A tecnologia ao serviço do ensino da Física

J. E. Villate
Universidade do Porto, Faculdade de Engenharia
E-mail: villate@fe.up.pt

Outubro de 2009

1 O Programa *Maxima*

Maxima (`http://maxima.sourceforge.net`) é um dos sistemas de álgebra computacional (CAS) mais antigos. Foi criado pelo grupo MAC no MIT, na década de 60 do século passado, e inicialmente chamava-se *Macsyma* (*project MAC's SYmbolic MAnipulator*). *Macsyma* foi desenvolvido originalmente para os computadores de grande escala DEC-PDP-10 que eram usados em várias instituições académicas.

Na década de 80, foi portado para várias novas plataformas e uma das novas versões foi denominada *Maxima*. Em 1982. o MIT decidiu comercializar o *Macsyma* e, simultaneamente, o professor William Schelter da Universidade de Texas continuou a desenvolver o *Maxima*. Na segunda metade da década de 80 apareceram outros sistemas CAS proprietários, por exemplo, *Maple* e *Mathematica*, baseados no *Macsyma*. Em 1998, o professor Schelter obteve autorização do DOE (*Department of Energy*), que tinha os direitos de autor sobre a versão original do *Macsyma*, para distribuir livremente o código-fonte do *Maxima*. Após a morte do professor Schelter em 2001, formou-se um grupo de voluntários que continuam a desenvolver e distribuir o *Maxima* como *software* livre.

No caso dos sistemas CAS, as vantagens do *software* livre são bastante importantes. Quando um método falha ou dá respostas muito complicadas é bastante útil ter acesso aos pormenores da implementação subjacente ao sistema. Por outro lado, no momento em que começarmos a depender dos resultados de um sistema CAS, é desejável que a documentação dos métodos envolvidos esteja disponível e que não existam impedimentos legais que nos proíbam de tentar descobrir ou modificar esses métodos.

1.1 A interface do *Maxima*

Existem várias interfaces diferentes para trabalhar com o *Maxima*. Pode ser executado desde uma "consola" ou pode ser usada algumas das interfaces gráficas como: *wxmaxima*, *texmacs* ou *xmaxima*. A figura 1 mostra o aspecto da interface *xmaxima*, que é a interface gráfica desenvolvida originalmente pelo professor William Schelter.

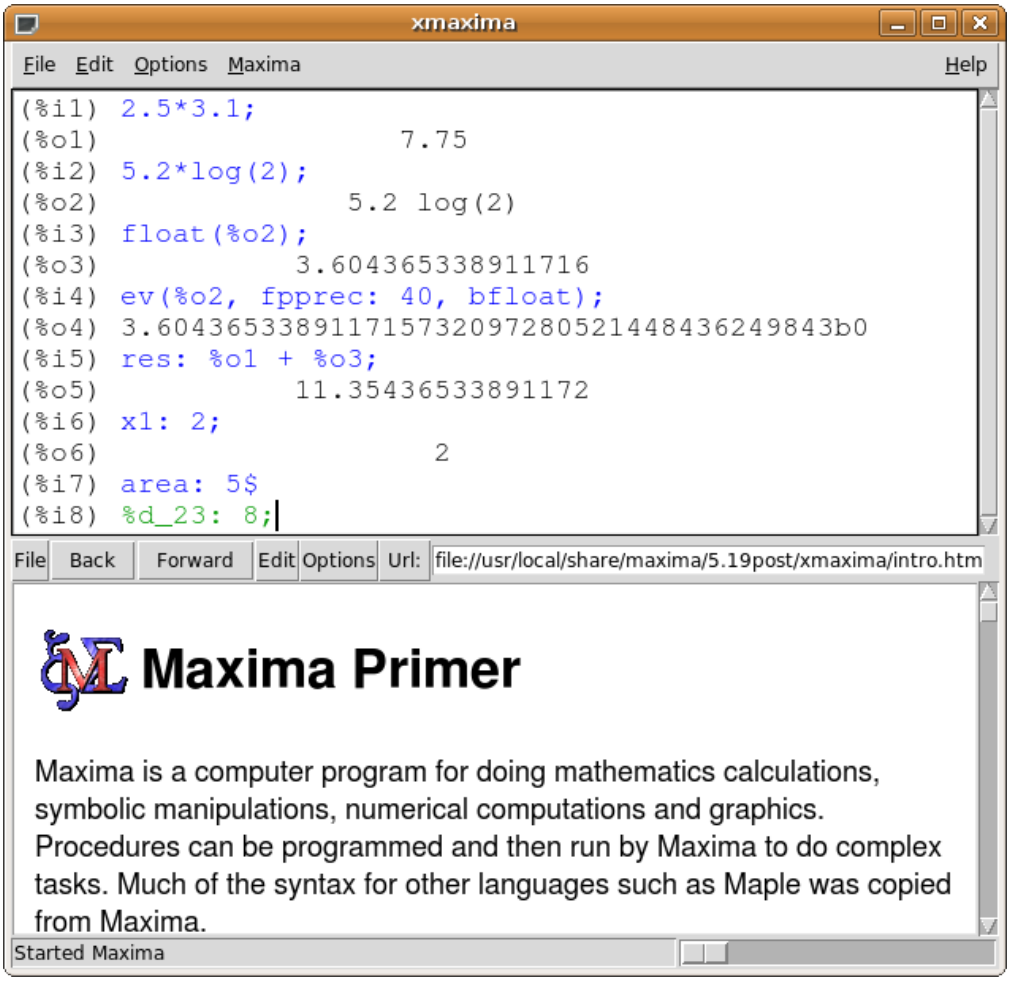

Figura 1: A interface gráfica *xmaxima*.

