Introdução ao Mathematica

Éliton Fontana

19 de Abril de 2010

1 Uma Visão Geral Sobre o Programa

A primeira característica marcante sobre a estrutura do Mathematica é o fato de ele ser um programa do tipo CAS (*Computer Algebra System*). A grande vantagem desta classe de softwares é a possibilidade de operações simbólicas, ou seja, permite a manipulação de equações matemáticas expressas na forma de símbolos. Este tipo de estrutura é bastante diferente da aproximação numérica realizada pela maioria dos programas na área das engenharias. Outros exemplos de softwares CAS são o MathCad, Maple, Cadabra (para Linux) e SAGE (Open Source e com licença livre).

1.1 Comandos Básicos

Para resolver uma operação matemática simples, basta digitar as expressões e em seguida usar o comando *Shift + Enter*. Este comando é utilizando sempre que se deseja resolver um conjunto de valores de entrada. Quando feito isto, o programa irá atribuir uma nomeação *In* para os dados de entrada e uma nomeação *Out* para os de saída.

```
In[6]:= (3 + 1) ^ 2 + Sqrt[16] * Sin[Pi / 2]
Out[6]= 20
```

Cada entrada pode ter uma ou mais saídas correspondentes (dependendo do número de expressões utilizadas).

Pode-se também alocar um valor para uma determinada variável:

```
In[7]:=
a = 45
b = a - 10
c = A - 10
Out[7]= 45
Out[8]= 35
Out[9]= -10 + A
```

Percebe-se que o Mathematica é um software *Case Sensitive*, ou seja, existe uma diferenciação entre maiúsculas e minúsculas ($a \neq A$). Este detalhe é muito importante quando forem definidas as funções, pois isto costuma ser fonte de muitos erros.

Sempre que possível, o resultado será expresso em sua forma mais simples, o que fica bem evidente quando se opera com frações. Por exemplo:

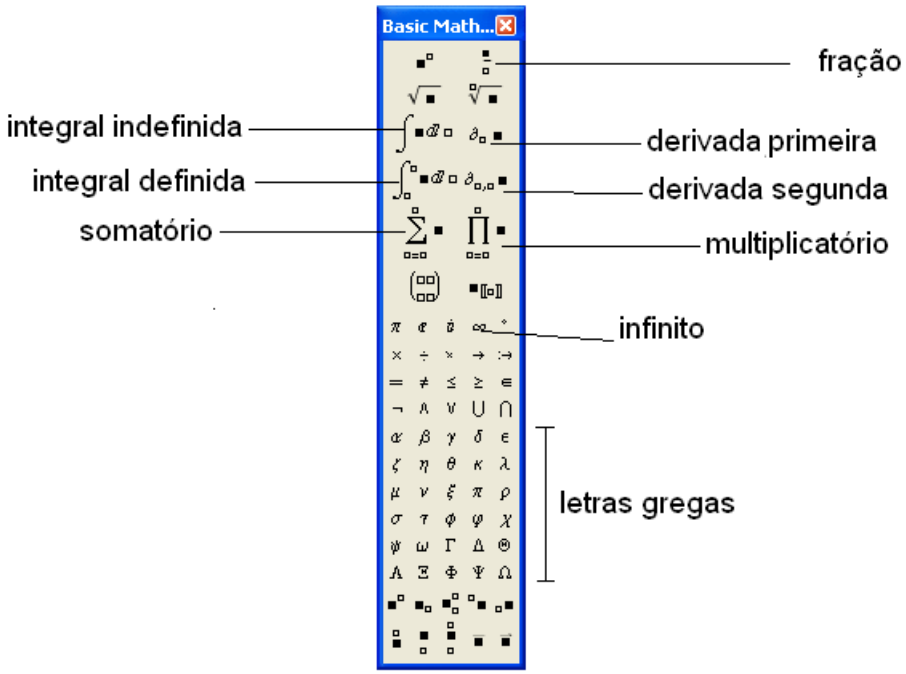
```
In[10]:=
21 / 28
Out[10]= 3/4
```

Caso queira-se obter o resultado numérico, basta "forçar" o programa para retornar neste formato, através do comando *N[ ]*:

```
In[11]:=
N[21 / 28]
Out[11]= 0.75
```

Dica de Sobrevivência: *Por padrão o Mathematica aloca o valor da última saída na variável genérica %. Porém, é sempre aconselhável dar nome às variáveis, pois isto pode evitar muitas confusões...*

Uma função muito útil do programa, especialmente quando não se tem muito conhecimento sobre os comandos básicos, são as *Palettes*, especialmente a *BasicMathInput*:



Exemplos de uso:

```
In[22]:=
        ∂x (x^2 + y^2)
Out[22]= 2 x

In[23]:=
        ∂x,x (x^2 + y^2)
Out[23]= 2

In[24]:=
        ∂x,y (x^2 + y^2)
Out[24]= 0
```

1.2 O Básico Sobre Comandos

Através de comandos específicos pode-se solicitar que o programa execute determinadas ações. Um comando normalmente possui a seguinte forma:

Comando1[{argumento1},{argumento2}]

onde o número de argumentos é variável, dependendo do comando.

