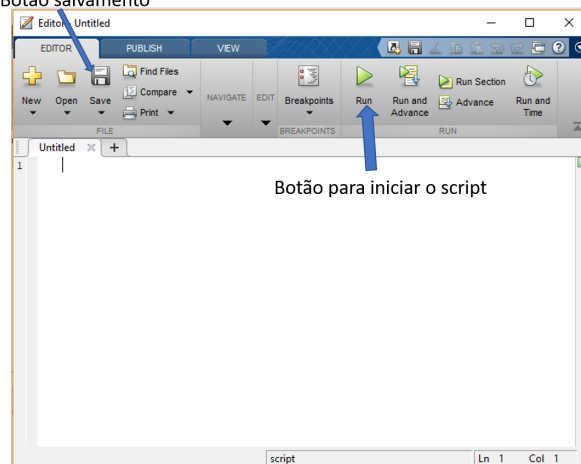
6.1.1 Programas comuns - *script files*

Um *script file* trata-se de uma série de comandos do MATLAB que são salvos em um arquivo e podem ser executados em mais de uma ocasião. Para usar o script, digite o nome do arquivo na command window ou selecione a opção **Run** no menu **Debug** da janela edit window.

Vamos a um exemplo de uso de um script.



Botão salvamento



Salve o arquivo com scriptDemo.m. Retorne a command window e digite

Listing 17: janela comando Matlab recortada

```
>>scriptDemo
```

```
Command Window

>> scriptDemo

v =

    50.6175

fx >> |
```

6.2 Funções (function files)

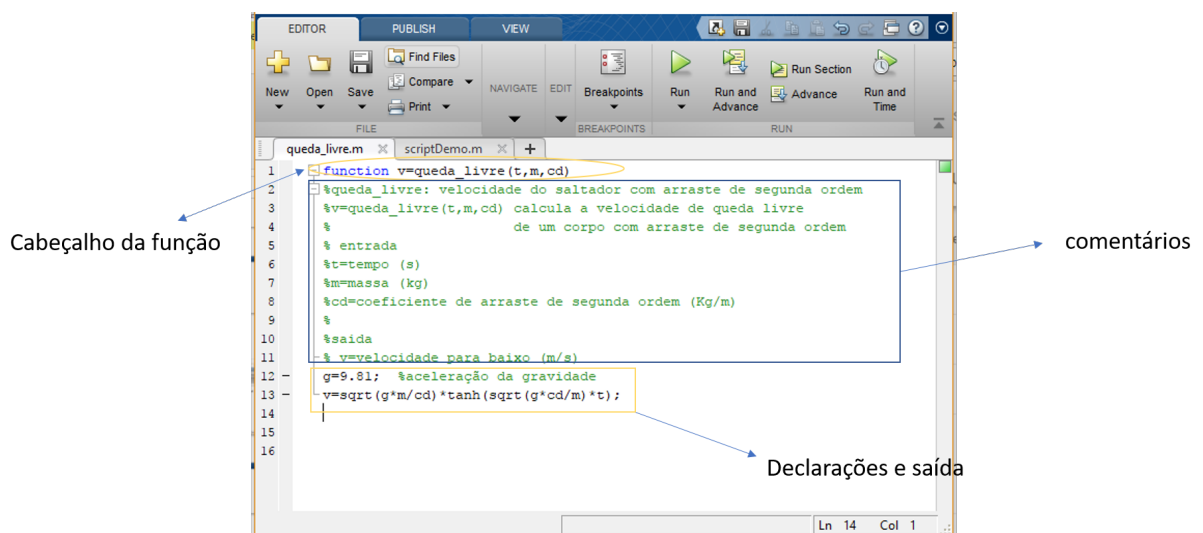
As funções ou *function files* são arquivos *.m* que começam com a palavra chave *function*. Ao contrário dos scripts, as funções podem aceitar argumentos de entrada e devolver argumentos de saída. Portanto, elas são análogas as funções definidas em linguagens de programação tais como fortran, Visual Basic ou C.

A sintaxe para a função pode ser representada genericamente como

Listing 18: janela comando Matlab recortada

```
function var_de_saida = nome_da_funcao(lista_de_argumentos)
%comentarios
declaracoes
var_de_saida = valor;
```

Onde *declaracoes* representa o conjunto de comandos do MATLAB que são chamados dentro da função, conforme o exemplo.



```
Command Window

>> queda_livre(12,68.1,0.25)

ans =

    50.6175

fx >> |
```


6.3 Entrada e Saída

A função *input*. Esta função solicita ao usuário diretamente da command window uma informação numérica ou em forma de texto. Sua sintaxe é:

Listing 19: janela comando Matlab recortada

```
variavel = input('mensagem_de_solicitacao')
```

Quando o usuário insere um valor este é atribuído a **variável**. A função input também pode receber uma entrada na forma de texto. Para isso basta utilizar um 's' na lista de argumentos.

Listing 20: janela comando Matlab recortada

```
variavel = input('mensagem_de_solicitacao','s')
```

A função *disp* é a responsável pela saída ou seja exibir algum valor. Sua sintaxe é simples:

Listing 21: janela comando Matlab recortada

```
disp(valor)
```

onde *valor* é a saída esperada para qualquer operação, na forma de constante, variável ou string.

A função *fprintf*. Com esta função é possível controlar melhor a exibição de informações. Sua sintaxe é simples:

Listing 22: janela comando Matlab recortada

```
fprintf('formato',x,...)
```

onde *formato* representa uma string associada a comandos para exibição do texto e *x* representa o conjunto de variáveis que serão associadas a string.

Tabela 14: Alguns códigos de formato e de controle empregados com a função

Código de formato	Descrição
%d	formato inteiro
%e	Formato científico com e minúsculo
%E	Formato científico com E maiúsculo
%f	Formato de ponto fixo
%g	O formato mais compacto de %e ou %f
Código de controle	Descrição
\n	Inicia uma nova linha
\t	Tabulação horizontal

A função *fprintf* também pode ser usada para exibir diversos valores por linha com diferentes formatos. Por exemplo,

7

Programação Estruturada

Os programas mais simples executam instruções sequencialmente, isto é, linha por linha, iniciando do topo da função e descendo até a última instrução. Como algumas sequências podem ser um tanto complicadas, todas as linguagens computacionais de alto nível incluem declarações que permitem aos programas tomar caminhos não sequenciais. Essas declarações podem ser classificadas como:

- a) decisões
- b) laços

7.1 Decisões

A estrutura de decisão *if*. Essa estrutura permite, executar um conjunto de declarações se uma condição lógica for verdadeira. Sua sintaxe é:

Listing 23: janela comando Matlab recortada

```
if condicao
    declaracoes
end
```

onde *condicao* é uma expressão lógica que pode ser verdadeira (1) ou falsa (0). por exemplo. a seguir tem-se uma função simples que determina se uma nota escolar é suficiente ou não para aprovação:

Listing 24: script MATLAB

```
function classificador(nota)
%Determinar se a nota e suficiente para aprovacao
%entrada:
%          nota= valor numerico de 0 a 100
%saida:
%          mensagem
if nota >=60
disp('Nota suficiente para aprovacao')
end
```

7.2 Condições lógicas

A forma mais simples da *condicao* é uma expressão relacional única que compara dois valores, na forma:

$$valor_1 \quad relacao \quad valor_2$$

onde *valor₁* e *valor₂* são constantes, variáveis ou expressões e a *relacao* é um operador relacional listado na Tabela 11.

7.3 Estrutura if-else-end

Caso haja duas alternativas, uma outra estrutura condicional deve ser usada

Listing 25: janela comando Matlab recortada

```
if condicao
    declaracoes_1
else
    declaracoes_2
end
```

Se o resultado da expressão lógica $< condicao >$ for 1 (verdadeiro), então a lista contendo $< declaracoes_1 >$ será executada. Se $< condicao >$ for 0 (falso), então será a lista $< declaracoes_2 >$ a ser executada. A sequência de comandos,

Listing 26: janela comando Matlab recortada

```
a = input('Entre com o valor de a: ');
if a > 0
    b = log(a)
else
    b = exp(a)
end
```

7.4 Estruturas de repetição

A estrutura de repetição faz com que uma sequência de comandos seja executada repetidamente até que uma dada condição de interrupção seja satisfeita. O MATLAB possui duas estruturas de repetição, as estruturas *for-end* e a *while-end*.

7.5 Estrutura for-end

A estrutura *for-end* permite que um grupo de comandos seja repetido um número de vezes definido. Sua sintaxe é

Listing 27: Laço de repetição

```
for<variavel>=<arranjo>
<comandos>
end
```

onde $< variavel >$ é a variável de controle que assume todos os valores contidos no vetor $< arranjo = inicio : passo : fim >$. Assim, o número de repetições da lista $< comandos >$ é igual ao número de elementos do vetor $< arranjo >$. A variável de controle não pode ser redefinida dentro da estrutura for-end. Os comandos

Listing 28: Exemplo de laço de repetição

```
n = input('Valor de n: '); s = 0; n2 = n^2;
for i = 1:2:2*n-1
    s = s + i;
end
```

7.6 Estrutura while-end

A estrutura while-end repete um grupo de comandos um número de vezes indefinido. Sua sintaxe é:

Listing 29: Sintaxe laço de repetição

```
while <condicao>
    <comandos>
end
```

Enquanto a expressão lógica *< condicao >* resultar em verdadeiro a lista *< comandos >* será repetida. Por exemplo, para determinar a precisão de um computador, os comandos:

Listing 30: Sintaxe laço de repetição

```
eps1 = 1; n = -1;
while 1 + eps1 > 1
    epsilon = eps1; n = n + 1; eps1 = eps1 / 2;
end
n, epsilon, eps
```

8

Funções e comandos úteis em Cálculo Numérico

8.1 Funções anônimas

As funções anônimas permitem criar funções simples sem desenvolver um arquivo-*M* e podem ser definidas na command window com a seguinte sintaxe

Listing 31: Sintaxe para funções anônimas

```
fhandle=@(lista_de_argumentos) expressao
```

onde fhandle é o function handle ou identificador da função que pode ser usado para chamar a função, *lista_de_argumentos* é uma lista de argumentos de entrada separados por vírgula a serem passados para a função e *expressao* é qualquer expressão válida do MATLAB. por exemplo

Listing 32: funções anônimas

```
>>f1=@(x,y) x^2+y^2;
```

Uma vez definidas nas command window, tais funções podem ser usadas como qualquer outra função:

Listing 33: funções anônimas

```
>>f1(3,4)
ans =
    25
```

9

Exercícios

Exercício 1.

