Primeiros passos no Maxima

Mario Rodríguez Riotorto

www.biomates.net

Tradutor - Jorge Barros de Abreu

16 de janeiro de 2006

Sumário

1 Introdução

Este é um manual introdutório sobre o programa de cálculo simbólico Maxima, sucessor direto do lendário MACSYMA.

O objetivo do manual é facilitar o acesso a este programa a todas aquelas pessoas que pela primeira vez se interesam por ele.

Maxima é um programa cujo objetivo é a realização de cálculos matemáticos, tanto numéricos quanto simbólicos, capaz de manipular expressões algebrícas, derivar e integrar funções e montar diversos tipos de gráfico.

As orígens do Maxima temos que procurá-las a partir do ano de 1967 no MIT AI Lab (Laboratório de Inteligência Artificial do Instituto Tecnológico de Massachussets) como uma parte do projeto MAC (Machine Aided Cognition - Cognição Auxiliada por Máquina). O programa receberia o nome de Macsyma (MAC's SYmbolic MAnipulator - Manipulador Simbólico do MAC), do qual o MIT mandaria uma cópia em 1982 ao DOE (Department Of Energy - Departamento de Energia), um dos organismos que contribuiram financeiramente para o desenvolvimento do projeto; esta primeira versão é conhecida como DOE-Macsyma. Posteriormente, o DOE concede a licença de exploração do programa à empresa Symbolics, que segue desenvolvendo o projeto durante alguns anos. Em 1992 o programa é adquirido por uma empresa que se chamaria precisamente Macsyma Inc, e o programa iria perdendo fôlego progressivamente diante a presença no mercado de outros programas similares como Maple ou Mathematica, ambos inspirados em suas orígens pelo próprio Macsyma.

Ocorreram duas histórias paralelas. Desde o ano de 1982, e até seu falecimento em 2001, William Schelter na Universidad do Texas manteve uma versão deste programa adaptada ao Lisp Comum padrão, a qual se conhecia com o nome de Maxima para diferenciá-la da versão comercial. No ano de 1998 Schelter conseguiu do DOE permissão para distribuir Maxima sob a licença GNU-GPL (www.gnu.org); com esse passo, muito mais pessoas começaram a dirigir seu olhar na direção do Maxima, justo no momento em que a versão comercial estava praticamente morta. Atualmente, o projeto está sendo liderado por um grupo de desenvolvedores oriundos de vários países, tanto do meio unversitário como do meio empresarial, assistidos e ajudados por outras muitas pessoas interessadas no Maxima e que mantêm um canal de comunicação através de uma lista de e-mails (maxima.sourceforge.net/maximalist.html).

Devido ao fato de que Maxima é distribuído sob a licença GNU-GPL, tanto o código fonte como os manuais são de livre acesso através da página web do projeto (maxima.sourceforge.net).

Como professor de matemáticas, não resisto a traduzir umas linhas do capítulo

introdutório do manual oficial do programa:

> Aquele que se prestam a utilizar o computador para fazer matemática, particularmente os estudantes, devem ter sempre em mente que estes ambientes não são substitutos do trabalho manual com as equações nem do esforço de compreender os conceitos. Estes meios não ajudam a formar a intuição nem a reforçar os conhecimentos fundamentais... Não se deve utilizar o computador como um substituto da formação básica.
>
> Sem impedimentos, o domínio do computador e das ferramentas matemáticas computacionais são cruciais na hora de abordar o grande número de problemas que não podem ser resolvidos simplesmente com lápis e papel. Em muitos casos, problemas que demorariam anos para serem resolvidos de forma manual podem ser resolvidos em questão de segundos com um computador... Além disso, em caso de erro, sua correção será mais rápida e simplesmente voltando a executar um código já escrito, mas convenientemente modificado para sanar a falha.
>
> Se bem que o computador pode corrigir erros humanos, o humano por sua parte tampouco deve confiar no computador de forma inquestionável. Todos estes sistemas possuem seus limites e quando esses limites são alcançados, é possível obter respostas incorretas... Dessa forma, não se pode abrir mão de revisar os resultados que forem obtidos. O computador não diz sempre a verdade, e se a disser, certamente não será completa.
>
> *Ferrol-A Coruña*

2 Instalação

É possível ter o programa Maxima tanto em Linux como em Windows. A informação para a instalação nestes dois sistemas pode ser encontrada na página *web* do projeto.