1.2 Entrada e saída de dados

Quando se inicia uma sessão do *Maxima*, aparece um símbolo `(%i1)`. Ao lado desse símbolo deverá ser escrito um comando válido, terminado pelo símbolo de ponto e vírgula. Carregando na tecla de fim de linha, o comando que foi escrito ficará gravado numa variável `%i1` e o resultado será gravado numa outra variável `%o1` e apresentado no ecrã. A seguir aparecerá o símbolo `(%i2)`, que permite dar um segundo comando e assim sucessivamente. Comecemos por fazer umas contas simples:

```
(%i1) 2.5*3.1;
(%o1)                          7.75
(%i2) 5.2*log(2);
(%o2)                        5.2 log(2)
```

No segundo resultado, o logaritmo natural de 2 não foi calculado, porque o resultado é um número irracional que não pode ser representado em forma numérica exacta. Se quisermos obter uma representação numérica aproximada do logaritmo de 2, podemos escrevê-lo como

`log(2.0)`; também podemos obter uma representação numérica aproximada do resultado `%o2` usando a função float:

```
(%i3) float(%o2);
(%o3)                3.604365338911716
```

A função float representa o seu argumento em ponto flutuante com 16 algarismos. O formato bfloat (*big float*) permite usar uma precisão numérica mais elevada, que pode ser controlada com a variável fpprec (*floating point precision*). Por omissão, essa variável é igual a 16; se, por exemplo, quisermos aproximar numericamente o resultado `%o2` com uma precisão de 40 algarismos significativos, usamos o comando ev, que quer dizer avaliar:

```
(%i4) ev(%o2, fpprec: 40, bfloat);
(%o4) 3.604365338911715608969607031582518153993b0
```

O valor de 40 para a variável `fpprec` só tem efeito dentro do bloco "`ev`" onde foi usado; fora do bloco, `fpprec` continua com o seu valor habitual de 16. A letra b no fim do resultado `%o4` indica que se trata de um número no formato de ponto flutuante de grande precisão. O número a seguir à letra é o expoente; nomeadamente, neste caso em que o expoente é zero, o número deverá ser multiplicado por $10^0 = 1$. O resultado `%o4` também podia ser obtido se tivéssemos escrito a entrada `%i2` na forma `5.2*log(2b0)`.

O comando ev pode ser escrito numa forma abreviada, quando não estiver dentro de outras funções, omitindo o nome ev e os parêntesis. Assim, o comando `%i4` poderia ter sido escrito de forma abreviada:

```
(%i4) %o2, fpprec: 40, bfloat;
```

1.3 Variáveis

Para dar um valor a uma variável usa-se ":" e não "=", que será utilizado para definir equações matemáticas. Por exemplo, se quisermos guardar a soma dos resultados `%o1` e `%o3` numa variável `res`,

```
(%i5) res: %o1 + %o3;
(%o5)                11.35436533891172
```

O nome das variáveis poderá ser qualquer combinação de letras, números e os símbolos % e _. O primeiro carácter no nome da variável não pode ser um número. *Maxima* faz distinção entre maiúsculas e minúsculas. Alguns exemplos:

```
(%i6) x1 : 2;
(%o6)                         2
(%i7) area : 5$
(%i8) %d_23 : 8;
(%o8)                         8
(%i9) a%2 : (x1 : x1 + 2, x1*x1);
(%o9)                        16
```

Na entrada %i7 usámos $ em vez de ponto e vírgula para terminar o comando. O sinal $ no fim faz com que o comando seja executado e o resultado gravado na variável %o7, mas sem que o resultado seja apresentado no ecrã. Vários comandos podem ser executados sequencialmente, colocando-os separados por vírgulas e entre parêntesis; isso foi feito atrás na entrada %i9; o resultado do último comando é armazenado na variável a%2; o primeiro comando na sequência incrementa o valor de x1 em 2, ficando a variável x1 com o valor de 4, e finalmente calcula-se o quadrado de x1, que fica gravado em a%2.

Alguns nomes de variáveis não podem ser usados por estarem reservados. Já vimos que nomes como %i3 ou %o6 estão reservados para referir os comandos inseridos numa sessão e os resultados obtidos. Uma variável também não pode ter o mesmo nome de algum comando do *Maxima*; por exemplo for, while e sum.

Uma variável pode conter também uma lista de valores, que são colocados entre parêntesis rectos, separados por vírgulas. Por exemplo, para criar uma lista com os quadrados dos 5 primeiros números inteiros:

```
(%i10) quadrados: [1, 4, 9, 16, 25]$
```

Os elementos da lista são contados a começar por 1; por exemplo, o terceiro elemento da lista anterior é obtido assim:

```
(%i11) quadrados[3];
(%o11)                          9
```

1.4 Constantes

Existem algumas constantes importantes já predefinidas no *Maxima*. Os seus nomes começam sempre por %. Três constantes importantes são o número π, representado por %pi, o número de Euler, e, base dos logaritmos naturais, representado por %e, e o número imaginário $i = \sqrt{-1}$, representado por %i.

Tanto %pi como %e são números irracionais, que não podem ser representados em forma numérica exacta, mas podemos obter uma aproximação numérica com o número de casas decimais desejadas; por exemplo:

```
(%i12) ev( %pi, fpprec:50, bfloat);
(%o12)   3.1415926535897932384626433832795028841971693993751b0
(%i13) float(%e);
(%o13)                2.718281828459045
```

O número %i é útil para trabalhar com números complexos. Por exemplo:

```
(%i14) (3 + %i*4) * (2 + %i*5);
(%o14)              (4 %i + 3) (5 %i + 2)
```

Para que o resultado anterior seja apresentado como um número complexo, com parte real e parte imaginária, usa-se a função rectform:

```
(%i15) rectform(%);
(%o15)                    23 %i - 14
```

O operador % refere-se ao "último resultado" no comando acima e é equivalente à variável %o14.

1.5 Expressões e equações

Uma expressão pode conter operações matemáticas com variáveis indefinidas. Por exemplo:

```
(%i16) 3*x^2 + 2*cos(t)$
```

Essas expressões podem ser depois usadas para produzir outras expressões. Por exemplo:

```
(%i17) %o16^2 + x^3;
                          2             2    3
(%o17)              (3 x  + 2 cos(t))  + x
```

Para dar valores às variáveis nessa expressão usa-se a função ev (*evaluate*):

```
(%i18) ev ( %, x=0.5, t=1.3);
(%o18)                 1.776218979135868
```

O sinal de igualdade é usado para definir equações matemáticas e não para atribuir valores como nas calculadoras; por exemplo:

```
(%i19) 3*x^3 + 5*x^2 = x - 6;
                          3      2
(%o19)               3 x  + 5 x  = x - 6
```

Para encontrar as raízes de um polinómio pode ser usado o comando allroots; por exemplo:

```
(%i20) allroots(%o19);

(%o20) [x = 0.90725099344225 %i + 0.27758381341005,
        x = 0.27758381341005 - 0.90725099344225 %i,
        x = -2.221834293486762]
```