Há formulas econômicas disponíveis para calcular pagamentos anuais de empréstimos. Considere que você tenha emprestado uma quantidade de dinheiro P e tenha concordado em pagar em n pagamentos anuais com taxa de juros i . A fórmula para calcular o pagamento anual A é:

$$A = P \frac{i(1+i)^n}{(1+i)^n - 1}$$

Escreva uma função no MATLAB para calcular A . Teste-a com $P = R\$100.000,00$ e uma taxa de juros de $3.3\%a.a.$ Calcule os resultados para $n = 1, 2, 3, 4$ e 5 e mostre o resultado em um tabela com cabeçalhos e colunas para n e A

Exercício 2.

O volume V de líquido em um cilindro horizontal oco de raio r e comprimento L é relacionado à profundidade do líquido h por:

$$V = \left[r^2 \cos^{-1} \left(\frac{r-h}{r} \right) - (r-h) \sqrt{2rh - h^2} \right] L$$

Desenvolva uma função no MATLAB para criar um gráfico do volume versus a profundidade e Teste seu programa.

Exercício 3.

O método babilônico, um antigo sistema para aproximação da raiz quadrada de qualquer número positivo a , pode ser formulado como:

$$x = \frac{x + \frac{a}{x}}{2}$$

Escreva uma função bem estruturada no MATLAB com base na estrutura de laço **while** para implementar esse algoritmo. Use indentação adequada, de modo que a estrutura do código fique clara. Em cada passo, estime o erro em sua aproximação como:

$$\epsilon = \left| \frac{x_{novo} - x_{velho}}{x_{novo}} \right|$$

Repita o laço até que ϵ seja menor ou igual a um valor específico e projete seu programa de modo que ele retorne o resultado e o erro. Certifique-se que ele possa avaliar a raiz quadrada de números iguais ou menores que zero. Para o ultimo caso, exiba o resultado como um número imaginário, por exemplo, a raiz quadrada de -4 seria retornada com $2i$. Teste seu programa avaliando $a = 0, 2, 10$ para $\epsilon = 1 \times 10^{-4}$

Exercício 4.

Funções definidas por partes às vezes são úteis quando a relação entre uma variável dependente e uma variável independente não pode ser adequadamente representada por uma única equação. Por exemplo, a velocidade de um foguete poderia ser descrita por:

$$v(t) = \begin{cases} 10t^2 - 5t & 0 \leq t \leq 8 \\ 624 - 3t & 8 < t \leq 16 \\ 36t + 12(t-16)^2 & 16 < t \leq 26 \\ 213e^{-0.1(t-26)} & t > 26 \\ 0 & c.c. \end{cases}$$

Desenvolva uma função no MATLAB para calcular v como uma função de t . em seguida, desenvolva um script que utilize essa função para gerar um gráfico de $v \times t$ para $t = -5 \dots 50$

Exercício 5.

Desenvolva uma função para produzir a animação de uma partícula se movendo em um círculo em coordenadas cartesianas com base em coordenadas radiais. Assuma um raio constante, r e permita que o ângulo θ , aumentar de $0 \dots 2\pi$ em incrementos iguais. Teste a função

10

Erros de arredondamento e de Truncamento

10.1 Conceito

A noção de erro está presente em todos os campos do Cálculo Numérico. De um lado, os dados, em si, nem sempre são exatos e, de outro lado, as operações sobre valores não exatos propagam esses erros a seus resultados. Finalmente, os próprios métodos numéricos, frequentemente métodos aproximados, buscam a minimização dos erros, procurando resultados o mais próximo possível do que seriam valores exatos.

Erro é a diferença entre o valor verdadeiro(x) e a aproximação obtida(\bar{x}).

Assim:

$$\varepsilon_A = x - \bar{x}$$

onde ε_t é utilizado para designar o valor exato do erro. Observe que o erro verdadeiro é geralmente expresso como um valor absoluto e referido como **erro absoluto**.

Esta definição não leva em conta a ordem de grandeza do valor que esta sendo examinado, ou seja, um erro de um centímetro é muito mais significativo quando se mede um rebite de que uma ponte. Um aforma de considerar o valor das quantidades que estão sendo calculadas é normalizar o erro com o valor verdadeiro, assim:

$$\varepsilon_R = \frac{x - \bar{x}}{x} \times 100\%$$

Ao se realizar cálculos normalmente não há preocupações com o sinal do erro, mas interesse em saber se o valor absoluto percentual é menor que uma tolerância percentual pré-especificada

$$|\varepsilon_A| < \varepsilon_S$$

Essa relação é chamada **critério de parada**: se ela for satisfeita, supõe-se que o resultado esteja dentro do nível aceitável pré-especificada ε_S .

Exemplo

Em matemática, as funções podem ser representadas por séries infinitas. Por exemplo, a função exponencial pode ser calculada usando-se

$$e^x \approx \sum_{n=0}^{\infty} \frac{x^n}{n!} \quad \forall x$$

Esta equação é chamada de expansão em série de Maclaurin.

Vamos estimar o valor de $e^{0.5}$ considerando que o valor verdadeiro, calculado previamente, é $e^{0.5} = 1.648721$

A função do MATLAB que implementa essa operação será:

Listing 34: Programa MATLAB solução

```

1 function erroExp(x,nMax,vlEsperado)
2 %x= valor do expoente
3 %nMax = Numero de termos
4 %iterMax = Numero maximo de iteracoes
5 clc
6 sol=0;
7 fprintf('-----\n
   ');
8 fprintf('Termo Valor Esperado Resultado Erro Absoluto Erro Relativo\n
   ');
9 fprintf('-----\n
   ');
10 n=0;
11 while(n<nMax)
12     sol_previa=sol;
13     sol=sol+x^n/factorial(n);
14     ea=abs(sol-sol_previa);
15     er=(ea/sol)*100;
16     fprintf('%d      %f      %f      %f      %6.2f%%\n',n,vlEsperado,
17             sol,ea,er);
18     n=n+1;
19 end

```

Terá então como saída:

Command Window				
Termo	Valor Esperado	Resultado	Erro Absoluto	Erro Relativo
0	1.648721	1.000000	1.000000	100.00%
1	1.648721	1.500000	0.500000	33.33%
2	1.648721	1.625000	0.125000	7.69%
3	1.648721	1.645833	0.020833	1.27%
4	1.648721	1.648438	0.002604	0.16%

f_x >> |

10.2 Erro de arredondamento

Ao se aplicar um método numérico, os erros devidos aos valores iniciais, intermediários e finais conduzem a um erro global (diferença entre o exato e o obtido) também chamado de arredondamento.

Erros iniciais são os cometidos no arredondamento dos dados iniciais;

Erros intermediários são decorrentes dos erros cometidos durante a aplicação do método numérico;

Erros finais decorrentes da apresentação final do resultado.

Os tipos de arredondamentos mais conhecidos são:

- Arredondamento para baixo ou por falta;
- Arredondamento para cima ou por excesso;
- Arredondamento para o número de máquina mais próximo.

Critério de Arredondamento: no cálculo manual, ao registrar um valor aproximado, costuma-se usar a seguinte regra:

1. somar meia unidade após a última casa decimal a conservar;
2. desprezar as demais casas.

Exemplo

$$\begin{aligned}\sqrt{2} &= 1.414 \dots \cong 1.41 \\ \sqrt[3]{2} &= 1.259 \dots \cong 1.26\end{aligned}$$

10.3 Erro de truncamento

São erros provenientes da utilização de processos que deveriam ser infinitos ou muito grandes para a determinação de um valor e que, por razões práticas, são truncados.

Estes processos infinitos são muito utilizados na avaliação de funções matemáticas, tais como, exponenciação, logaritmos, funções trigonométricas e várias outras que uma máquina pode ter.

10.3.1 A série de Taylor

Em matemática, uma série de Taylor é a série de funções da forma:

$$f(x) = \sum_{n=0}^{\infty} a_n (x - a)^n \quad \text{sendo} \quad a_n = \frac{f^{(n)}(a)}{n!}$$

onde $f(x)$ é uma função analítica dada. Neste caso, a série acima é dita ser a série de Taylor de $f(x)$ em torno do ponto $x = a$.

Associadamente, o polinômio de Taylor de ordem n em torno de $x = a$ de uma dada função n -vezes diferenciável neste ponto é dado por:

$$p(x) = f(a) \frac{(x-a)^0}{0!} + f'(a) \frac{(x-a)^1}{1!} + f''(a) \frac{(x-a)^2}{2!} + \dots + f^{(n)}(a) \frac{(x-a)^n}{n!}$$