Existem algumas regras que devem ser seguidas na entrada de comandos, dentre as quais pode-se destacar:

1. Os argumentos devem ser inseridos entre colchetes;

2. Caso houver mais de um argumento, estes devem ser separados por vírgula;

3. Todos os comandos começam com letra maiúscula. Comandos compostos por mais de uma palavra devem possuir letra maiúscula no começo de cada palavra, como por exemplo: *ExpToTrig, ArcSin, NSolve, etc.*

Um detalhe importante a ser observado é quanto à coloração da fonte. Quando o comando começa a ser digitado, a cor atribuída será azul. Assim que o conjunto de caracteres adquirir a forma de um comando conhecido, a sua coloração será preta.

Existem centenas de comandos que podem ser utilizados, de modo que torna-se inviável discutir sobre todos eles. A seguir serão apresentados alguns comandos básicos de grande utilidade.

1.2.1 Aproximação Numérica

Conforme mostrado anteriormente, o comando N[ ] pode ser utilizado para encontrar uma aproximação numérica para uma determinada função. Por exemplo:

```
In[26]:=
N[Pi, 300]
Out[26]= 3.14159265358979323846264338327950288419716939937510582097494459230781640628620899862803482534211706798214808651328230664709384
4460955058223172535940812848111745028410270193852110555964462294895493038196442881097566593344612847564823378678316527120190914
145648566923460348610454326648213393607260249141 27
```

Neste comando, o segundo argumento refere-se a quantidade de casas decimais desejada.

1.2.2 Logaritmos

A função Log[x] retorna o logaritmo natural de x. Para obter o logaritmo em outra base, utiliza-se a função como Log[b,x], onde b é a base desejada. Por exemplo:

```
In[38]:=
Log[E]
Log[10, 0.1]
Out[38]= 1
Out[39]= -1.
```

1.3 Funções Trigonométricas

As funções trigonométricas são acessadas pelos seus nomes usuais, por exemplo Sin, Cos, Tan, Cot, ArcSin, etc. O único cuidado a ser tomado é utilizar o argumento das funções em radianos. Por exemplo:

```
In[50]:=
        Cos[Pi]
        N[Cos[180]]
Out[50]= -1
Out[51]= -0.59846
```

Na próxima seção será tratado sobre a definição e manipulação de funções matemáticas, sendo que muitos outros comandos serão apresentados ao longo do desenvolvimento do nosso estudo.

1.4 Exercícios Propostos I

Resolva as seguintes expressões:

1) $y = \sin\left(\frac{\pi}{2} + \cos(2\pi - 1)\right)$

2) $a = \sum_{m=0}^{\infty} \frac{(-1)^m 4^{2m+2}}{2^{2m+2} m!(2+m)!}$

3) $f = \int_0^y (y^2 + 3xy + 3)dy$

4) $b = \frac{\partial(x^2+y^2)}{\partial x} + \frac{\partial(x^2+y^2)}{\partial y}$

2 Construindo e Visualizando Funções

O Mathematica possibilita a definição de funções arbitrárias. O contexto de função utilizado pelo programa é o mesmo utilizado na matemática clássica, ou seja, a função é um meio de relacionar um domínio com uma respectiva imagem. A definição de uma função do tipo *f(x)=a* é dada da seguinte forma:

$$f[x\_] := a$$

Por exemplo, considere a seguinte função quadrática:

```
In[5]:=
        f[x_] := x^2 + 2 * x + 1
In[6]:= y = f[2]
Out[6]= 9
```

Percebe-se que a definição da função não gera uma saída. A saída só é gerada quando define-se o argumento da função.

De maneira semelhante, pode-se definir funções de várias variáveis separando-se as variáveis por uma vírgula. Por exemplo:

```
In[15]:=
        g[y_, z_] := y^2 + z^2 + y * z
        g[4, z]
        g[4, 4]
Out[16]= 16 + 4 z + z^2
Out[17]= 48
```

Assim como para a definição dos comandos, algumas regras básicas precisam ser respeitadas na definição das funções. As principais são:

1. O argumento da função deve ser definido entre colchetes;
2. O(s) argumento(s) da função devem ser seguidos por um *underline* _;
3. A definição da função deve ser antecedida por := (símbolo chamado de SetDelayed). Isto faz com que a definição seja alocada para a função definida, o que é diferente de uma simples igualdade. Em algumas linguagens de programação, o símbolo := significa "recebe". Assim, a função *f[x_]* recebe um determinado valor, o que é diferente de afirmar que *f[x_]* é igual a determinado valor.