Há duas soluções complexas e uma real. As três equações entre parêntesis rectos em %o20 fazem parte de uma lista com 3 elementos. Por exemplo, o terceiro elemento nessa lista é a raiz real:

```
(%i21) %o20[3];
(%o21)  x = - 2.221834293486762
```

A variável x continua indefinida, já que o sinal de igualdade não é usado aqui para atribuir valores numéricos às variáveis. Os resultados em %o20 são aproximados e não exactos. As raízes podem ser calculadas em forma algébrica exacta, em alguns casos, usando o comando solve, que também resolve outros tipos de equações diferentes de polinómios, em forma algébrica exacta. Por exemplo, para encontrar as raízes do polinómio atrás referido com o comando solve:

```
(%i22) solve(%o19, x)$
(%i23) float ( rectform (%) );
(%o23) [x = 0.90725099344225 %i + 0.27758381341005,
        x = - 2.221834293486761,
        x = 0.27758381341005 - 0.90725099344225 %i]
```

O resultado do comando solve não foi apresentado no ecrã porque envolve várias linhas de expressões algébricas. O resultado foi simplificado mostrando as suas coordenadas "rectangulares", com partes real e imaginária separadas (função rectform) e finalmente foi escrito em forma numérica aproximada (função float).

Para resolver um sistema de equações, que podem ser lineares ou não lineares, o primeiro argumento para o comando solve deverá ser uma lista com as equações e o segundo uma lista com as variáveis; as equações podem ser guardadas em variáveis. Por exemplo:

```
(%i24) malha1: (4 + 8)*I1 - 8* I2 = 6 + 4$
(%i25) malha2: (2+ 8 + 5 + 1)*I2 - 8*I1 = -4$
(%i26) solve([malha1,malha2],[I1,I2]);
                                    1
(%o26)              [[I1 = 1, I2 = -]]
                                    4
```

O sistema anterior também poderia ter sido resolvido mais rapidamente com o comando linsolve, em vez de solve, por se tratar de um sistema de equações lineares.

1.6 Gráficos de funções

Para desenhar o gráfico de uma ou várias funções de uma variável, usa-se o comando plot2d. Por exemplo, para desenhar o gráfico do polinómio $3x^3+5x^2-x+6$, no intervalo de x entre -3 e 1, usa-se o comando:

```
(%i27) plot2d(3*x^3 + 5*x^2 - x + 6, [x, -3, 1])$
```

É preciso indicar o domínio de valores de x que vai ser apresentado no gráfico. O resultado aparece numa nova janela (ver figura 2). Passando o rato sobre um ponto no gráfico, são apresentadas as coordenadas desse ponto. O gráfico é produzido por um programa externo, *Gnuplot*, que é instalado conjuntamente com *Maxima*. Para gravar o gráfico num ficheiro gráfico, usam-se duas opções, uma para seleccionar o tipo de ficheiro gráfico e outra para definir o nome do ficheiro.

Por exemplo, para gravar o gráfico obtido com o comando `%i27` num ficheiro GIF, podemos usar o seguinte comando:

```
(%i28) plot2d(3*x^3+5*x^2-x+6, [x,-3,1], [gnuplot_term, gif],
                          [gnuplot_out_file, "funcao1.gif"])$
```

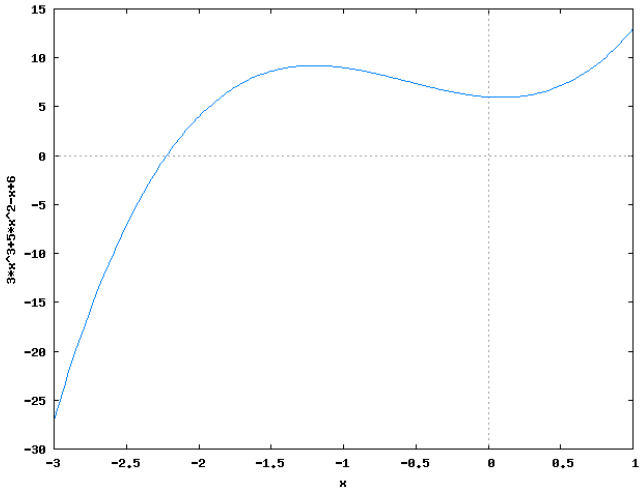

Figura 2: Gráfico do polinómio $3x^3 + 5x^2 - x + 6$.

O gráfico fica gravado no ficheiro funcao1.gif, no formato GIF. Outros dois formatos gráficos que podem ser usados são png (*Portable Network Graphics*) e ps (PostScript).

Para desenhar várias funções no mesmo gráfico, colocam-se as funções dentro de uma lista. Por exemplo:

```
(%i29) plot2d([sin(x), cos(x)], [x, -2*%pi, 2*%pi])$
```

O resultado é apresentado na figura 3.

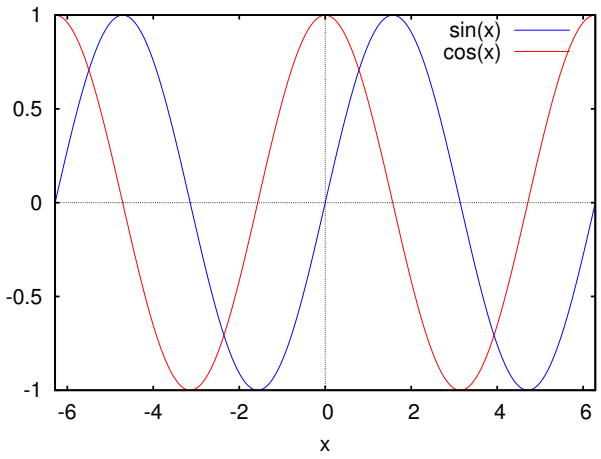

Figura 3: Gráfico das funções seno e co-seno.

É possível também fazer um gráfico de um conjunto de pontos em duas coordenadas. As duas coordenadas de cada ponto podem ser indicadas como uma lista dentro de outra lista com todos os pontos; por exemplo, para desenhar os três pontos (1.1, 5), (1.9, 7) e (3.2,9), as coordenadas dos pontos podem ser guardadas numa lista *p*:

```
(%i30) p: [[1.1, 5], [1.9, 7], [3.2, 9]]$
```

Para desenhar o gráfico, é preciso dar ao comando plot2d uma lista que comece com a palavra--chave "discrete", seguida pela lista de pontos. Neste caso não é obrigatório indicar o domínio para a variável no eixo horizontal:

```
(%i31) plot2d([discrete,p])$
```

O gráfico é apresentado na figura 4.