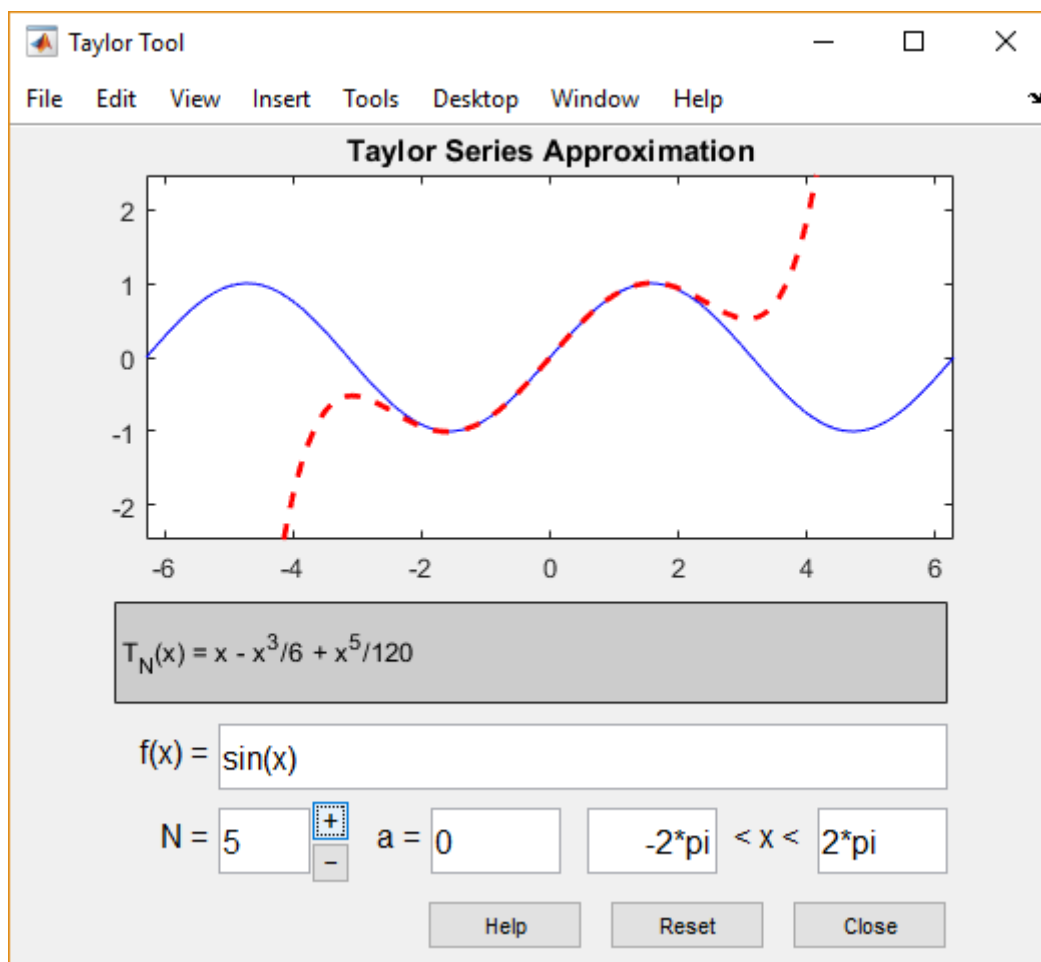
No caso particular de $a = 0$, série acima também é chamada de Série de Maclaurin ou, quando for o caso, de polinômio de Maclaurin.

Tais séries recebem seu nome em homenagem a Brook Taylor que as estudou no trabalho *Methodus incrementorum directa et inversa* em 1715. Condorcet atribuiu estas séries a Taylor e d'Alembert. O nome série de Taylor só começou a ser usado em 1786, por l'Huillier.

Analisando a série de Taylor no MATLAB

Listing 35: Série de Taylor

```
>> taylorTool('sin(x)')
```



Em uma série de Taylor $S(x) = \sum_{n=1}^{\infty} a_n x^n$, o erro de truncamento de ordem N em ponto x , $R_N(x)$ é definido como a diferença entre o valor exato de $S(x)$ e a soma dos N primeiros termos da série:

$$R_N(x) = \sum_{n=N+1}^{\infty} a_n x^n$$

Exemplo

Use a expansão em série de Taylor com ordem variando de 0...6 para aproximar $f(x) = \cos(x)$ em $x_{i+1} = \frac{\pi}{3}$ com base no valor de $f(x)$ e suas derivadas em $x_i = \frac{\pi}{4}$ código de solução no MATLAB:

Listing 36: Programa MATLAB solução

```

1 clear
2 clc
3 syms x;
4 f=cos(x);
5 n = input('Entre com o numero de termos (n): ');
6 x1 = pi/3;
7 vl_computado = vpa(subs(f,x1));
8 Tf=0;
9 fprintf('_____ \n');
10 fprintf('n      fn(x)      f(pi/3)      |er| \n');
11 fprintf('_____ \n');
12 for i = 0:n
13     an=vpa(subs(f,0))/factorial(i);
14     Tf=Tf+an*(x1-0)^i;
15     ea=abs(vl_computado-Tf);
16     er=(ea/Tf)*100;
17     fprintf('%d\t%s\t%6.6f\t%8.4f%%\t\n',i,f, Tf,er);
18     f=diff(f);
19 end

```

Terá então como saída:

Command Window			
n	fn(x)	f(pi/3)	er
0	cos(x)	1.000000	50.0000%
1	-sin(x)	1.000000	50.0000%
2	-cos(x)	0.451689	10.6957%
3	sin(x)	0.451689	10.6957%
4	cos(x)	0.501796	0.3580%
5	-sin(x)	0.501796	0.3580%
6	-cos(x)	0.499965	0.0071%

11**Exercícios**

Lista de série de Taylor para funções mais comuns:

a) $e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}$ para todo x

$$b) \ln(1+x) = \sum_{n=0}^{\infty} \frac{(-1)^n}{n+1} x^{n+1} \quad \text{para } |x| < 1$$

$$c) \frac{x^m}{1-x} = \sum_{n=m}^{\infty} x^n \quad \text{para } |x| < 1$$

$$d) \cos x = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n)!} x^{2n} \quad \text{para todo } x$$

$$e) \sin x = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1} \quad \text{para todo } x$$

Exercício 1.

Usando a série de Taylor, determine o valor aproximado do erro, utilizando 4 dígitos significativos, para o cálculo do $\sin(47^\circ)$, em torno do ponto $a = \frac{\pi}{4}$, com polinômios de grau:

1. 2;

2. 3;

Exercício 2.

Considere a série Harmônica dada por $S = \sum_{n=1}^{\infty} \frac{1}{n}$. Mostra-se que $S > 1 + \frac{1}{2} + \frac{1}{3} + \dots$ e portanto é divergente. No entanto, se calcularmos S , usando o algoritmo :

$$S_1 = 1$$

$$S_{k+1} = S_k + \frac{1}{k+1}, \quad k \geq 1, \text{ obtemos um resultado finito. Explique o que ocorre.}$$

Exercício 3.

A série infinita

$$\sum_{i=1}^n \frac{1}{i^4}$$

converge para $f(n) = \frac{\pi^4}{90}$ quando n tende para infinito. Escreva um programa com precisão variada e para calcular o valor de $f(n)$ quando $n = 10.000$ computando a soma de $i = 1$ até 10.000. Em seguida repita o procedimento em ordem decrescente, ou seja com $i = 10.000$ até 1. Analise os resultados em ambas as tomadas de valor, compare com algumas precisões diferentes e explique os resultados.

Exercício 4.

Determine o valor de e^{-5} usando duas abordagens:

$$a) e^{-x} = 1 - x + \frac{x^2}{2} - \frac{x^3}{3!} + \dots$$

$$b) e^{-x} = \frac{1}{e^x} = \frac{1}{1 - x + \frac{x^2}{2} - \frac{x^3}{3!} + \dots}$$

Tomando como verdadeiro o valor 6.737947×10^{-3} . Use 20 termos na avaliação da série e compare com o valor verdadeiro e compute o erro associado.

Exercício 5.

A derivada de $f(x) = \frac{1}{(1-3x^2)}$ é dada por $f'(x) = \frac{6x}{(1-3x^2)^2}$. Qual o valor da função para $x = 0.577$? Tente usando 3 e 4 dígitos e compare os resultados.

Exercício 6.

Repita o processo com as seguintes funções:

a) $f(x) = x^3 - 7x^2 + 8x - 0.35$ com $x = 1.37$ com 3 e 4 dígitos.

b) $f(x) = ((x-7)x+8)x - 0.35$ com $x = 1.37$ com 3 e 4 dígitos.

Exercício 7.

Escreva um programa que determine o número de termos necessário para aproximarmos o valor do $\cos x$ com 8 termos significativos usando um a série de Maclaurin.

$$\cos x = 1 - \frac{x^2}{2} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!}$$

e calcule está aproximação considerando que $x = 0.3\pi$

Exercício 8.

Calcule o valor da expressão $\sqrt{\cos(\frac{\pi}{2} - x)}$ para $x = 0.49 \times 10^{-4}$ em um sistema de ponto flutuante com 6 dígitos de precisão. Agora leve em consideração que para x pequeno e positivo $\cos(\frac{\pi}{2} - x) \approx x$ e recalcule o valor da expressão. Se assumirmos que a segunda forma é a mais acurada, qual o erro relativo cometido no cálculo da primeira expressão?

12 Zeros de Equações Transcendentes e Polinomiais

Seja $F(x)$ uma função real definida num intervalo $[a, b]$. Chama-se raiz(es) desta função em $[a, b]$ a $\forall \xi \in (a, b)$ tal que $F(\xi) = 0$, como mostra a figura 5.

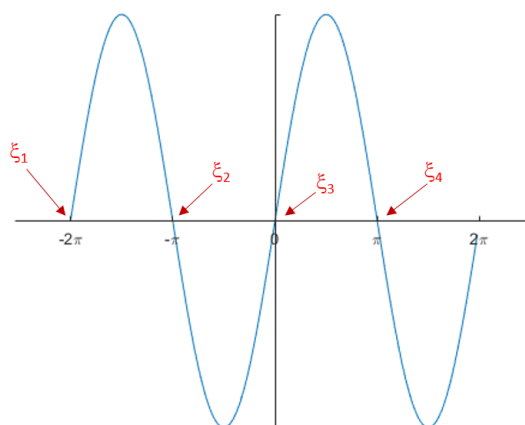


Figura 5: Zeros da função num intervalo dado

12.1 Tipos de Métodos

Pode-se dizer que são dois os métodos para se achar a(s) raiz(es) de uma equação:

Método direto: quando fornece solução em apenas um único passo. Esta raiz é exata e a menos de erros de arredondamento.

Exemplo

Seja $F(x) = x^2 - 3x + 2$. A solução direta pode ser obtida através da fórmula:
$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a},$$
 que terá como conjunto solução $\{1, 2\}$.

Método iterativo ou indireto: é um processo de cálculo infinito, recursivo, em que o valor obtido a cada passo depende de valores obtidos em passos anteriores. Este tipo de método, na maioria das vezes, não obtém solução exata para as raízes, mas sim uma solução aproximada dentro de uma faixa de erro considerada aceitável.