Dica de Sobrevivência II: *É aconselhável que o nome da função definida comece com letra minúscula, para diferenciar das funções (comandos) padrão inseridos no código do programa.*

A esta altura, cabe um comentário adicional:

Comentário Adicional: *Existe uma diferenciação clara entre parênteses, colchetes e chaves na sintaxe do programa. Os parênteses são utilizados somente para agrupar termos*

em uma expressão matemática (Ex: $(x+y)^2$), os colchetes são usados para delimitar os argumentos de uma função ou comando (Ex: $Sin[\pi]$) e as chaves possuem a importante função de definir listas (Ex: N[{Pi, E},30])

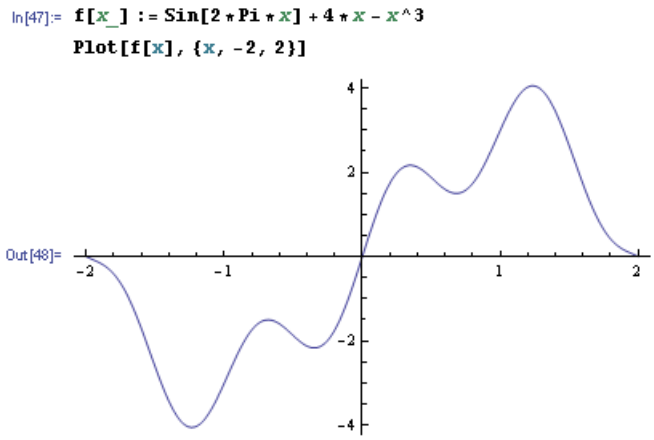
2.1 Visualização Gráfica de Funções

O comando básico para a visualização de gráficos 2D é o comando Plot. A sua sintaxe é feita da seguinte forma:

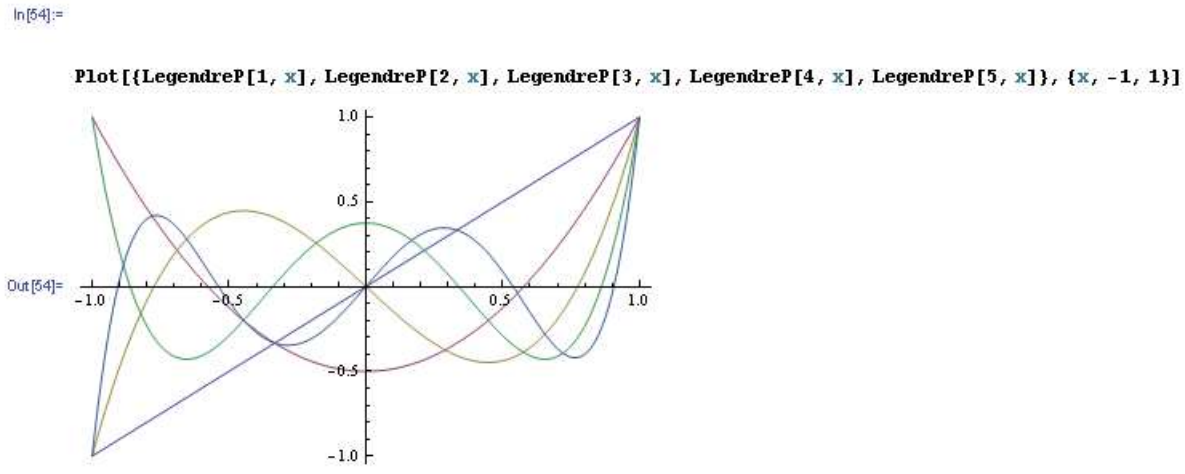
$$\text{Plot}[f[x], \{x, x_{min}, x_{max}\}]$$

onde o termo entre chaves define qual a variável utilizada para o eixo x e quais os limites desta variáveis utilizados na construção do gráfico.

Por exemplo:



O comando Plot pode ser utilizado também para a construção de gráficos com mais de uma função. Exemplo:



Onde a função $LegendreP[n, x]$ é uma função especial, que representa o polinômio de Legendre de ordem n. Mais detalhes sobre funções especiais serão abordados ao longo do curso.

A função Plot é bastante útil para a visualização de gráficos simples, porém não permite a visualização de funções de mais de uma variável. Para suprir esta necessidade, existem dois tipos de gráficos bastante utilizados, os gráficos de contorno e os gráficos 3D.