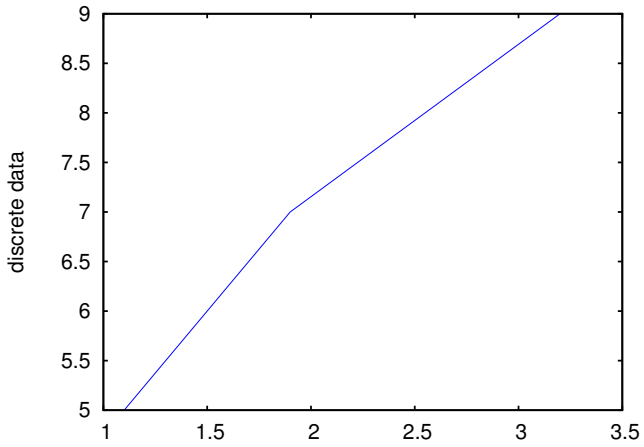

Figura 4: Gráfico de um conjunto de 3 pontos.

Por omissão, os pontos são ligados entre si por segmentos de recta; para mostrar apenas os pontos, sem segmentos de recta, usa-se a opção style, com o valor points. Podemos também combinar o gráfico dos pontos com o gráfico de uma ou várias outras funções; para o conseguir, é preciso colocar a lista com a palavra-chave discrete dentro de outra lista com as outras funções. Devido ao uso de funções, será agora necessário especificar o domínio para a variável no eixo horizontal. Podemos também especificar um domínio no eixo vertical, para uma melhor apresentação, usando a opção y:

```
(%i32) plot2d([[discrete,p], 3+2*x], [x,0,4], [y,0,15],
                                  [style, points, lines])$
```

A opção `style` em `%i34` indica que o primeiro conjunto de pontos deverá ser representado por pontos e a função que vem a seguir será representada com segmentos de recta. O gráfico é apresentado na figura 5. Existem muitas outras opções para o comando `plot2d` que podem ser consultadas na secção `plot_options` do manual. A opção y é especialmente útil para limitar os valores apresentados no eixo vertical, no caso de funções com assimptotas verticais.

Para fazer gráficos de funções de duas variáveis, em 3 dimensões, usa-se o comando `plot3d`. Por exemplo, o gráfico da figura 6 foi produzido com o comando:

```
(%i33) plot3d(sin(x)*sin(y), [x, 0, 2*%pi], [y, 0, 2*%pi]);
```

Existe ainda outro programa gráfico incluído com o *xmaxima*, para além de *Gnuplot*, designado por *Openmath*. Os gráficos anteriores podem ser produzidos com esse programa, adicionando uma opção para alterar o formato gráfico para *Openmath*. Por exemplo, o gráfico em 3 dimensões que acabámos de desenhar pode ser obtido com o *Openmath* como se segue:

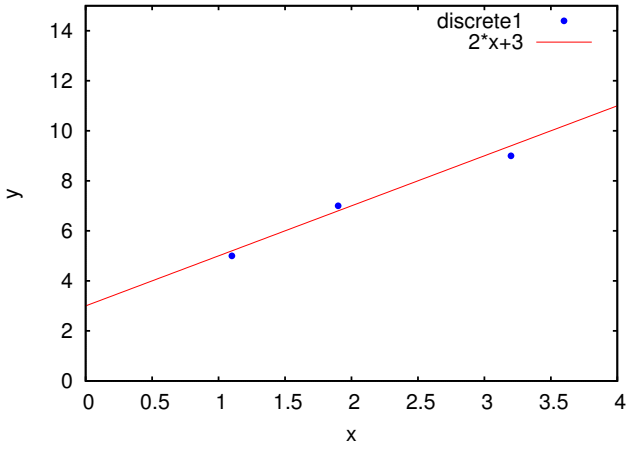

Figura 5: Gráfico de um conjunto de 3 pontos, conjuntamente com uma recta.

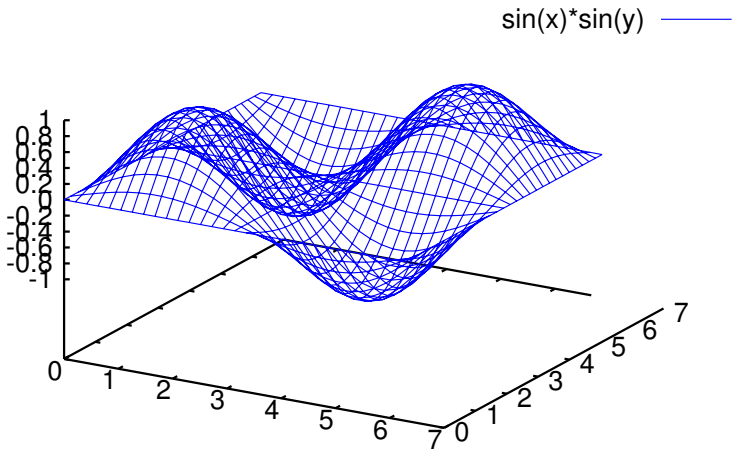

Figura 6: Gráfico da função $\sin(x)\sin(y)$, obtido com **Gnuplot**.

```
(%i34) plot3d(sin(x)*sin(y), [x, 0, 2*%pi], [y, 0, 2*%pi],
                                    [plot_format,openmath]);
```

O resultado é apresentado na figura 7.

Clicando no botão do rato enquanto se desloca, consegue rodar-se o gráfico para ser visto de diferentes pontos de vista. O comando plot3d não admite várias funções em simultâneo. O primeiro argumento de plot3d deverá ser uma única função, ou uma lista de 3 funções, que representam as 3 componentes do vector posição que define uma superfície em 3 dimensões (gráfico paramétrico).