É importante salientar, que normalmente, os métodos iterativos são mais precisos quando executados em um computador que permite agilizar os cálculos matemáticos, obtendo assim uma melhor precisão.

Mesmo um polinômio de grau maior que três já não tem uma solução algébrica simples como a da equação do segundo grau, a não ser em casos particulares. Vamos analisar como enfrentar esse problema, tão comum em diversas áreas da engenharia, da economia, das ciências, da física, entre tantas outras.

Para se calcular uma raiz duas etapas devem ser seguidas:

- 1ª) isolar a raiz, ou seja, encontrar um intervalo $[a, b]$, o menor possível, que contenha uma e somente uma raiz da equação $f(\xi) = 0$
- 2ª) Melhorar o valor da raiz aproximada, isto é, refina-lá até o grau de exatidão requerido. Com a abordagem iterativa precisamos determinar um intervalo inicial para construirmos a sequência $\{x_i\}$ e teremos que a raiz x' será dada por:

$$\lim_{i \rightarrow \infty} x_i$$

Além disto, temos que estipular critérios de parada, pois na prática não calcularemos infinitos termos, mas apenas o suficiente para atingirmos a exatidão desejada.

12.2 Isolamento das Raízes

Nesta fase é feita uma análise teórica e gráfica da função $f(x)$. Para tal fim, usa-se frequentemente um importante teorema da álgebra.

Teorema 1

Se uma função $f(x)$ contínua num intervalo $[a, b]$ assume valores de sinais opostos nos pontos extremos deste intervalo, isto é, $f(a) \cdot f(b) < 0$, então o intervalo conterá, no mínimo, uma raiz da equação $f(x) = 0$; em outras palavras haverá, no mínimo, um número $\xi \in (a, b)$ tal que $f(\xi) = 0$.

12.2.1 Número de Raízes Reais

Vimos como delimitar as raízes reais de $F(x) = 0$. Agora iremos verificar quantas raízes existem no intervalo delimitado. Os métodos a seguir dão uma boa indicação sobre o número de raízes do intervalo.

Teorema 2: Teorema de Bolzano

Seja $F(x) = 0$ uma equação algébrica com coeficientes reais e $x \in (a, b)$

Se $F(a).F(b) < 0$, então existe um número **ímpar** de raízes (contando suas multiplicidades) no intervalo (a, b) .

Se $F(a).F(b) > 0$, então existe um número **par** de raízes reais (contando suas multiplicidades) ou **não existe** raízes reais no intervalo (a, b)

Um programa pode ser desenvolvido no MATLAB para implementar uma **busca incremental** a fim de localizar as raízes de uma função dentro de um intervalo.

Listing 37: Programa MATLAB solução

```

1 function xb=busca_inc(func,xmin,xmax,ns)
2 %Programa para localizacao de raizes de uma funcao
3 %Entrada:
4 %      func -> nome de uma funcao
5 %      xmin,xmax -> intervalo de busca
6 %      ns -> numero de subintervalos
7 if nargin<3, error('Sao neccessarios 3 argumentos de entrada'), end
8 if nargin<4, ns=100; end
9 x=linspace(xmin,xmax,ns);
10 f=func(x);
11 nb=0;
12 xb=[];
13 for k=1:length(x)-1
14     if sign(f(k))~=sign(f(k+1))
15         nb=nb+1;
16         xb(nb,1)=x(k);
17         xb(nb,2)=x(k+1);
18     end
19 end
20 if isempty(xb)
21     disp('Nenhum subintervalo obtido')
22     disp('Verifique o intervalo ou aumente ns')
23 else
24     fplot(func,[xmin xmax])
25     ax=gca;
26     ax.XAxisLocation='origin';
27     ax.YAxisLocation='left';
28     ax.Box='off';
29     hold on
30     for i=1:length(xb)

```

```

31     plot([xb(i,1) xb(i,2)],[0 0], 'marker', 'square')
32     end
33     xticks(xmin:0.3:xmax);
34     fprintf('numero de subintervalos:%d',length(nb));
35     disp(xb)
36 end

```

O exemplo anterior ilustra que métodos do tipo *força bruta*, como a busca incremental, não são infalíveis, por isso seria sensato complementar tais técnicas automáticas com outras informações que auxiliem na localização das raízes. Essas informações podem ser encontradas com o uso de gráficos e a compreensão do problema físico do qual a equação se originou.

12.2.2 Refinamento

Depois de isolar a raiz no intervalo $[a, b]$, passa-se a calculá-la através de métodos numéricos. Como veremos adiante, estes métodos devem fornecer uma sequência x_i de aproximação, cujo limite é a raiz exata ξ . Em cada aproximação x_n , da raiz exata ξ , usa-se um destes critérios e compara-se o resultado com a tolerância ε pré-fixada.

A verificação, de que x_n está “suficientemente” próxima da raiz, pode ser feita de dois modos diferentes (que podem levar a resultados diferentes):

$$|f(x_n)| \leq \varepsilon \quad \text{Em relação ao eixo } y$$

$$|x_n - x_{n-1}| \leq \varepsilon \quad \text{Em relação ao eixo } x$$

Observa-se que dependendo dos números envolvidos é aconselhável usar os testes de erro relativo:

$$\frac{|x_n - x_{n-1}|}{|x_{n-1}|} \leq \varepsilon$$

12.3 Método da Bisseção

Seja $f(x)$ uma função contínua no intervalo $[a, b]$ e seja ξ uma raiz desta função, sendo que $\xi \in (a, b)$, tal que $f(\xi) = 0$.

Dividindo o intervalo $[a, b]$ ao meio, obtém-se x_1 , havendo, pois, dois subintervalos, $[a, x_1]$ e $[x_1, b]$, a ser considerados. Se $f(x_1) = 0$, então $\xi = x_1$; caso contrário, a raiz estará no subintervalo onde a função tem sinais opostos nos pontos extremos, ou seja, se $f(a) \cdot f(x_1) < 0$ então $\xi \in [a, x_1]$, senão $f(a) \cdot f(x_1) > 0$ e $\xi \in [x_1, b]$.

O processo se repete até que se obtenha uma aproximação para a raiz exata ξ , ou seja,

que o critério de parada seja satisfeito. Então, por indução, temos:

Algoritmo 1: ALGORITMO PARA BISSEÇÃO

Entrada: Função f , a , b , tolerância ' es ', máximo de iterações $maxit$

Saída: Raiz de $f(x) = 0$, ea - Erro relativo, iterações

```

1 início
2   N ← 1
3   while N ≤ nmax do
4     c ← (a + b) / 2
5     if f(c) = 0 or (b - a) / 2 < es then
6       return c
7     end
8     N ← N + 1
9     if f(c)f(a) < 0 then
10      a ← c
11    else
12      b ← c
13    end
14  end
15 fim
  
```

A implementação deste algoritmo no MATLAB fica:

Listing 38: Rotina para calculo da raiz pela bisseção

```

1 function bissec(func,a,b,tol,maxit)
2 %Programa criado por Prof Marcos Figueredo
3 %data: 01/02/2018
4 %entrada
5 %func -> funcao objetivo
6 %a,b -> extremos que contenham uma raiz
7 %tol -> tolerancia maxima, caso nao seja informado sera tol=0.0001
8 %maxit -> maximo de iteracoes, caso nao seja informado sera 50
9 %Exemplo:bissec(@(x) x^3-x-2,1,2)
10
11 if nargin<3
12     error('Sao necessarios pelo menos 3 argumentos de entrada!'),end
13 test=func(a)*func(b);
14 if test>0,error('nao existem solucoes no intervalo!'),end
15 if nargin<4||isempty(tol),tol=0.0001;end
16 if nargin<5||isempty(maxit), maxit=50;end
17 i=1;
18 x=0;
19 fprintf('_____ \n');
20 fprintf('%2s%6s%12s%13s%13s\n','N','a','b','xr','|er|');
21 fprintf('_____ \n');
22 while(abs(a-b)>tol)
23     if maxit==i,break,end
24     if(i==1)
  
```



```

25         x_old=a;
26     else
27         x_old=x;
28     end
29     x=(a+b)/2;
30     er=abs((x_old-x)/x)*100;
31     fprintf('%d\t%6.6f\t%6.6f\t%6.6f\t%8.4f%%\t\n',i,a,b,x,er);
32     if func(x)*func(a)<0
33         b=x;
34     else
35         a=x;
36     end
37     i=i+1;
38 end

```

Considerando uma precisão ε e um intervalo inicial $[a, b]$ é possível saber, a priori, quantas iterações serão efetuadas pelo método da bisseção até que se obtenha $|b - a| \leq \varepsilon$, usando o algoritmo deste método. Vimos que

$$b_k - a_k = \frac{b_{k-1} - a_{k-1}}{2} = \frac{b_0 - a_0}{2^k}$$

Deve-se obter o valor de k tal que $b_k - a_k < \varepsilon$, ou seja:

$$\frac{b_0 - a_0}{2^k} < \varepsilon \Rightarrow \frac{b_0 - a_0}{\varepsilon} < 2^k$$

$$\log(b_0 - a_0) - \log(\varepsilon) < k * \log(2)$$

$$\frac{\log(b_0 - a_0) - \log(\varepsilon)}{\log(2)} < k$$

Portanto, se k satisfaz a relação acima, ao final da iteração k teremos o intervalo $[a, b]$ que contem a raiz ξ .

12.4 Método da Falsa Posição

Seja $f(x)$ uma função contínua no intervalo $[a, b]$ e seja ξ uma raiz desta função, sendo que $\xi \in (a, b)$, tal que $f(\xi) = 0$.