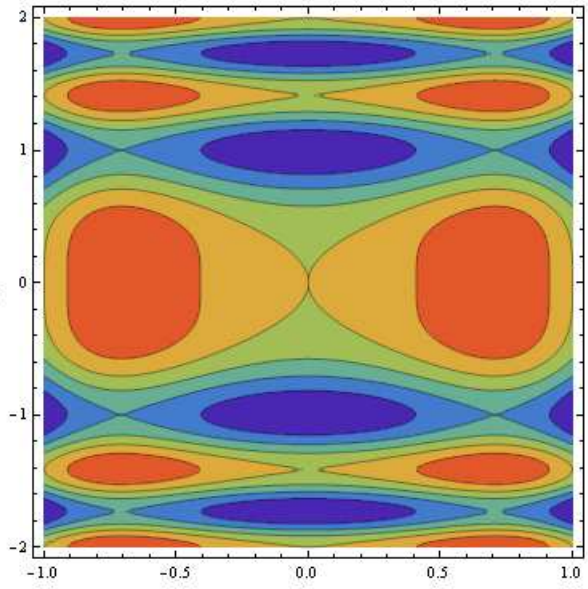
Os gráficos de contorno são acessados pelo comando CountourPlot, sendo definido como:

$$\text{ContourPlot}[f[x, y], \{x, x_{min}, x_{max}\}, \{y, y_{min}, y_{max}\}]$$

Por exemplo:

```
In[97]:=
ContourPlot[{Sin[Pi * x^2] + Cos[Pi * y^2]}, {x, -1, 1}, {y, -2, 2}, ColorFunction → "Rainbow"]
```

Out[97]=

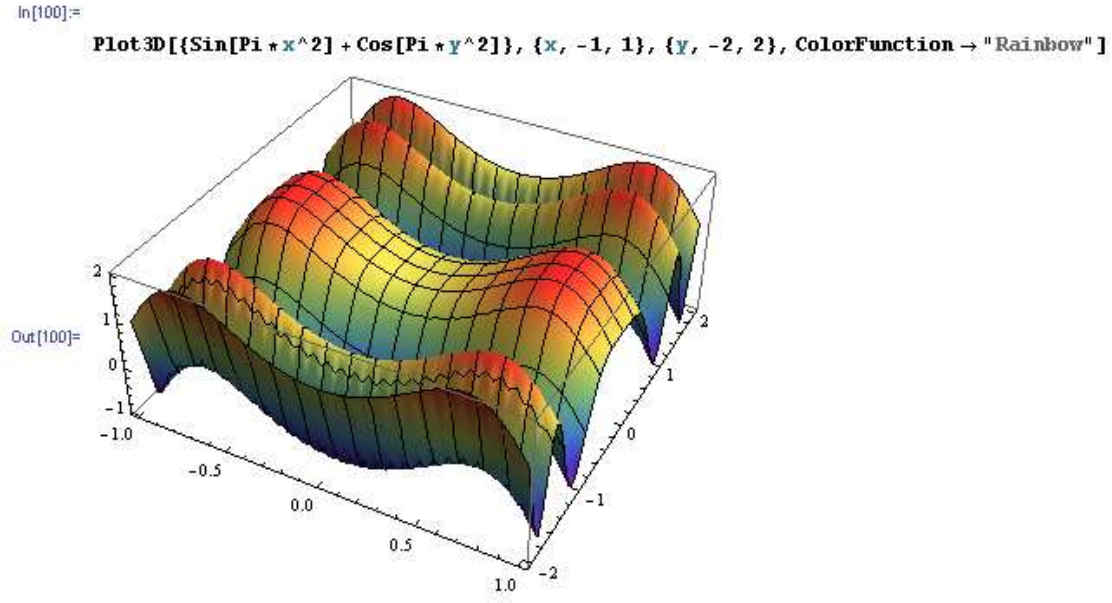


Pode-se notar que na definição deste gráfico, um quarto parâmetro é utilizado (ColorFunction). Esta função representa uma opção do gráfico, no caso, utilizada para modificar o esquema de cores. Existem muitas outras funções que podem ser inseridas de maneira semelhante, sendo que não será abordado sobre todas estas funções secundárias neste curso introdutório.

O comando Plot3D permite a visualização de gráficos em 3D, sendo um comando bastante utilizado no Mathematica. A sua forma geral é:

$$\text{Plot3D}[f[x, y], \{x, x_{min}, x_{max}\}, \{y, y_{min}, y_{max}\}]$$

Por exemplo, a mesma função do exemplo anterior, plotada em 3D resulta em:



Outra função bastante interessante na visualização de gráficos de mais de uma variável é a função Manipulate. Esta função permite a criação de uma barra de rolagem onde o valor de uma variável pode ser alterado interativamente pelo usuário. A forma básica deste comando é bastante simples:

$$\text{Manipulate}[Expr, \{Var, min, max, passo\}]$$

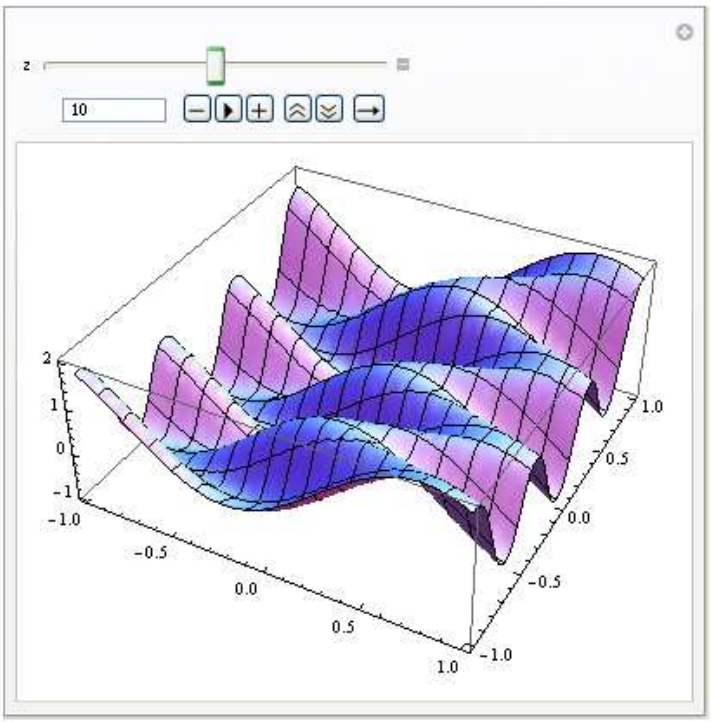
onde Expr é a expressão a ser avaliada, Var a variável manipulada, min e max os valores inicial e final da barra de rolagem e o passo representa o passo de avanço da variável (este parâmetro é optativo). Um exemplo do uso desta função é mostrado na página seguinte.

Com o conjunto de funções apresentadas até agora, é possível visualizar a grande maioria das funções de interesse. Porém, existem muitos outros tipos de gráficos disponíveis no Mathematica, como por exemplo gráficos de densidade, de vetores, de linhas de superfície, etc.

In[117]:=

```
Manipulate[Plot3D[x^2 + Sin[Pi * y + Cos[x * z]], {y, -1, 1}, {x, -1, 1}], {z, 0, 20, 0.5}]
```

Out[117]=



2.2 Exercícios Propostos II

Defina e grafique as seguintes funções:

1) $f(x) = \dfrac{\ln(2+x)}{x+3}$ 2) $f(x,y) = \exp(-(x^2+y^2))$

3) As funções de Laguerre (LaguerreL[n,x]) de ordem par (2 a 10), no intervalo de $0 \leq x \leq 6$.

4) $f(x) = \sin(x+y)^2 \times \cos(x-y)^2$, nos intervalos de $-\pi \leq x \leq \pi$ e $-\pi \leq y \leq \pi$. Utilize as funções Plot3D e ContourPlot.

Comentário Adicional II: *Quando uma função ou variável é definida, um determinado valor ou função é alocado para o símbolo utilizado. Por isso, muitas vezes torna-se necessário limpar o valor de um determinado símbolo. Isto pode ser feito através do comando Clear[f], onde f é a variável ou função a ser "limpa". Um comando bastante útil é o Clear["Global`*"], que limpa o valor de todas as variáveis e funções.*

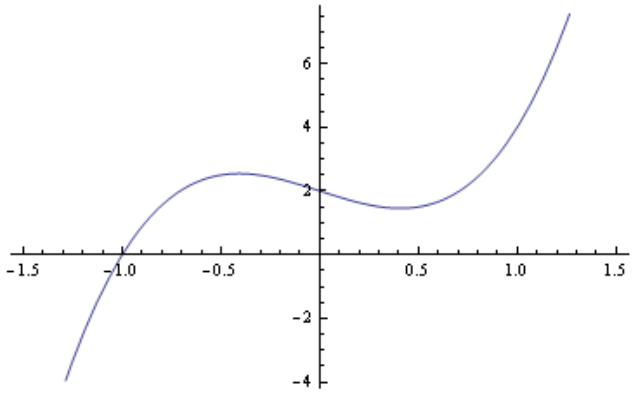
3 Manipulação Algébrica

O Mathematica possui uma extensa função nas mais diversas áreas da algebra. Nesta seção iremos focar na resolução de equações (em especial envolvendo funções trigonométricas e polinomiais).