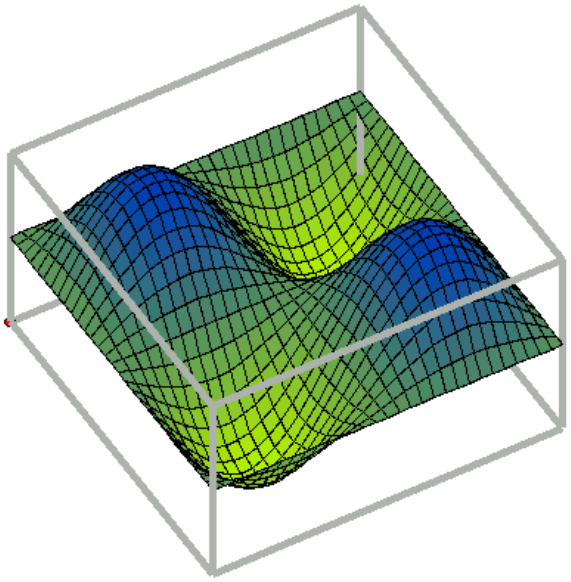

Figura 7: Gráfico da função $\sin(x)\sin(y)$, obtido com **Openmath**.

1.7 Procedimentos

Para definir procedimentos usa-se o símbolo :=. Alguns exemplos:

```
(%i35) kill(x)$
(%i36) f(x) := 3 + x^2;
                                    2
(%o36)                f(x) := 3 + x
(%i37) f(5);
(%o37)                        28
(%i38) g(x,y,z) := x*y^2 + z;
                                    2
(%o38)             g(x, y, z) := x y  + z
(%i39) g(2,3,4);
(%o39)                        22
```

O comando kill foi usado para eliminar qualquer valor que tenha sido associado à variável que usámos a seguir, x, para representar um valor de entrada qualquer. Estes procedimentos não são funções no sentido matemático; cada procedimento pode ser visto como a definição dum novo comando para o *Maxima*. Falaremos da representação de funções matemáticas na secção sobre cálculo (secção 1.9).

Também é possível definir procedimentos que definem sequências com valores de entrada inteiros. Por exemplo:

```
(%i40) cubo[n] := n^3;
                                  3
(%o40)                  cubo   := n
                           n
```

```
(%i41) makelist(cubo[i], i, 1, 8);
(%o41)      [1, 8, 27, 64, 125, 216, 343, 512]
```

O comando makelist foi usado para criar uma lista com os oito primeiros cubos. Há que ter algum cuidado com os procedimentos que definem sequências, pois os elementos já calculados ficam guardados na memória e o procedimento não voltará a ser executado quando for pedido um elemento da sequência que já foi determinado. Isso é uma vantagem do ponto de vista computacional, mas poderá conduzir a erros se não se tiver algum cuidado. Por exemplo, se agora redefinirmos cubo

```
cubo[n] := 4*n^3;
```

e pedirmos o valor de `cubo[3]`, este continuará com o valor 27, e não 108, pois quando criámos a lista dos oito primeiros cubos, já foi calculado `cubo[3]`, ficando com o valor 27. Se quisermos redefinir uma sequência, primeiro será preciso apagá-la usando kill, assim:

```
(%i42) kill(cubo)$
(%i43) cubo[n] := 4*n^3$
(%i44) cubo[3];
(%o44)                       108
```

1.8 Álgebra e trigonometria

Maxima facilita a manipulação de expressões algébricas. Por exemplo, vamos expandir um polinómio:

```
(%i45) (x + 4*x^2*y + 2*y^2)^3;
                         2      2       3
(%o45)             (2 y  + 4 x  y + x)
(%i46) expand(%);
          6       2  5       4  4         4       6  3
(%o46) 8 y  + 48 x  y  + 96 x  y  + 12 x y  + 64 x  y
                    3  3       5  2      2  2       4      3
              + 48 x  y  + 48 x  y  + 6 x  y  + 12 x  y + x
```

O comando factor é usado para factorizar polinómios. Outros comandos úteis para simplificar expressões algébricas são ratsimp e radcan. Será preciso experimentar cada um deles num caso concreto, pois alguns simplificam melhor algumas expressões do que outras.

Para substituir uma expressão algébrica por outra, por exemplo, para substituir x por $1/(z-3)$ no resultado `%o46`, podemos fazê-lo como se mostra:

```
(%i47) ev ( %, x=1/(z-3) );
```

```
           4         5           2           3
       12 y     48 y        6 y       48 y         1
(%o47) ----- + -------- + -------- + -------- + --------
       z - 3          2           2          3          3
              (z - 3)     (z - 3)    (z - 3)    (z - 3)
                4                      2          3
             96 y         12 y       48 y       64 y         6
          + -------- + -------- + -------- + -------- + 8 y
                   4          4          5          6
            (z - 3)    (z - 3)    (z - 3)    (z - 3)
```

e para reduzir tudo a um denominador comum usamos a função ratsimp (o resultado ocupa várias linhas e não vamos apresentá-lo):

```
(%i48) ratsimp(%);
```

Existem também vários comandos para simplificar expressões com funções trigonométricas. A função trigexpand serve para expandir senos ou co-senos de somas ou diferenças de ângulos:

```
(%i49) trigexpand(sin(u+v)*cos(u)^3);
             3
(%o49)    cos (u) (cos(u) sin(v) + sin(u) cos(v))
```

trigreduce tenta expandir de forma que cada termo só tenha uma função seno ou co-seno:

```
(%i50) trigreduce(%);
       sin(v + 4 u) + sin(v - 2 u)
(%o50) ---------------------------
                     8
                                    3 sin(v + 2 u) + 3 sin(v)
                                  + -------------------------
                                                8
```

O comando trigsimp usa a identidade trigonométrica $sin^2x + cos^2x = 1$ e as relações entre as funções trigonométricas para tentar escrever uma expressão apenas em termos das funções seno e co-seno. Por exemplo:

```
(%i51) tan(x)*sec(x)^2 + cos(x)*(1 - sin(x)^2);
              2                              2
(%o51)     sec (x) tan(x) + cos(x) (1 - sin (x))
(%i52) trigsimp(%);
                                 6
                       sin(x) + cos (x)
(%o52)                 ----------------
                              3
                         cos (x)
```

1.9 Cálculo

A forma mais conveniente de definir funções matemáticas consiste em usar uma variável com a expressão que define a função. Por exemplo, a função $f(x,y) = x^3/y$ seria definida assim:

```
(%i53) f: x^3/y;
                                3
                               x
(%o53)                         --
                               y
```

Para calcular o valor da função para valores dados das variáveis, usaremos a função ev:

```
(%i54) ev ( f, x=2, y=3 );
                               8
(%o54)                         -
                               3
```

Para calcular a derivada de uma função, usa-se o comando diff. O primeiro argumento deverá ser uma expressão de uma ou mais variáveis, o segundo argumento é a variável em ordem à qual vai ser derivada a função e um terceiro argumento optativo, que indica a ordem da derivação (se não aparecer entender-se-á que se trata de uma derivada de primeira ordem). Alguns exemplos, usando a função f atrás definida:

```
(%i55) diff(x^n, x);
                                 n - 1
(%o55)                        n x
(%i56) diff(f, x, 2);
                               6 x
(%o56)                         ---
                                y
(%i57) diff(f, y, 1, x, 2);
                                6 x
(%o57)                        - ---
                                  2
                                 y
```

Em `%i57` foi calculada a derivada parcial $\partial^3 f/\partial y \partial^2 x$.