No que se refere ao Linux, o pacote básico tem o programa Maxima no ambiente do console de texto, Figura 1, mas também é possível a instalação de módulos ou programas adicionais que permitam a utilização do programa através de um ambiente gráfico; aqui se pode escolher várias opções: xmaxima, Figura 2, baseado em Tcl-Tk e disponível na própria página do projeto Maxima; wxmaxima (wxmaxima.sourceforge.net), Figura 3, baseado em wxWidgets que surgiu mais recentemente; também existe a possibilidade de acessar o Maxima a partir de uma sessão do editor de texto WYSIWYG TeXmacs (www.texmacs.org), Figura 4, que ao fazer uso das fontes TEX
(www.tug.org/teTeX) aumenta consideravelmente a qualidade na impressão de fórmulas e expressões matemáticas. Um programa adicional que não deve faltar é o gnuplot (www.gnuplot.info) para a representação de gráficos em 2D e 3D. Todos estes programas podem ser baixados gratuitamente da internet. Para a instalação destes programas deve-se ler a documentação correspondente a cada caso.

Também, e para os mais valentes, é possível baixar o código fonte e compilá-lo, para o que será necesario ter operacional um ambiente Lisp na máquina, como GCL (www.gnu.org/software/gcl), CLISP (clisp.cons.org) o CMUCL (www.cons.org/cmucl).

Quanto ao Windows, também a partir da página do projeto pode-se baixar um executável que instala xmaxima, com uma aparência similar à da Figura 2. Se o wxmaxima tiver a preferência, uma vez instalado o pacote anterior (xmaxima), se fará uso do executável para este sistema operacional que se encontrará no sítio wxmaxima.sourceforge.net.