3.1 Fatorando e Expandindo Polinômios

Quando se opera com polinômios, muitas vezes é necessário fatorá-los em diversos termos ou unir diversos termos em um único polinômio. No Mathematica, estas operações são facilmente realizadas através dos comandos Factor e Expand. Considere o exemplo a seguir

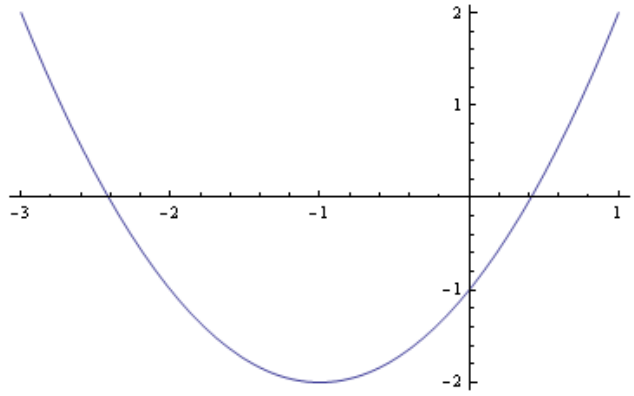
```
In[91]:= Clear[f, x];
         f[x_] := 4 x^3 - 2 x + 2
         Plot[f[x], {x, -1.5, 1.5}]
         Factor[f[x]]
```




```
Out[93]=

Out[94]= 2 (1 + x) (1 - 2 x + 2 x^2)

In[95]:=
         Expand[2 (1 + x) (1 - 2 x + 2 x^2)]

Out[95]= 2 - 2 x + 4 x^3
```

Muitas vezes (na verdade, a maioria delas), o comando Factor retorna somente a função original (sem fatorá-la). Isso ocorre pois este comando, a princípio, opera somente com a mesma classe de números definidos na função. Então, se a função for definida somente com coeficientes inteiros, e alguma das raízes for um número irracional (o que ocorre muito frequentemente) ou complexo, este fator será desconsiderado. Uma maneira muito simples de contornar este problema é representar algum dos coeficientes como um número real, como mostrado no exemplo a seguir. Este procedimento irá retornar as raízes aproximadas do polinômio.

```
In[114]:=
g[x_] := x^2 + 2 * x - 1.0
Plot[g[x], {x, -3, 1}]
Factor[g[x]]
```




```
Out[115]=

Out[116]= 1. (-0.414214 + x) (2.41421 + x)
```

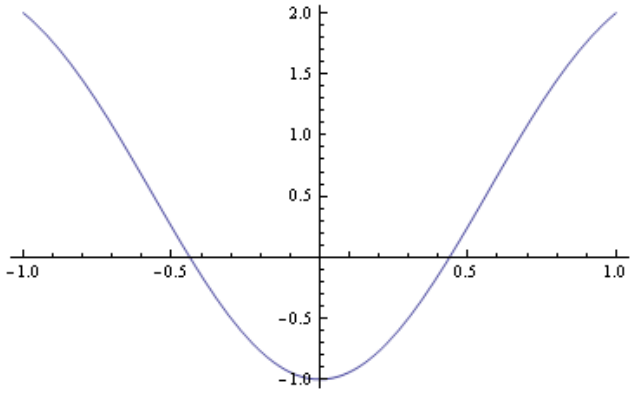
3.2 Encontrado Raízes com a Função FindRoot

A função FindRoot é uma função específica para a determinação de raízes de funções (de uma ou de um conjunto de funções). Esta função é basicamente expressa da seguinte forma:

$$\text{FindRoot}[f, \{x, x_0\}]$$

onde x_0 é o ponto inicial a partir do qual será buscada uma raíz. Esta função utiliza o método de Newton-Raphson para encontrar a raíz, daí então a necessidade de um "chute inicial". Observe o exemplo:

```
Clear[f, x]

In[136]:= f[x_] := x^2 - Cos[Pi * x]
Plot[f[x], {x, -1, 1}]
FindRoot[f[x], {x, 0}]
```




```
Out[137]=

FindRoot::jsing : Encountered a singular Jacobian at the point {x} = {0.}. Try perturbing the initial point(s). >>

Out[138]= {x → 0.}
```

Como pode ser visto pelo gráfico, a função possuir claramente duas raízes (próximas a -0.5 e 0.5). Porém, a função FindRoot não foi capaz de encontrar estas raízes a partir do chute inicial dado (x=0). Porém, uma mensagem de erro foi retorna, indicando que um ponto Jacobiano singular foi encontrado no ponto x=0. Como o método de Newton-Raphson utiliza o Jacobiano para encontrar a raíz, pode ocorrer divergência em alguns casos. A solução mais simples nestes casos é a sugerida pelo programa, ou seja, alterar o valor inicial. O resultado pode ser visto a seguir:

```
In[117]:=
        Clear[f, x]

In[144]:= f[x_] := x^2 - Cos[Pi * x]
        FindRoot[f[x], {x, 0.1}]
        FindRoot[f[x], {x, 0, -1}]

Out[145]= {x → 0.438431}

Out[146]= {x → -0.438431}
```

O comando FindRoot também pode ser utilizado para buscar uma solução numérica para uma determinada equação. Para tal, deve-se utilizar o símbolo == para igualar as funções. Veja o exemplo a seguir:

```
In[157]:=

        FindRoot[E^(x + 1) == Cos[x], {x, 0}]

Out[157]= {x → -4.73623}
```