Para calcular primitivas, usa-se `integrate`, com a expressão a integrar, seguida da variável de integração. Por exemplo, a primitiva de x^n obtém-se do seguinte modo:

```
(%i58) integrate(x^n, x);
Is  n + 1  zero or nonzero?
nonzero;
                              n + 1
                             x
(%o58)                       ------
                             n + 1
```

Maxima perguntou se $n+1$ é nula, isto é, se n é igual a -1. A nossa resposta foi "nonzero", seguido por ponto e vírgula, que produz o resultado anterior, para n diferente de -1.

Um integral definido calcula-se de forma semelhante, incluindo os limites de integração a seguir à variável de integração; por exemplo:

```
(%i59) integrate(1/(1 + x^ 2), x, 0, 1);
                                %pi
(%o59)                          ---
                                 4
```

1.10 Guardar informação entre sessões

Para guardar o conteúdo de uma sessão em *xmaxima*, existe a opção **Save Console to File** no menu "Edit". Essa opção guarda toda a informação que apareceu no ecrã, incluindo os símbolos `%i` e `%o`.

Para gravar os comandos executados, numa forma que possa ser aproveitada em sessões posteriores, usa-se o comando stringout. Vejamos um exemplo[1]:

```
(%i60) stringout("/home/maxima/trig.txt", %i49, %o49)$
(%i61) stringout("/home/maxima/graficos.txt", [27, 33])$
(%i62) stringout("/home/maxima/tutorial.txt", input)$
```

No ficheiro /home/maxima/trig.txt fica armazenado o comando da entrada `%i49` e a resposta `%o49`. No ficheiro /home/maxima/graficos.txt ficam guardados os comandos (`%i27`, `%i28`, ..., `%i33`). Finalmente, o ficheiro /home/maxima/tutorial.txt terá uma cópia de todos os comandos usados neste apêndice. O conteúdo desses ficheiros é texto simples, que pode ser modificado com um editor de texto e executado posteriormente usando a opção **Batch file**, no menu "File" do *xmaxima*, ou com o comando batch("nome_do_ficheiro").

Pode procurar mais informação sobre o programa *Maxima* na Web, no sítio: `http://maxima.sourceforge.net`.

2 Python e VPython

O **Python** é uma linguagem de programação que tem ganho muita popularidade no ensino e na investigação, por ser de aprendizagem clara e devido à sua facilidade de extensão que faz com que existam muitos módulos disponíveis.

Um desses módulos que foi usado para criar as simulações incluídas no Manual Virtual do livro *Eu e a Física 12.º* é o **VPython**, que inclui classes para criar vários tipos de formas geométricas a três dimensões, que podem ser colocadas facilmente em movimento. VPython usa a livraria gráfica **OpenGL**.

[1] Em Windows será preciso usar algo como `C:\\MeusDocumentos\\trig.mac` para os nomes dos ficheiros (com barras a dobrar).

Python, VPython e OpenGL são *software* livre, que podem ser instalados e utilizados em todos os principais sistemas operativos. Na página Web do VPython (`http://www.vpython.org/`) pode ser descarregado um pacote que inclui o VPython, o Python e uma interface gráfica para o Python: **Idle**. Em algumas distribuições do sistema GNU/Linux os programas Python, VPython e Idle já estão incluídos em pacotes separados.

2.1 Idle

A interface Idle pode ser lançada desde um menu ou usando o comando "idle" numa consola. Quando o programa Idle arranca, é criada uma janela *Python Shell* (figura 8), onde podem ser escritos comandos do Python em forma interactiva. Os três caracteres >>> indicam o ponto na janela onde deverá ser inserido o próximo comando.

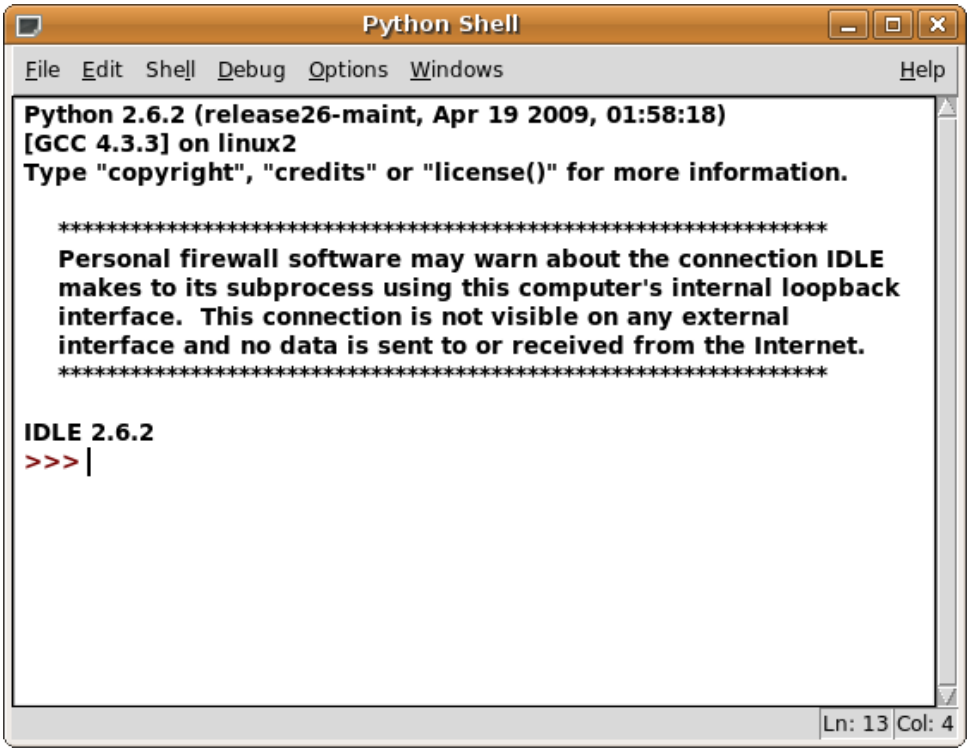

Figura 8: A interface do programa Idle.