No caso do Método da Bisseção, x_n é obtido através da média aritmética entre os extremos a e b :

$$x_n = \frac{a + b}{2}$$

Na maioria das vezes a raiz está mais próxima de um dos extremos do intervalo. Se partirmos do princípio de que a raiz deve estar mais próxima do ponto que apresenta o menor valor da função, então, em vez de tomar a média aritmética entre a e b , o método da falsa posição toma a média aritmética ponderada entre a e b com pesos $|f(b)|$ e $|f(a)|$, respectivamente:

$$x_n = \frac{a|f(b)| + b|f(a)|}{|f(b)| + |f(a)|}$$

, visto que $|f(b)|$ e $|f(a)|$ tem sinais opostos, temos então:

$$x_n = \frac{af(b) - bf(a)}{f(b) - f(a)} = \frac{af(b) - bf(a) - af(a) + af(a)}{f(b) - f(a)} = a - \frac{(b-a)f(a)}{f(b) - f(a)}$$

Para $n = 1, 2, 3, \dots$

Algoritmo 2: ALGORITMO PARA FALSA POSIÇÃO

Entrada: Função f , a , b , tolerância ' ϵ ', máximo de iterações maxit

Saída: Raiz de $f(x) = 0$, ea - Erro relativo, iterações

```

1 início
2   if  $f(a) * f(b) < 0$  then
3     while  $|f(x)| > \epsilon$  E  $|b - a| > \epsilon$  do
4        $x \leftarrow \frac{af(b) - bf(a)}{f(b) - f(a)}$ 
5       if  $f(x)f(a) < 0$  then
6          $a \leftarrow x$ 
7       else
8          $b \leftarrow x$ 
9       end
10    end
11  end
12 fim
  
```

Listing 39: Rotina para calculo da raiz pela bisseção

```

1 function falsaPosicao(func,a,b,tol,maxit)
2 %Programa criado por Prof Marcos Figueredo
3 %data: 01/02/2018
4 %entrada
5 %func -> funcao objetivo
6 %a,b -> extremos que contenham uma raiz
7 %tol -> tolerancia maxima, caso nao seja informado sera tol=0.0001
8 %maxit -> maximo de iteracoes, caso nao seja informado sera 50
9 %Exemplo:bissec(@(x) x^3-x-2,1,2)
10
11 if nargin<3
12     error('Sao necessarios pelo menos 3 argumentos de entrada!'),end
13 test=func(a)*func(b);
14 if test>0,error('nao existem solucoes no intervalo!'),end
15 if nargin<4||isempty(tol),tol=0.0001;end
16 if nargin<5||isempty(maxit), maxit=50;end
17 i=1;
18 x=0;
19 clc;
20 fprintf('\t=====\\n');
21 fprintf('\t\tMetodo da Falsa Posicao\\n');
22 fprintf('\t=====\\n\\n\\n');
23 fprintf('_____|\\n');
24 fprintf('%2s%6s%12s%13s%13s\\n','N','a','b','xr','|er|');
25 fprintf('_____|\\n');
26 while(abs(func(x))>tol)
  
```

```

27     if maxit==i,break,end
28     if(i==1)
29         x_old=a;
30     else
31         x_old=x;
32     end
33     x=(a*func(b)-b*func(a))/(func(b)-func(a));
34     er=abs((x_old-x)/x)*100;
35     fprintf('%d\t%.6f\t%.6f\t%.6f\t%.8f%%\t\n',i,a,b,x,er);
36     if func(x)*func(a)<0
37         b=x;
38     else
39         a=x;
40     end
41     i=i+1;
42 end

```

12.5 Exercícios

Exercício 1.

Determine as raízes de $f(x) = -12 - 21x + 18x^2 - 2.7x^3$ graficamente. Além disso, determine a primeira raiz da função com a bisseção e o método da falsa posição.

Exercício 2.

Localiza a primeira raiz não trivial de $\sin(x) = x^3$, onde está x em radianos? Use a técnica gráfica, bisseção e Falsa posição co o intervalo inicial de $[0.5, 1]$.

Exercício 3.

Seja uma viga uniforme sujeita a uma carga distribuída de forma linearmente crescente. A equação para a curva elástica resultante é:

$$y = \frac{\omega_0}{120EIL} (-x^5 + 2L^2x^3 - L^4x)$$

Use a bisseção (e falsa posição) para determinar o pondo de deflexão máxima(isto é, onde a derivada é igual a zero). A seguir substitua esse valor da equação para determinar o valor da deflexão máxima. Use os seguintes valores dos parâmetros nos seus cálculos: $L = 600\text{cm}$, $E = 50.000\text{kN/cm}^2$, $I = 30.000\text{cm}^4$ e $\omega_0 = 2.5\text{kN/cm}$.

Exercício 4.

Determine a(s) raiz(es) da funções, usando o método da Bisseção, e da Falsa posição com tolerância $\varepsilon = 1.10^{-3}$. Quantas iterações foram necessárias para cada um dos métodos?

(a) $f_1(x) = \sqrt{x} - e^{-x}$

(b) $f_2(x) = \ln(x) - x + 2$

(c) $f_3(x) = e^{x/2} - x^3$

(d) $f_4(x) = \sin(x) - x^2$

(e) $f_5(x) = \frac{x}{4} - \cos(x)$

Exercício 5.

Uma carga total Q está uniformemente distribuída ao redor de um condutor circular de raio a . Uma carga q está localizada a uma distância x do centro do anel. A força exercida na carga pelo anel é dada por:

$$F = \frac{1}{4\pi\epsilon_0} \frac{qQx}{(x^2 + a^2)^{3/2}}$$

onde $\epsilon_0 = 8.9 \times 10^{-12} C^2/(Nm^2)$. Encontre a distância x onde a força é $1.25N$ se q e Q são 2×10^{-5} para um anel de raio $a = 0.85m$

12.6 Método de Newton-Raphson ou Método das Tangentes

Este método, sob determinadas condições, apresenta vantagens sobre os métodos anteriores: é de convergência rápida e, para encontrar as raízes, não é obrigatória a condição $f(a) * f(b) < 0$. Observe o gráfico da figura 6

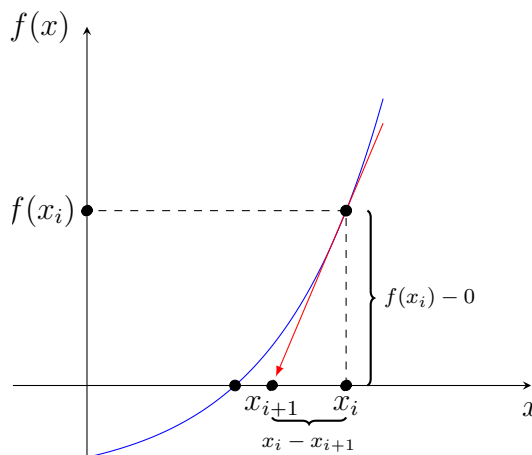


Figura 6: Descrição gráfica do método de Newton-Raphson. A tangente à função em x_i é prolongada até o eixo x para fornecer uma estimativa da raiz em x_{i+1}

O método de Newton-Raphson pode ser deduzido com base na sua interpretação geométrica. Como na figura 6, a primeira derivada em x é equivalente à inclinação:

$$f'(x_i) = \frac{f(x) - 0}{x_i - x_{i+1}}$$

que pode ser reorganizada para fornecer

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \quad (3)$$

que é chamada fórmula de Newton-Raphson.

Seu algoritmo básico é:

Algoritmo 3: ALGORITMO PARA NEWTON RAPHSON

Entrada: Função f , x_i , tolerância 'tol', máximo de iterações 'maxit'

Saída: Raiz de $f(x) = 0$, ea - Erro relativo, iterações

```

1 início
2    $k \leftarrow 1$ 
3   while  $k \leq \text{maxit}$  do
4        $x_{i+1} \leftarrow x_i - \frac{f(x_i)}{f'(x_i)}$ 
5       if  $|x_{i+1} - x_i| < \text{tol}$  ou  $|f(x_{i+1})| < \text{tol}$  then
6           return  $x_{i+1}$  como solução e pare.
7       end
8        $k \leftarrow k + 1$ 
9        $x_i \leftarrow x_{i+1}$ 
10  end
11  return método falhou apos 'maxit' iterações. Pare
12 fim
  
```

Listing 40: Rotina para cálculo pelo método de Newton

```

1 function newtonRap(func,a,tol,maxit)
2 %Programa criado por Prof Marcos Figueredo
3 %data: 01/02/2018
4 %entrada
5 %func -> funcao objetivo
6 %a -> chute inicial
7 %tol -> tolerancia maxima, caso nao seja informado sera tol=0.0001
8 %maxit -> maximo de iteracoes, caso nao seja informado sera 50
9 %Exemplo:bissec(@(x) x^3-x-2,1,2)
10
11 if nargin<2
12     error('Sao necessarios pelo menos 2 argumentos de entrada!'),end
13 if nargin<3||isempty(tol),tol=0.0001;end
14 if nargin<4||isempty(maxit), maxit=50;end
15 i=1;
16 xr=0;
17 syms x;
18 f1=matlabFunction(diff(func(x)));
19 clc;
20 fprintf('\t=====\\n');
21 fprintf('\t\tMetodo de Newton Raphson\\n');
22 fprintf('\t=====\\n\\n');
23 fprintf('_____\\n');
24 fprintf('%2s%6s%12s%13s%13s\\n','N','xi','x(i+1)','|er|');
25 fprintf('\\n_____\\n');
26 while(abs(func(xr))>tol)
27     if maxit==i,break,end
28     xr=a-func(a)/f1(a);
29     er=abs((xr-a)/xr)*100;
  
```

```
30     fprintf('%d\t%.6f\t%.6f\t%.8f%%\t\n',i,a,xr,er);  
31     a=xr;  
32     i=i+1;  
33 end
```

12.7 Método da Secante

Uma grande desvantagem do método de Newton é a necessidade de se obter a derivada $f'(x)$ e calcular o seu valor numérico a cada iteração.

Para contornar este problema podemos substituir o cálculo da primeira derivada $f'(x_i)$ pelo quociente das diferenças, usando assim, um modelo linear baseado nos dois valores calculados mais recentemente:

$$f'(x_i) \cong \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i} \quad (4)$$

Essa equação (4) pode ser substituída na equação 3 para fornecer a seguinte equação iterativa:

$$x_{i+1} = x_i - \frac{f(x_i) \cdot (x_{i-1} - x_i)}{f(x_{i-1}) - f(x_i)} \quad (5)$$

A equação 5 é a fórmula para o método da secante. Observe que a abordagem exige duas estimativas iniciais para x . No entanto, como não é exigido que $f(x)$ mude de sinal entre as estimativas, ele não é classificado como um método intervalar.