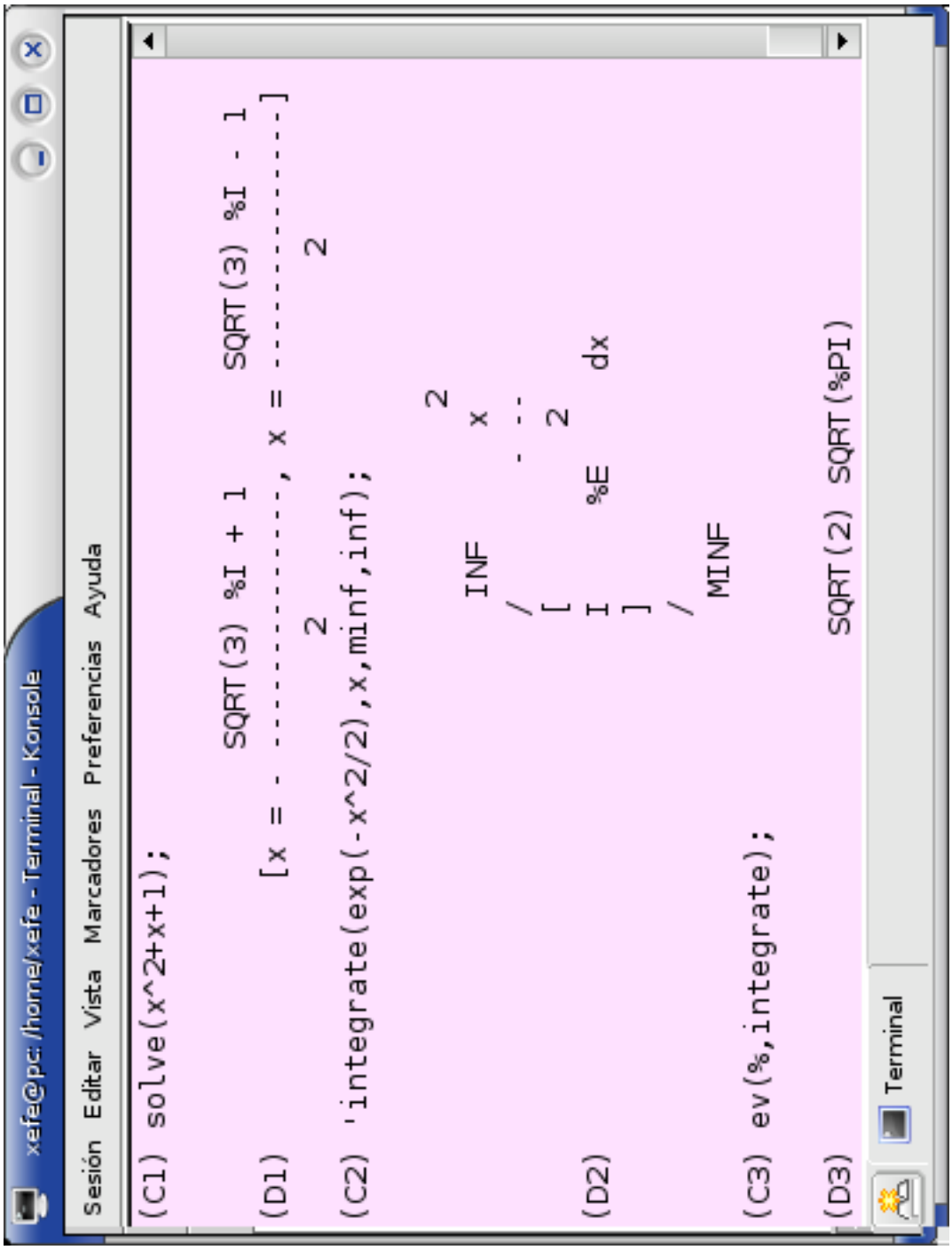


Figura 1: Maxima operando no console modo texto.