O menu "File" permite abrir um programa já escrito, em outra janela, e executá-lo. Por exemplo, para abrir e executar a simulação de uma bola saltitona, copie o ficheiro `bola.py` do Manual Virtual para o seu computador; a seguir use a opção "Open" no menu "File" para abrir esse ficheiro. O ficheiro aparecerá numa nova janela, tal como no lado esquerdo da figura 9.

A janela onde aparece o programa terá também um menu "Run" que pode usar para executar o programa. Quando executar o programa, será aberta uma terceira janela que mostra o programa em execução (lado direito da figura 9).

Na janela que mostra a bola em movimento, pode clicar no botão direito do rato e manter o botão premido enquanto desloca o rato: assim, poderá rodar a imagem a 3 dimensões. Se premir o botão do meio do rato, enquanto o desloca, mudará a escala da imagem.

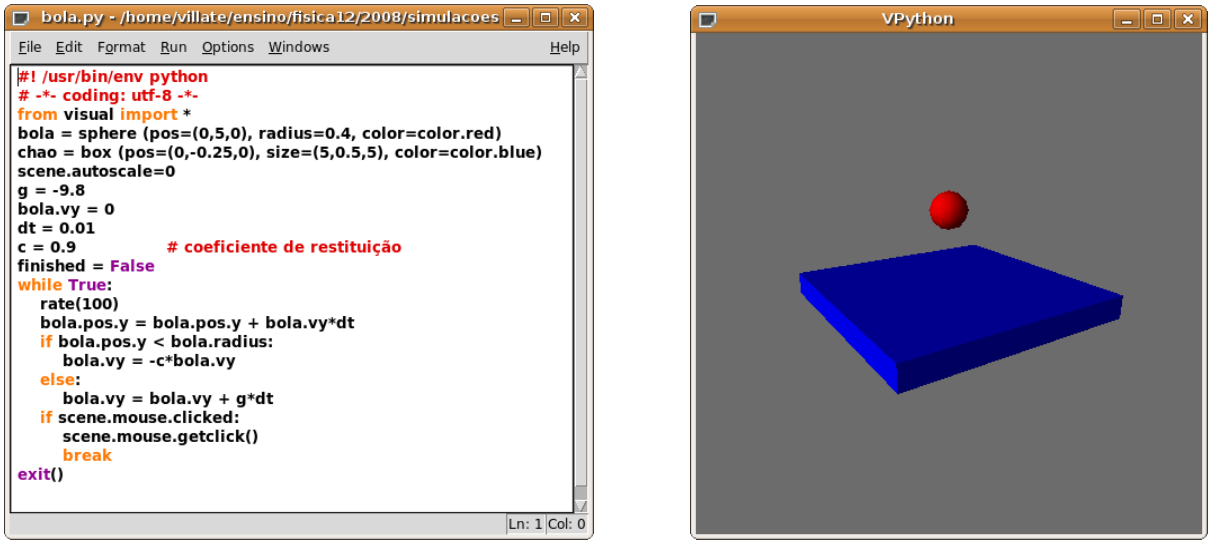

Figura 9: Simulação da bola saltitona. No lado esquerdo, a janela do código do programa e no lado direito a janela da execução do programa.

Experimente modificar o programa na janela do mesmo, por exemplo, aumentando o coeficiente de restituição de 0.9 para 1 e a seguir executando-o novamente. Agora, a bola terá um movimento contínuo sem diminuição da altura e o programa nunca acabará; quando quiser parar o programa, clique com o botão esquerdo sobre a janela da simulação — aparecerá uma mensagem a pedir para confirmar que quer terminar o programa.

Repare que, enquanto o programa estiver a ser executado, existirá uma janela *Python Shell* a ele associada e nela serão apresentadas as mensagens de erro e outras que sejam enviadas pelo programa.

Algumas simulações com várias etapas diferentes não começam automaticamente, mas só passam à etapa seguinte quando clicar sobre o gráfico com o botão esquerdo do rato. Por exemplo, na simulação do lançamento de projécteis (ficheiro projecteis.py) um primeiro clique faz com que seja lançado um electrão; o alcance é apresentado na janela gráfica e a janela *Python Shell* mostrará alguma informação adicional sobre a trajectória do electrão: tempo de voo, distância percorrida e velocidade média. Um segundo clique faz com que seja lançada a bola de ténis e para lançar a bola de ténis de mesa deverá clicar mais uma vez. Por último, clica-se novamente para terminar o programa.

2.2 O Python como calculadora

O Python também pode ser usado como calculadora para fazer algumas contas. Por exemplo, se quisermos calcular o tempo que a luz do sol demora a chegar à Terra, usamos o valor da distância entre a Terra e o Sol, que é de $1,496 \times 10^{11}$ m e a velocidade da luz que é $3,0 \times 10^{8}$ m/s. Assim, o tempo em segundos será:

$$\frac{1,496 \times 10^{11}}{3,0 \times 10^{8}}$$

Essa conta é feita na Shell do Python da seguinte forma:

```
>>> 1.146e11/3e8
382.0
```

O resultado 382.0 aparece (a azul) quando se carrega na tecla "Enter". Se quisermos saber esse tempo em minutos, teremos de dividir por 60; para não termos que escrever o resultado novamente, podemos recuperar a nossa primeira entrada "1.146e11/3e8" carregando nas teclas "Alt" e "p" em simultâneo e a seguir dividindo por 60:

```
>>> 1.146e11/3e8/60
6.3666666666666663
```

Também podemos usar variáveis para guardar resultados intermédios:

```
>>> tsegundos = 1.146e11/3e8
>>> tminutos = tsegundos/60
>>>
```

O valor que é guardado na variável não é apresentado, mas podemos vê-lo dando o nome da variável:

```
>>> tminutos
6.3666666666666663
```

As funções matemáticas habituais (logaritmo, seno, etc.) não estão definidas previamente, mas podem ser incorporadas importando o módulo de funções matemáticas. Uma forma de importar todas as funções e símbolos existentes num módulo, neste caso o módulo com nome **math**, é a seguinte:

```
>>> from math import *
```

Observe que quando terminamos de escrever uma palavra-chave do Python, como *from* e *import*, o Idle identifica essas palavras com a cor laranja. Essa é uma valiosa ajuda para detectar erros de sintaxe e para evitar tentar usar uma dessas palavras-chave como nome de variável, o que daria um erro.