Cabe ressaltar novamente que o símbolo == (igual) deve ser utilizado. Este símbolo expressa a igualdade entre os dois termos, e possui um significado bastante distinto do símbolo =, que é utilizado para *definir* um certo termo. Veja por exemplo o que ocorre caso o símbolo = for utilizado:

```
In[158]:=

        FindRoot[E^(x + 1) = Cos[x], {x, 0}]
        Set::write : Tag Power in e^(1+x) is Protected. >>

Out[158]= {x → 0.}
```

O programa retorna uma mensagem de erro dizendo que a expressão e^{1+x} é protegida, ou seja, ela já possui um significado. Isto ocorre pois a expressão escrita desta forma esta dizendo para o programa atribuir o valor de $\cos x$ para uma variável escrita como e^{1+x}.

3.3 Exercícios Propostos III

1) Encontre as raízes da função $f(x) = 10x^4 + 2x^3 + 5x + 1$ através de:

a) visualizando o gráfico da função (valores aproximados para as raízes reais);

b) fatorando a função;

c) através do comando FindRoot.

3.4 Resolvendo Equações com os Comandos Solve e NSolve

Os comandos Solve e NSolve (Numerical Solve) são muito utilizados para a resolução de equações, quando busca-se determinar um valor (ou expressão) para determinada variável. São bastante semelhantes ao comando FindRoot, porém são mais abrangentes. Estes comandos são expressos por:

$$\text{Solve}[eq1 == eq2, var]$$

onde *var* é a variável a ser encontrada. O comando NSolve é utilizado da mesma forma, sendo que a diferença entre eles é que o comando Solve busca uma solução algébrica, enquanto que o NSolve busca uma solução numérica. Por exemplo:

```
In[180]:=
Clear[f, x]

In[195]:=
f[x_] = a * x^2 + b * x + c
Solve[f[x] == 0, x]
NSolve[f[x] == 0, x]
```

$$\text{Out[195]=}\quad c + b\,x + a\,x^2$$

$$\text{Out[196]=}\quad \left\{\left\{x \to \frac{-b - \sqrt{b^2 - 4\,a\,c}}{2\,a}\right\}, \left\{x \to \frac{-b + \sqrt{b^2 - 4\,a\,c}}{2\,a}\right\}\right\}$$

$$\text{Out[197]=}\quad \left\{\left\{x \to \frac{0.5\left(-1.\,b - 1.\,\sqrt{b^2 - 4.\,a\,c}\right)}{a}\right\}, \left\{x \to \frac{0.5\left(-1.\,b + \sqrt{b^2 - 4.\,a\,c}\right)}{a}\right\}\right\}$$

Estas funções também podem ser utilizadas para encontrar mais de uma variável, como mostrado no exemplo a seguir.

```
In[208]:=
    g[x_, y_] := x^2 - y^2
    Solve[g[x, y] == 0, x, y]
Out[209]= {{x → -y}, {x → y}}

In[210]:=
    Solve[g[x, y] == f[x], x]
```

$$\text{Out[210]=}\ \left\{\left\{x \to \frac{-b-\sqrt{b^2+4c-4ac+4y^2-4ay^2}}{2(-1+a)}\right\}, \left\{x \to \frac{-b+\sqrt{b^2+4c-4ac+4y^2-4ay^2}}{2(-1+a)}\right\}\right\}$$

Como pode ser visto, a resposta é dada em função de uma relação entre as variáveis. Assim, no primeiro caso, a igualdade será satisfeita quando $x \to |y|$

Alguns comentários podem ser feitos a respeito a respeito da utilização destas funções:

1. A igualdade das funções deve ser definida com ==;
2. Quando não existir solução para as equações, a função Solve retorna o símbolo { };
3. Quando as variáveis podem assumir qualquer valor, a função Solve retorna o símbolo { { } };
4. A função NSolve[f[x],x] retorna o mesmo valor de N[Solve[f[x],x]].

3.5 Resolvendo Sistemas de Equações

Os comandos Solve e NSOlve podem ser utilizados também para a resolução de sistemas de equações. Para tanto, pode-se utilizar uma lista de expressões associadas a uma lista de variáveis a serem encontradas. Vale lembrar que a inserção de listas é feita através do uso de { }. A expressão geral para a solução de um sistema de duas equações (que pode ser extendido para quantas equações forem necessárias) é da forma:

$$\text{Solve}[\{exp1, exp2\}, \{var1, var2\}]$$

Por exemplo, um sistema de três equações lineares pode ser facilmente resolvido:

```
In[3]:=
    Solve[{2x + 3y + z == 0, 4y + 4x - 2z == 0, x + y + z == 1}, {x, y, z}]
```

$$\text{Out[3]=}\ \left\{\left\{x \to \frac{5}{3}, y \to -\frac{4}{3}, z \to \frac{2}{3}\right\}\right\}$$