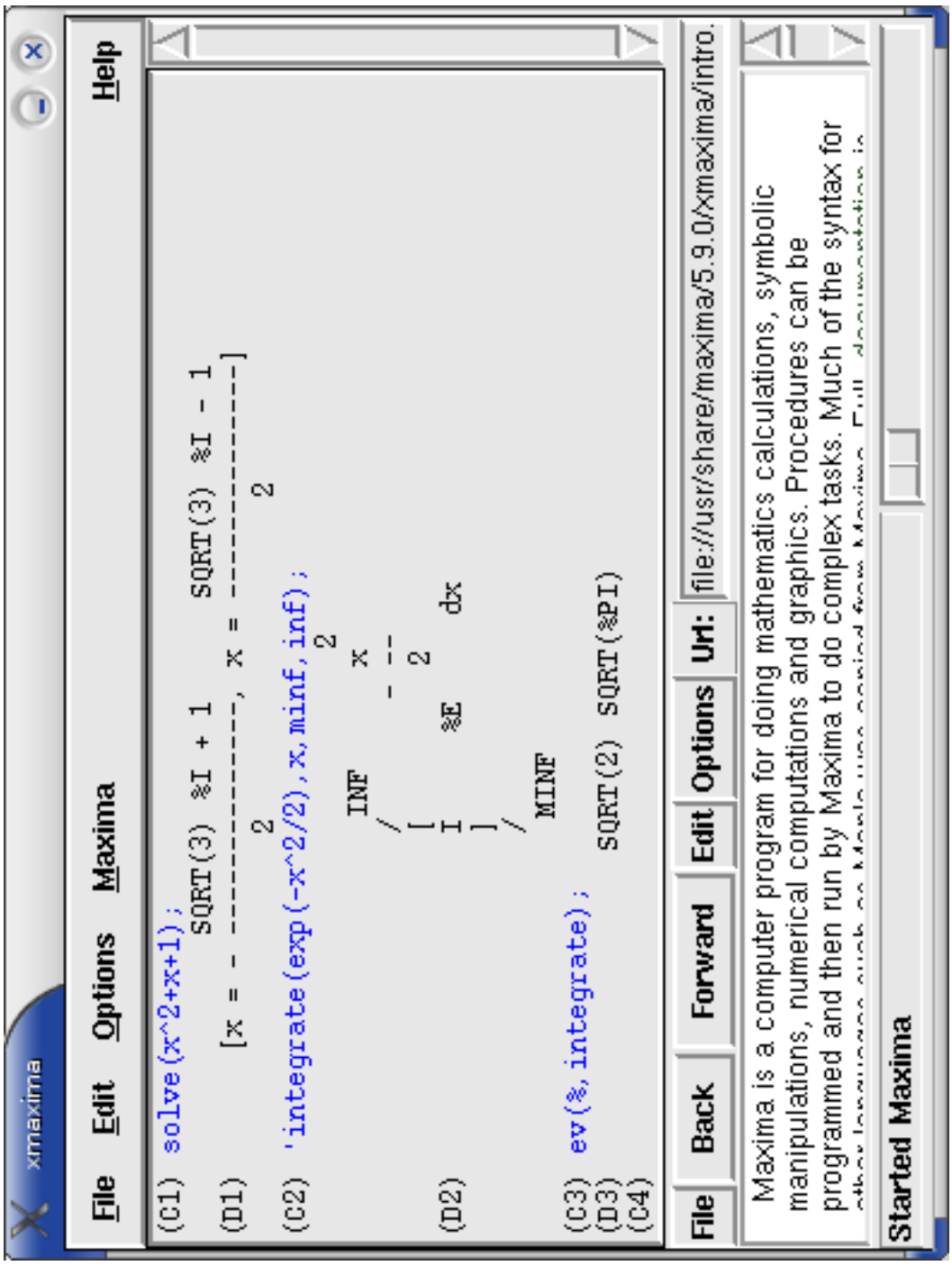


Figura 2: xmaxima: Maxima trabalhando em ambiente gráfico Tcl-Tk.