Em vez de usar dois valores arbitrários para estimar a derivada, uma abordagem alternativa envolve uma pequena perturbação da variável independente para estimar $f'(x)$.

$$f'(x) \cong \frac{f(x_i + \delta x_i) - f(x_i)}{\delta x_i}$$

onde δx_i é uma pequena fração de perturbação. Essa aproximação pode ser substituída na Equação 3 para fornecer a seguinte equação iterativa:

$$x_{i+1} = x_i - \frac{\delta x_i \cdot f(x_i)}{f(x_i + \delta x_i) - f(x_i)} \quad (6)$$

Essa fórmula do chamado método das secantes modificado.

Seu algoritmo básico é:

Algoritmo 4: ALGORITMO DO MÉTODO DAS SECANTES

Entrada: Função f , x_{i-1} , x_i , tolerância 'tol', máximo de iterações 'maxit'

Saída: Raiz de $f(x) = 0$, ea - Erro relativo, iterações

```

1 início
2    $k \leftarrow 1$ 
3   while  $k \leq \text{maxit}$  do
4        $x_{i+1} \leftarrow x_i - \frac{f(x_i) \cdot (x_i - x_{i-1})}{f(x_i) - f(x_{i-1})}$ 
5       if  $|x_{i+1} - x_i| < \text{tol}$  ou  $|f(x_{i+1})| < \text{tol}$  then
6           return  $x_{i+1}$  como solução e pare.
7       end
8        $k \leftarrow k + 1$ 
9        $x_{i-1} \leftarrow x_i$ 
10       $x_i \leftarrow x_{i+1}$ 
11  end
12  return método falhou apos 'maxit' iterações. Pare
13 fim

```

Listing 41: Rotina MATLAB para cálculo pelo método da Secante

```

1 function secantes(func,a,b,tol,maxit)
2 %Programa criado por Prof Marcos Figueredo
3 %data: 01/02/2018
4 %entrada
5 %func -> funcao objetivo
6 %a -> chute inicial
7 %tol -> tolerancia maxima, caso nao seja informado sera tol=0.0001
8 %maxit -> maximo de iteracoes, caso nao seja informado sera 50
9 %Exemplo:bissec(@(x) x^3-x-2,1,2)
10
11 if nargin<3
12     error('Sao necessarios pelo menos 3 argumentos de entrada!'),end
13 if nargin<4||isempty(tol),tol=0.0001;end
14 if nargin<5||isempty(maxit), maxit=50;end
15 i=1;
16 xr=0;
17 clc;
18 fprintf('\t===== \n');
19 fprintf('\t\tMetodo da Secante \n');
20 fprintf('\t===== \n \n \n');
21 fprintf('_____ \n');
22 fprintf('%2s%10s%10s%12s%14s%13s \n', 'N', 'x(i-1)', 'xi', 'x(i+1)', '|er|');
23 fprintf('\n _____ \n');
24 while(abs(func(xr))>tol)
25     if maxit==i,break,end
26     xr=a-(func(a)*(a-b))/(func(a)-func(b));
27     er=abs((xr-a)/xr)*100;
28     fprintf('%d\t%6.6f\t%6.6f\t%6.6f\t%8.4f%% \n',i,a,b,xr,er);

```

```

29     b=a;
30     a=xr;
31     i=i+1;
32 end

```

12.8 Exercícios

Exercício 6.

Localize graficamente os zeros das funções a seguir:

1. $f(x) = 4 \cos(x) - e^{2x}$
2. $f(x) = \frac{x}{2} - \operatorname{tg}(x)$
3. $f(x) = 1 - x \ln(x)$
4. $f(x) = 2^x - 3x$
5. $f(x) = x^3 + x - 1000$

Determine suas raízes mais utilizando os métodos de Newton e Secante

Exercício 7.

Use o método Newton-Raphson para obter a menor raiz positiva das equações a seguir com precisão 10^{-2}

1. $\frac{x}{2} - \operatorname{tg}(x) = 0$
2. $2 \cos(x) - \frac{e^x}{2} = 0$
3. $x^5 - 6 = 0$

Exercício 8.

Determine a raízes reais da função $f(x) = -0.6x^2 + 2.4x + 5.5$:

- a) Graficamente
- b) Usando a formula de resolução conhecida (computacionalmente)
- c) Usando o método da Bisseção, Newton-Raphson e secante determine a maior raiz. utilize como chute inicial $x_1 = 5$ e $x_2 = 10$. Compute o erro relativo após cada iteração.

Exercício 9.

Determine a raízes reais da função $f(x) = 4x^3 - 6x^2 + 7x - 2.3$:

- a) Graficamente
- b) Usando o método da Bisseção, Newton-Raphson e secante determine a maior raiz. utilize como chute inicial $x_1 = 0$ e $x_2 = 1$. Compute até um erro relativo atingir 10%.

Exercício 10.

Determine a raízes reais da função $f(x) = -26 + 85x - 91x^2 + 44x^3 - 8x^4 + x^5$:

- a) Graficamente
- b) Usando o método da Bisseção, Newton-Raphson e secante determine uma raiz com chute inicial $x_1 = 0.5$ e $x_2 = 1.0$. Compute até um erro relativo atingir 10%.

Exercício 11.

Desenvolva uma função no MATLAB para o método da secante modificado. Com as aproximações e resolva os problemas do exercício 6. Qual a principal diferença notada entre os métodos?

13

Interpolação

Em matemática, denomina-se interpolação o método que permite construir um novo conjunto de dados a partir de um conjunto discreto de dados pontuais previamente conhecidos.

Em engenharia e ciência, dispõe-se habitualmente de dados pontuais obtidos a partir de uma amostragem ou de um experimento. Tal conjunto de dados pontuais (também denominado conjunto degenerado) não possui continuidade, e isto muitas vezes torna demasiado irreal a representação teórica de um fenômeno real empiricamente observado.

Através da interpolação, pode-se construir uma função que aproximadamente se "encaixe" nestes dados pontuais, conferindo-lhes, então, a continuidade desejada.

Outra aplicação da interpolação é a aproximação de funções complexas por funções mais simples. Suponha que tenhamos uma função, mas que seja complicada demais para que seja possível avaliá-la de forma eficiente. Podemos, então, escolher alguns dados pontuais da função complicada e tentar interpolá-los com uma função mais simples. Obviamente, quando utilizamos a função mais simples para calcular novos dados, normalmente não se obtém o mesmo resultado da função original, mas dependendo do domínio do problema e do método de interpolação utilizado, o ganho de simplicidade pode compensar o erro.

A interpolação permite fazer a reconstituição (aproximada) de uma função, bastando para tanto conhecer apenas algumas das suas abscissas e respectivas ordenadas (imagens no contra-domínio da função). A função resultante garantidamente passa pelos pontos fornecidos, e, em relação aos outros pontos, pode ser considerada um mero ajuste.

13.1 Polinômios

Em matemática, função polinomial é uma função \mathcal{P} que pode ser expressa da forma:

$$\mathcal{P}(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x^1 + a_0 x^0 = \sum_{i=0}^n a_i x^i$$

em que n é um número inteiro não negativo e os números $a_0, a_1, \dots, a_{n-1}, a_n$ são constantes, chamadas de coeficientes do polinômio.

As funções polinomiais podem ser classificadas quanto a seu grau. O grau de uma função polinomial corresponde ao valor do maior expoente da variável do polinômio, ou seja, é o valor de n da função $\mathcal{P}(x) = \sum_{i=0}^n a_i x^i$.

Sejam $f(x)$ e $g(x)$ polinômios de graus quaisquer. Sempre valem as seguintes leis:

- (a) O grau de $f(x) \cdot g(x)$ é a soma do grau de $f(x)$ e o grau de $g(x)$;
- (b) Se $f(x)$ e $g(x)$ têm grau diferente, então o grau de $f(x) + g(x)$ é igual ao maior dos dois;
- (c) Se $f(x)$ e $g(x)$ têm o mesmo grau, então o grau de $f(x) + g(x)$ é menor ou igual ao grau de $f(x)$.

13.2 Valores Numéricos de um Polinômio

Definição 1

Seja

$$\mathcal{P}(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x^1 + a_0 x^0$$

e $\alpha \in \mathbb{R}$ dizemos que o valor numérico de \mathcal{P} é simplesmente $\mathcal{P}(\alpha)$ ou seja:

$$\mathcal{P}(\alpha) = a_n \alpha^n + a_{n-1} \alpha^{n-1} + \cdots + a_1 \alpha^1 + a_0 \alpha^0$$

Exemplo

Para o polinômio $\mathcal{P}(x) = -x^2 + 3x - 1$, vamos obter seu valor numérico para $\alpha = 2$

Listing 42: janela comando Matlab recortada

```
>> %Defiindo o polinomio
>>p=[-1 3 -1];
>>% observe que definimos um vetor com os coeficiente do
    polinomio
>>a=2;
>>polyval(p,x)
>>%funcao do MATLAB que retorna o valor numerico do polinomio
>>ans = 1
```

Definição 2

Seja

$$\mathcal{P}(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x^1 + a_0 x^0$$

e $\alpha \in \mathbb{R}$ dizemos que α é raiz do um polinômio $\mathcal{P}(x)$ se $\mathcal{P}(\alpha) = 0$

Exemplo

Para o polinômio $\mathcal{P}(x) = -x^2 + 3x - 1$, vamos obter suas raízes.