Este método pode ser utilizado também para a resolução de sistemas não-lineares. Porém, deve-se sempre estar atento para o fato de que alguns sistemas não podem ser resolvidos algebricamente. Considere o exemplo:

```
In[8]:=
Solve[{Sin[x] + Log[y] == 0, x * y^2 == 1}, {x, y}]
Solve::tdep : The equations appear to involve the variables to be solved for in an essentially non-algebraic way. >>
Out[8]= Solve[{Log[y] + Sin[x] == 0, x y^2 == 1}, {x, y}]
```

Este sistema de equações não pode ser resolvido algebricamente (conforme indicado pela mensagem de erro), pois contém a expressão transcendental $\sin(x) = -\ln(\sqrt{1/x})$. Para resolver este tipo de problema, pode-se recorrer a uma aproximação numérica, tal como a função FindRoot:

```
In[28]:=
y = Sqrt[1/x]
FindRoot[{Sin[x] + Log[y] == 0}, {x, 0.1}]
Out[28]= Sqrt[1/x]
Out[29]= {x → 2.63571}
```

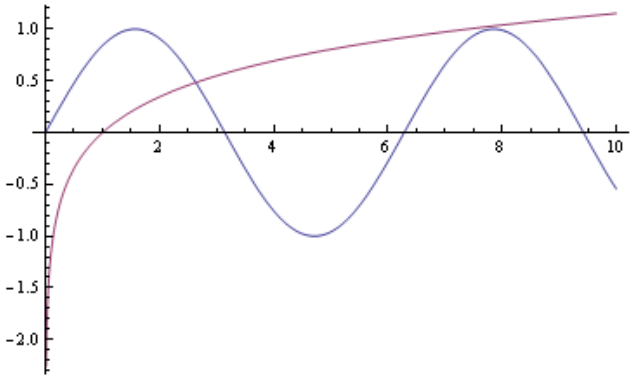
Porém, como se trata de uma função transcendental, podem existir outros valores de x (neste caso, outros autovalores) que satisfaçam a igualdade. Quando se avalia a função em um intervalo mais amplo, obtém-se o seguinte resultado:

```
In[74]:=
y = Sqrt[1/x]
FindRoot[{(Sin[x] + Log[y]) == 0}, {x, 0.1}]
FindRoot[{(Sin[x] + Log[y]) == 0}, {x, 6}]
Plot[{Sin[x], -Log[Sqrt[1/x]]}, {x, 0, 10}]
Out[74]= Sqrt[1/x]
Out[75]= {x → 2.63571}
FindRoot::lstol :
  The line search decreased the step size to within tolerance specified by AccuracyGoal and PrecisionGoal
      but was unable to find a sufficient decrease in the merit function. You may need
      more than MachinePrecision digits of working precision to meet these tolerances. >>
Out[76]= {x → 7.78975}
```

Avaliando o comportamento das duas funções envolvidas graficamente, obtém-se:

```
Plot[{Sin[x], -Log[Sqrt[1/x]]}, {x, 0, 10}]
```

Out[77]=



Avaliando uma raíz na região próxima a $x = 8$, o programa detectou que $x \approx 7.78975$ é uma solução para o sistema de equações. Porém, uma mensagem de erro foi emitida, informando que a raíz não obteve a tolerância especificada pelos parâmetros AccuracyGoal e PrecisionGoal. Estes parâmetros são utilizados para definir a precisão da aproximação numérica, representando o número de digitos efetivos de precisão que a resposta final deve ter.

Como pode ser visto pelo gráfico, as funções avaliadas possuem um valor muito próximo em $x \approx 7.78975$, porém, possivelmente, este valor não representa um autovalor algébrico (exato) para o sistema de equações. Porém, como a função FindRoot é uma *aproximação* numérica, foi detectado que este valor se aproxima muito de uma raíz. Caso o critério de convergência for diminuído, ele será tratado efetivamente como uma raíz, como pode ser visto a seguir:

In[134]:=
```
y = Sqrt[1/x]
FindRoot[{(Sin[x] + Log[y]) == 0}, {x, 0.1}]
FindRoot[{(Sin[x] + Log[y]) == 0}, {x, 6}, AccuracyGoal → 1]
```

Out[134]= $\sqrt{\frac{1}{x}}$

Out[135]= {x → 2.63571}

Out[136]= {x → 7.78878}

3.6 Exercícios Propostos IV

Obtenha o valor de x nas seguintes equações:

1) $e^x = 0$ 2) $\dfrac{x^3 + 5x + 1}{x^2} = 0$ 3) $\tan(\pi x) = x$

Resolva os seguintes sistemas de equações:

4) $y = x^2$
$y = 4x^2 + 2x - 2$

5) $y = Sin[x] + Cos[y]$
$y^3 - 2xy + x = 0$