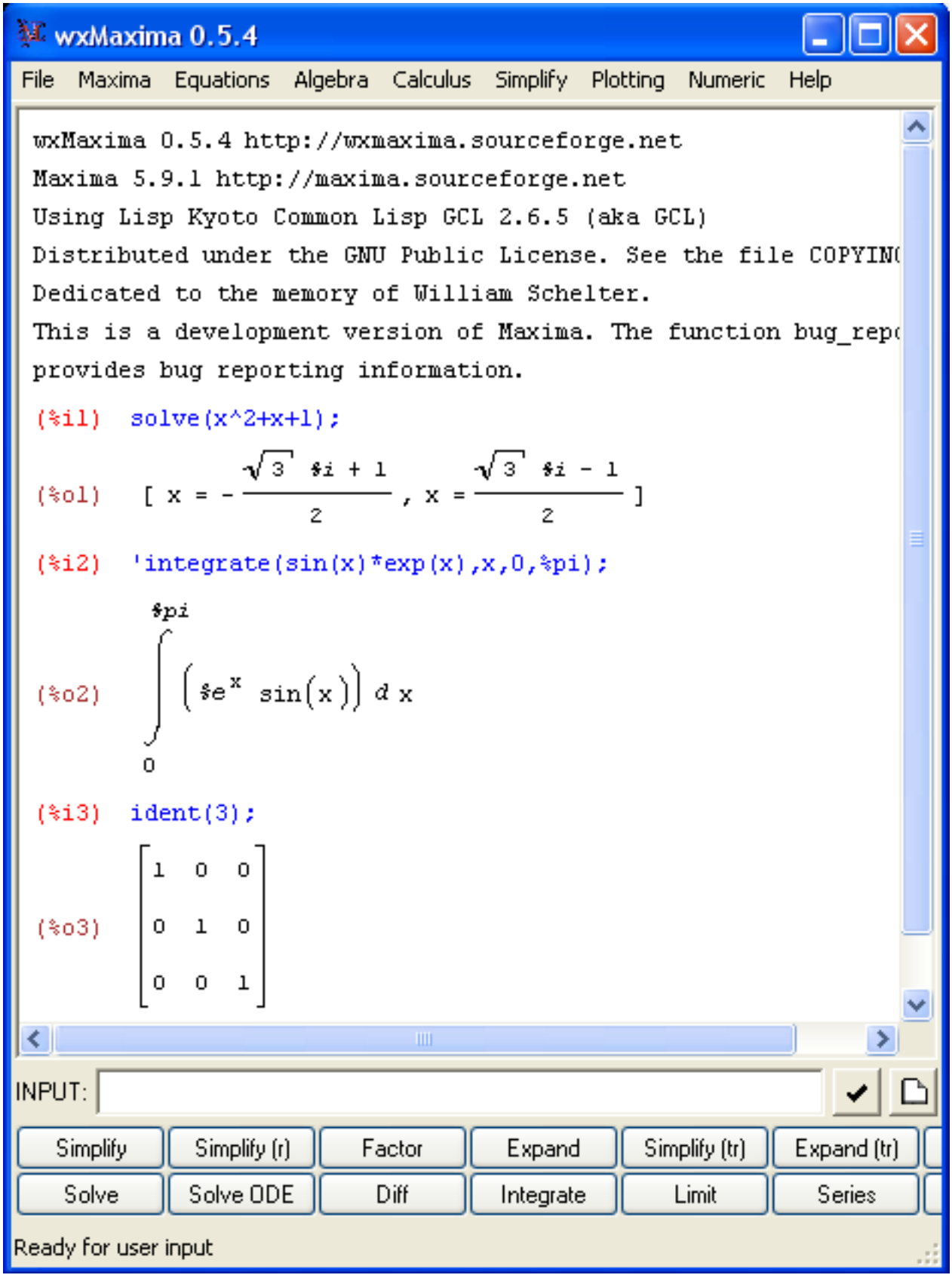


Figura 3: wxmaxima: Maxima sendo executado no Windows.