Listing 43: janela comando Matlab recortada

```
>>p=[-1 3 -1];
>> roots(p) %funcao MATLAB que retorna o zero
>> ans =
    2.6180
    0.3820
```

13.3 Importando dados para o MATLAB

Considere um arquivo comum **txt**, contendo dados de alguma informação. Vamos carregar este conjunto de dados no MATLAB

Listing 44: janela comando Matlab recortada

```
>>A=importdata(filename); %filename e o arquivo contendo as informacoes  
>> %usando o comando disp podemos visualizar qualquer informacao destes  
dados
```

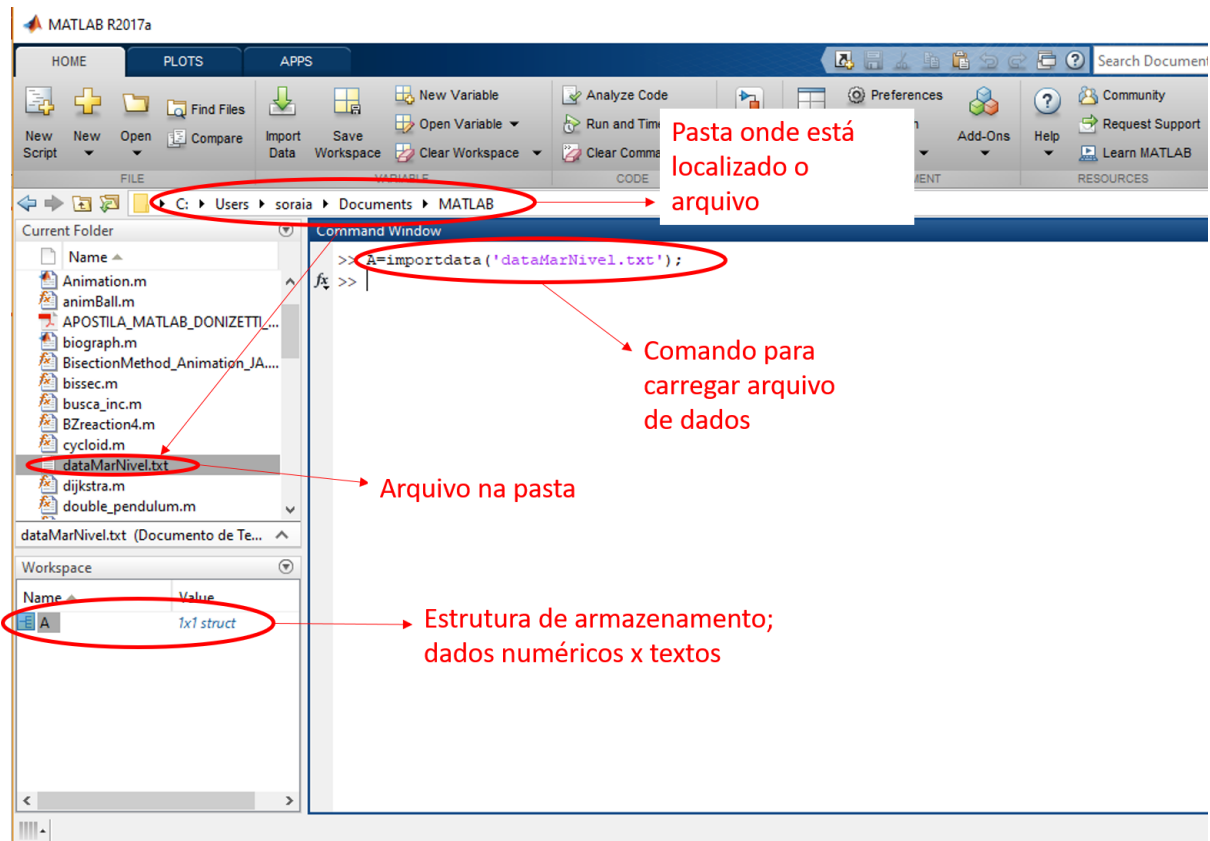


Figura 7: Exemplo de carregamento de dados no MATLAB

Agora que temos os dados carregados vamos separar as colunas de interesse:

Listing 45: janela comando Matlab recortada

```
>>x=A.data(:,1);  
>>y=A.data(:,2);  
>>%Para visualizar o conjunto dos dados çfaa:  
>>plot(x,y,'o');  
>>axis([1930 2030 24 30]);
```

13.4 Introdução

Este capítulo e o seguinte são dedicados a aproximação de funções, i.e., dado uma função f obter uma função g , em geral mais simples, que aproxima f segundo um certo critério. Ao enunciar um problema de aproximação consideram-se dois aspectos:

1. Definição da função a aproximar

- É conhecida uma relação que defina a função no seu domínio (expressão analítica, desenvolvimento em série, fórmula integral, equação diferencial ou integral, etc.)

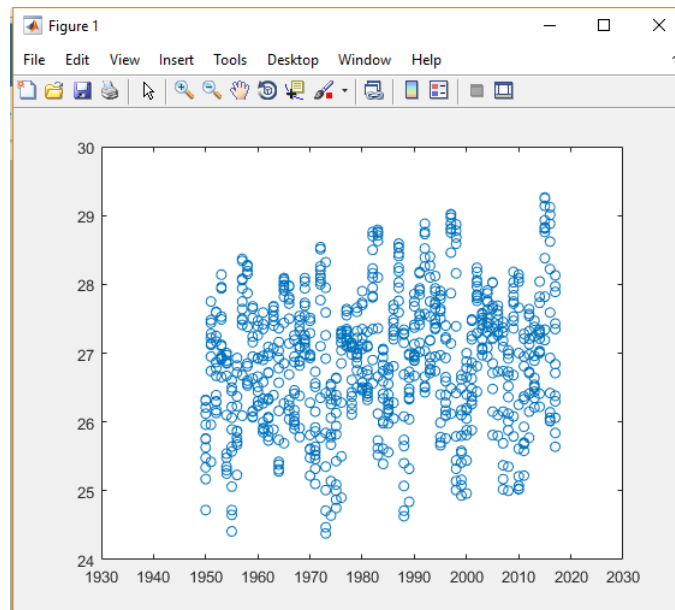


Figura 8: Imagem do arquivo de dados no MATLAB

- (b) São só conhecidos valores da função, habitualmente afetados de erros, num conjunto numerável de pontos.
2. Finalidade da aproximação:
Resolução de equações não lineares, diferenciais e integrais; derivação ou integração da função; cálculo, traçado do gráfico ou obtenção de uma expressão analítica da função, etc.
- Na resolução de um dado problema de aproximação consideram-se ainda mais dois aspectos.
3. Escolha da classe de funções aproximadoras:
Polinômios de um determinado grau, splines, funções trigonométricas, racionais (quocientes de polinômios) ou exponenciais, etc.
4. Escolha do critério de aproximação:
Interpolação ou minimização de uma norma ou semi-norma do erro (habitualmente as normas do máximo, Euclidiana e L_1).

Iremos estudar a interpolação polinomial. A interpolação consiste em escolher dentro de uma classe de funções a função g (interpoladora) cujo gráfico, definido por $y = g(x)$, passa por um dado conjunto de pontos (x_i, f_i) , $i = 0, 1, \dots, n$. O critério de aproximação consiste, portanto, em impor $g(x_i) = f(x_i)$, $i = 0, 1, \dots, n$.

Devido às suas propriedades algébricas e analíticas, os polinômios são as funções mais frequentemente utilizadas na aproximação e em particular na interpolação. A interpolação polinomial é utilizada preferencialmente na resolução de equações não lineares, diferenciais e integrais, na derivação e integração numérica e na interpolação de tabelas; constituindo a base de muitos métodos numéricos.

Dado que a interpolação polinomial fornece, frequentemente, uma aproximação grosseira da função, tem vindo a ser substituída pela interpolação por *spline* e pela interpolação racional. O interesse pela interpolação por spline tem vindo a crescer nos últimos anos.

É utilizada, por exemplo, no traçado de gráficos, na derivação e, tal como a interpolação racional, no cálculo computacional de funções.

Como já referimos, a interpolação tem várias aplicações importantes. A mais antiga consiste em interpolar uma tabela, i.e., definida uma função $f(x)$ por meio de uma tabela, obter um valor aproximado de $f(x)$ correspondente a um argumento x não tabelado ($x \neq x_i, \quad i = 0, 1, \dots, n$).

13.5 Determinação dos coeficientes de um polinômio

Considere a tabela 15:

$T(^{\circ}C)$	-40	0	20	50	100	150	200	250	300	400	500
$\rho(kg/m^3)$	1.52	1.29	1.20	1.09	0.946	0.835	0.746	0.675	0.616	0.525	0.457

Tabela 15: Densidade (ρ) como uma função da temperatura T em 1atm

cujo gráfico pode ser representado por

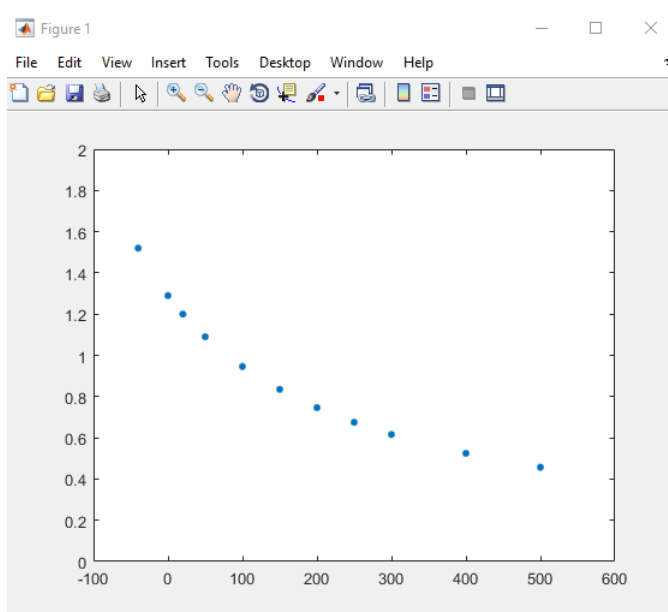


Figura 9: Imagem do arquivo de dados no MATLAB

Listing 46: Código MATLAB para gerar o gráfico

```

1 x=[-40 0 20 50 100 150 200 250 300 400 500];
2 y=[1.52 1.29 1.20 1.09 0.946 0.835 0.746 0.675 0.616 0.525 0.457];
3 plot(x,y,'.','MarkerSize', 14)
4 axis([-100 600 0 2]);

```

Uma maneira simples de calcular os coeficientes da equação 7 é baseada no fato que n pontos dados são necessários para determinar os n coeficientes.

$$\mathcal{P}(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x^1 + a_0 \quad (7)$$

Assim, o problema é reduzido à resolução de três equações algébricas lineares simultâneas para os três coeficientes desconhecidos. Usando o MATLAB temos:

Exemplo

Considere determinar os coeficientes da parábola

$$\mathcal{P}(x) = a_1x^2 + a_2x + a_0 \quad (8)$$

que passa através dos três últimos valores de densidade da tabela 15.