Uma vez importado o módulo matemático **math**, teremos disponíveis dois símbolos predefinidos, o valor de π e a constante de Euler, e as 12 funções apresentadas na tabela 1.

Por exemplo, se quisermos calcular o volume, em metros cúbicos, de uma gota de água com raio $r = 2$ mm, usamos a expressão do volume da esfera, $4\pi r^3/3$:

```
>>> 4*pi*2e-3**3/3
3.3510321638291127e-08
```

O operador é usado para potências; por exemplo, a função exponencial de x pode ser escrita $\exp(x)$ ou, em forma equivalente, $e{**}x$.

Símbolo ou função	Descrição
`pi`	Número π
`e`	Número de Euler
`fabs(x)`	Valor absoluto de x
`sqrt(x)`	Raiz quadrada de x
`log(x)`	Logaritmo natural de x
`exp(x)`	Função exponencial de x
`log10(x)`	Logaritmo de base 10 de x
`sin(x)`	Seno de x (x em radianos)
`cos(x)`	Co-seno de x (x em radianos)
`tan(x)`	Tangente de x (x em radianos)
`asin(x)`	Seno inverso de x (em radianos)
`acos(x)`	Co-seno inverso de x (em radianos)
`atan(x)`	Tangente inversa de x (em radianos)
`floor(x)`	Elimina as casas decimais de x, dando um inteiro

Tabela 1: Símbolos e funções no módulo math.

2.3 Blocos iterativos e condicionais

Como em qualquer linguagem de programação, o Python inclui vários tipos de blocos iterativos e condicionais. Uma característica peculiar do Python, diferente de outras linguagens de programação, é que não são usados comandos especiais para indicar o fim de uma linha ou de um bloco. Para indicar quais os comandos que fazem parte de um bloco é preciso que esse bloco seja indentado em forma consistente: todos os comandos que pertencem a um bloco deverão ter a mesma indentação e um bloco interno deverá ter maior indentação. Para indentar as linhas pode usar-se espaço ou caracteres TAB, mas convém utilizar apenas um ou o outro consistentemente.

O Idle ajuda na tarefa de indentar as linhas. Por exemplo, se quisermos escrever no ecrã uma lista com os cubos dos primeiros 5 números naturais, basta escrevermos duas linhas de código:

```
>>> for i in [1, 2, 3, 4, 5]:
        i**3


1
8
27
64
125
```

O símbolo-chave são os dois pontos; após os escrevermos e clicarmos em "Enter", Idle já não coloca o símbolo `>>>` no início da linha, porque está a espera que completemos o bloco. A indentação da segunda linha foi feita automaticamente pelo Idle. Quando terminámos de escrever a segunda linha, foi preciso clicar duas vezes seguidas em "Enter" para que o ciclo fosse finalizado e executado.

Se, agora, quisermos escrever unicamente os números naturais com cubos que comecem pelo algarismo 2, considerando apenas os primeiros 29 números naturais, usamos um bloco condicional **if** para decidir quais são os cubos com o primeiro algarismo igual a 2:

```
>>> for n in range(1, 30):
        cubo = n**3
        ordem = floor(log10(cubo))
        if floor(cubo/10**ordem) == 2:
                print n, "ao cubo é", cubo


3 ao cubo é 27
6 ao cubo é 216
13 ao cubo é 2197
14 ao cubo é 2744
28 ao cubo é 21952
29 ao cubo é 24389
```

Neste caso, em vez de escrevermos por extenso a lista dos 29 primeiros números naturais, usamos uma função *standard* do Python, **range**, que produz essa lista. Repare que Idle identifica as funções *standard* a roxo, mas as funções definidas por módulos adicionais, como **floor**, não são identificadas.

Usamos também o comando **print** para imprimir variáveis e texto (colorido a verde pelo Idle) no ecrã.

Outros operadores para comparar números são <, >, <=, >= e != (diferente). Um bloco **if** pode ter um sub-bloco **else**, que pode incluir outro bloco **if**, usando o comando **elif**. Por exemplo:

```
>>> n = 2
>>> if 0 < n < 3:
        print "entre 0 e 3"
elif n >= 3:
        print "maior ou igual a 3"
else:
        print "menor ou igual a zero"


entre 0 e 3
```

2.4 Funções

Para definir funções, por exemplo a função **floor** definida pelo módulo **math**, escreve-se o procedimento dentro de um bloco que começa com a palavra-chave **def**, seguida pelo nome da função e a lista de variáveis de entrada entre parêntesis.

Por exemplo, para definir uma função **rad** que converta um ângulo em graus para radianos, podemos usar:

```
>>> def rad(x):
        return pi*x/180
```

a seguir já podemos calcular funções trigonométricas de ângulos em graus. Por exemplo, o seno de 30°:

```
>>> sin(rad(30))
0.49999999999999994
```

Uma função em Python pode chamar a própria função em forma recursiva.

2.5 Módulos

Um programa ou **módulo** pode ser escrito interactivamente na Shell, ou pode ser copiado para um ficheiro e depois executado. A opção **New Window** no menu "File" do Idle permite abrir outra janela de um editor de texto onde é possível escrever um programa e gravá-lo num ficheiro.

O módulo pode conter apenas definições de funções e variáveis, como no caso do módulo math. Nesse caso, para usar essas funções e variáveis em outros módulos ou na shell, é preciso importar o módulo.

O módulo pode ser também um programa que pede alguns valores de entrada através da entrada-padrão, realiza alguma acção e imprime alguns resultados na saída-padrão.

Para obter valores de entrada em forma iterativa, usa-se a função **input**. Por exemplo:

```
>>> n = input("Indique o valor de n: ")
Indique o valor de n: 7
```

Se o programa não for auto-executável, pode ser carregado com a opção **Open** no menu "File" do Idle e a seguir executado com a opção **Run Module** no menu "Run".

Existe muita documentação adicional disponível na Web. O sítio `http://www.vpython.org` é uma boa referência para encontrar manuais, tutoriais, livros e módulos adicionais.