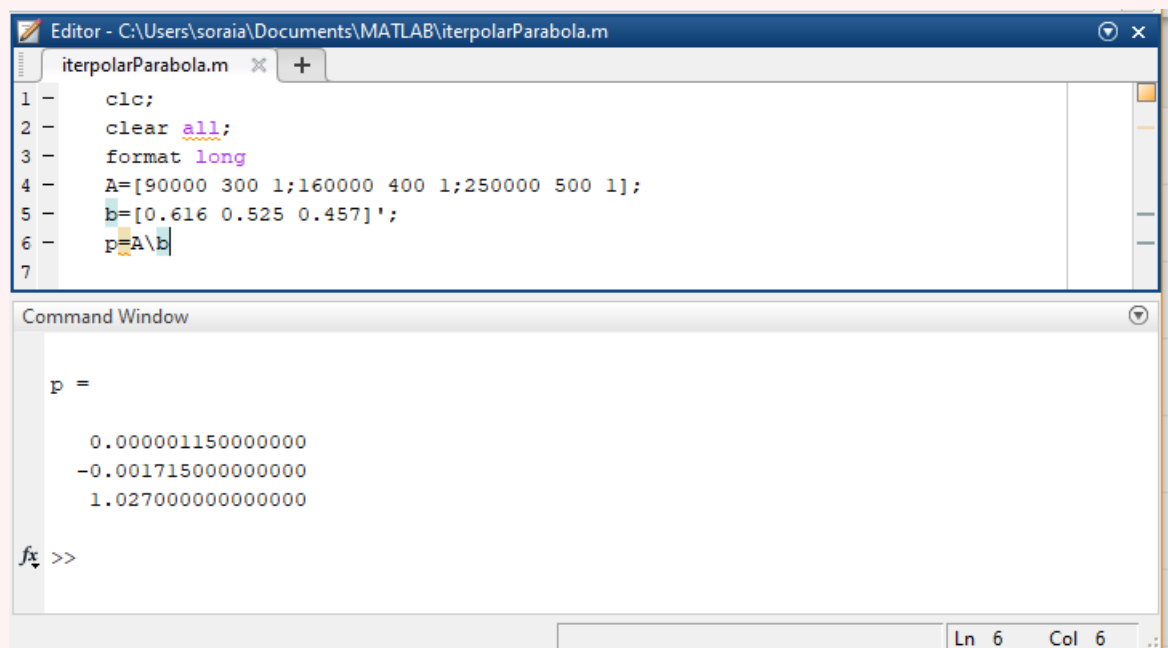
$$\begin{array}{rcl} x_1 & = & 300 \quad f(x_1) = 0.616 \\ x_2 & = & 400 \quad f(x_2) = 0.525 \\ x_3 & = & 500 \quad f(x_3) = 0.457 \end{array}$$

Cada um desses pares pode ser substituído na equação 8 para produzir um sistema de três equações:

$$\begin{cases} 0.616 = a_1(300)^2 + a_2(300) + a_0 \\ 0.525 = a_1(400)^2 + a_2(400) + a_0 \\ 0.457 = a_1(500)^2 + a_2(500) + a_0 \end{cases}$$

ou, na forma matricial

$$\begin{bmatrix} 90.000 & 300 & 1 \\ 160.000 & 400 & 1 \\ 250.000 & 500 & 1 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_0 \end{bmatrix} = \begin{bmatrix} 0.616 \\ 0.525 \\ 0.457 \end{bmatrix}$$



The screenshot shows a MATLAB script editor window titled 'Editor - C:\Users\soraia\Documents\MATLAB\iterpolarParabola.m'. The script contains the following code:

```
1 - clc;
2 - clear all;
3 - format long
4 - A=[90000 300 1;160000 400 1;250000 500 1];
5 - b=[0.616 0.525 0.457]';
6 - p=A\b;
7
```

Below the script editor is the Command Window, which displays the output of the script:

```
p =
    0.0000011500000000
   -0.0017150000000000
    1.0270000000000000
```

The Command Window also shows the prompt 'fx >>' and the status bar indicates 'Ln 6 Col 6'.

Portanto, a parábola que passa exatamente através dos três pontos é

$$\mathcal{P}(x) = 0.00000115x^2 - 0.001715x + 1.027$$

me particular

$$\mathcal{P}(350) = 0.567625$$

O problema anterior tem uma solução gráfica na qual observamos o comportamento da função obtida. A função *polyfit* pode ser utilizada para realizar uma regressão poli-

nomial. No caso em que o número de pontos dados é igual ao número de coeficientes, a função *polyfit* efetua a interpolação; ou seja, retorna os coeficientes do polinômio que passa diretamente através dos pontos dados. Pode-se, além disso, utilizar a função *polyval* para estimar o valor no ponto.

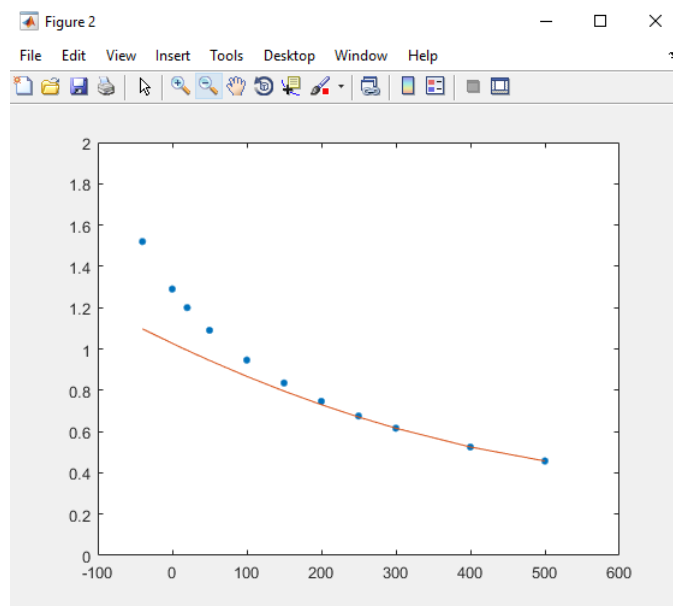


Figura 10: Gráfico dos pontos(azul) e sua aproximação por uma parábola(vermelho) MATLAB

Listing 47: Código MATLAB para gerar o gráfico

```

1 x=[-40 0 20 50 100 150 200 250 300 400 500];
2 y=[1.52 1.29 1.20 1.09 0.946 0.835 0.746 0.675 0.616 0.525 0.457];
3 T=[300 400 500];
4 d=[0.616 0.525 0.457]
5 p=polyfit(T,d,2);
6 y1=polyval(p,x);
7 figure
8 plot(x,y,'.','MarkerSize', 14)
9 axis([-100 600 0 2]);
10 hold on
11 plot(x,y1);
12 y2=polyval(p,350);
13 plot(350,y2,'.','MarkerSize',15,'color','green');
```

13.6 Polinômios interpoladores de Newton

13.6.1 Interpolação Linear

Dados dois pontos distintos de uma função $y = f(x) : (x_0, y_0)$ e (x_1, y_1) , deseja-se calcular o valor de y para um determinado valor de x entre x_0 e x_1 , usando a interpolação polinomial.

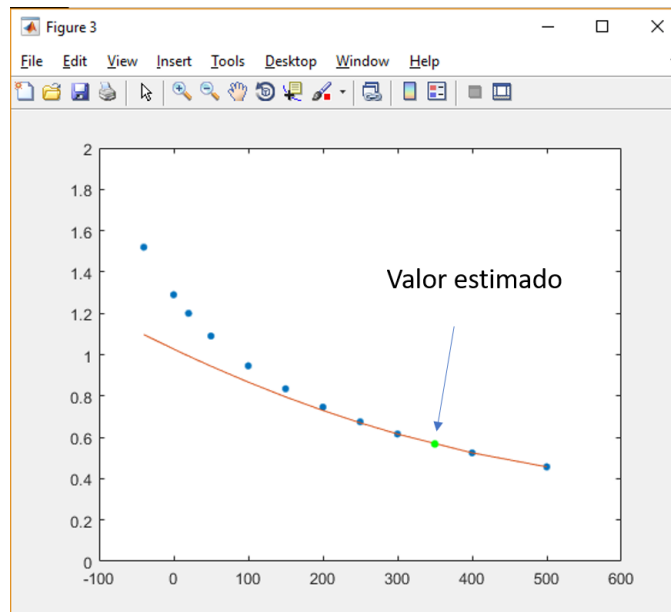


Figura 11: Valor estimado para $\mathcal{P}(350) = 0.567625$

O polinômio interpolador é uma unidade menor que o número de pontos conhecidos. Assim sendo, o polinômio interpolador nesse caso terá grau 1, isto é, $P_1(x) = a_1x + a_0$. Para determiná-lo, os coeficientes a_0 e a_1 devem ser calculados de forma que tenha:

$$\begin{cases} P_1(x_0) = f(x_0) = y_0 \\ P_1(x_1) = f(x_1) = y_1 \end{cases}$$

ou seja, basta resolver o sistema linear abaixo:

$$\begin{cases} a_1x_0 + a_0 = y_0 \\ a_1x_1 + a_0 = y_1 \end{cases}$$

onde a_1 e a_0 são as incógnitas e

$$A = \begin{bmatrix} x_0 & 1 \\ x_1 & 1 \end{bmatrix}$$

é a matriz dos coeficientes.

O determinante da matriz A é diferente de zero, sempre que $x_0 \neq x_1$, logo para pontos distintos o sistema tem solução única.

O polinômio interpolador $P_1(x) = a_1x + a_0$ tem como imagem geométrica uma reta, portanto estaremos aproximando a função $f(x)$ por uma reta que passa pelos dois pontos conhecidos (x_0, y_0) e (x_1, y_1) .