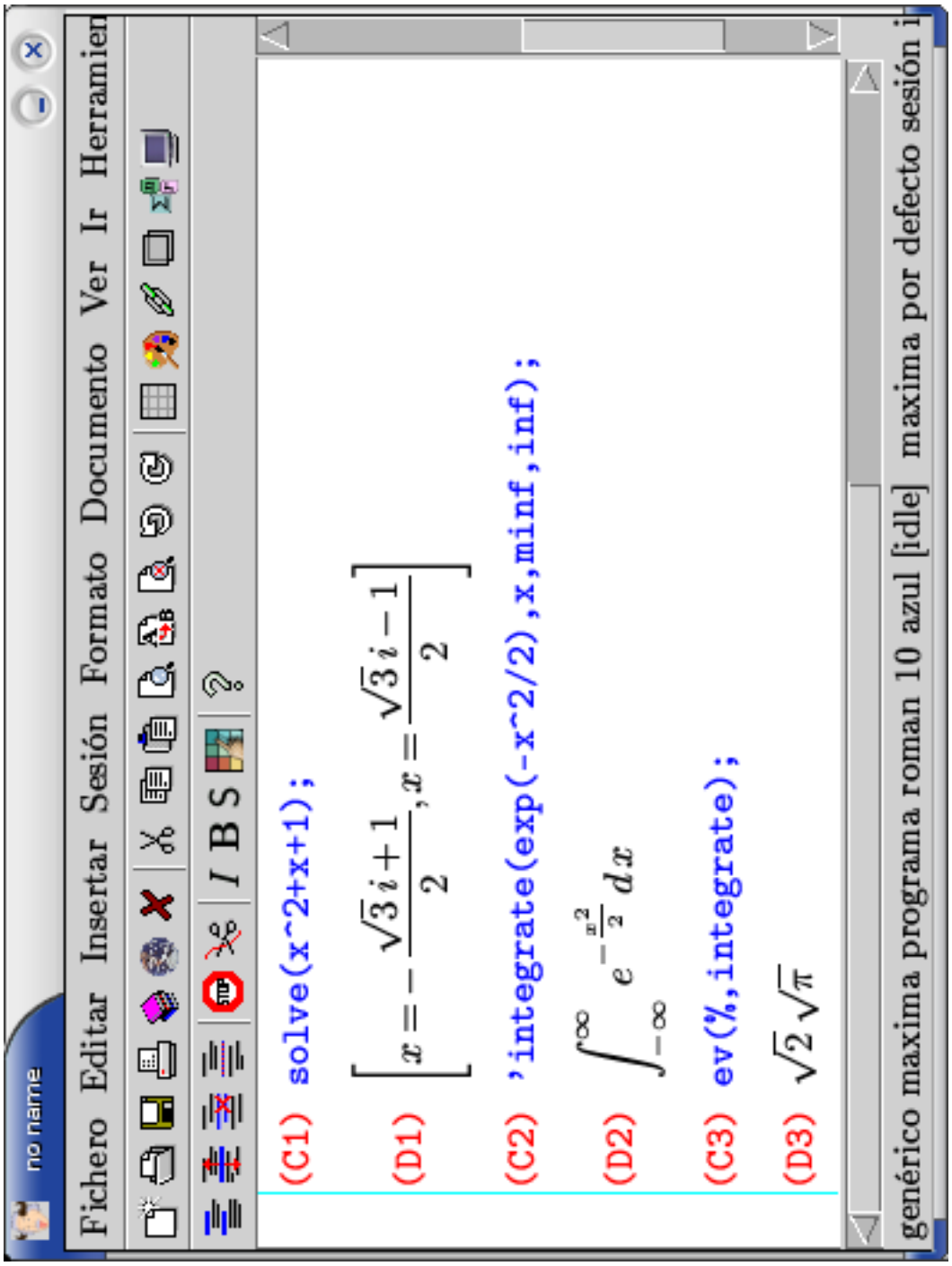


Figura 4: Maxima trabalhando no TeXmacs.

3 Primeira sessão com Maxima

Uma vez que entramos no Maxima, o que veremos primeiro será a seguinte informação:

```
Maxima 5.9.2 http://maxima.sourceforge.net
Using Lisp CLISP 2.33.2 (2004-06-02)
Distributed under the GNU Public License. See the file COPYING.
Dedicated to the memory of William Schelter.
This is a development version of Maxima. The function bug_report()
provides bug reporting information.
(%i1)
```

Trás a informação sobre a licença, GNU-GPL, nos informa sobre a versão que estamos trabalhando e a página *web* do projeto. A continuação, muito importante, aparece o indicador `(%i1)` esperando nossa primeira pergunta. Se queremos calcular uma simples soma teclamos a operação desejada seguida de um ponto e vírgula (`;`) e uma apertamos a tecla Enter

```
(%i1) 45 + 23;
```

ao que Maxima nos responderá

```
(%o1)                          68
(%i2)
```

indicando `(%i2)` que Maxima espera nossa segunda instrução.

```
(%i2) x:34578; y:984003; x*y;
(%o2)                       34578
(%o3)                       984003
(%o4)                    34024855734
(%i5)
```

é conveniente prestar atenção ao fato de que a atribuição de um valor a uma variável se faz com os dois pontos, não com o sinal de igualdade, que se reserva para as equações.

É possível que não desejemos os resultados intermediários que Maxima vá calculando ou, como neste caso, que se deseje apenas as atribuições às variáveis; em tais situações convém fazer uso do delimitador `$`, que não devolve ao console os resultados que forem sendo calculados. Repetindo o cálculo da forma

```
(%i5) x:34578$ y:984003$ x*y;
(%o7)                  34024855734
```

podemos conseguir uma saída mais limpa. As atribuições a variáveis permanecem ativas enquanto durar a sessão com o Maxima, pelo que podemos subtrair as variáveis `x` e `y` anteriores

```
(%i8) x-y;
(%o8)                     - 949425
```

Esta operação deixa um resto; se queremos resolver a equação $x^2 - 3x + 1 = 0$,

```
(%i9) solve(x^2-3*x+1=0,x);
A number was found where a variable was expected -'solve'
 -- an error.  Quitting.  To debug this try debugmode(true);
```

nos devolve uma mensagem de erro, uma vez que onde se supõe que há uma incógnita, o que realmente encontra é um número, neste caso 34578. O problema se resolve esvaziando o conteúdo da variável `x` mediante a função `kill`,

```
(%i10) kill(x)$
(%i11) solve(x^2-3*x+1=0,x);
                   sqrt(5) - 3      sqrt(5) + 3
(%o11)      [x = - -----------, x = -----------]
                        2                2
```

logo as soluções da equação formam o conjunto $\{-\frac{\sqrt{5}-3}{2}, \frac{\sqrt{5}+3}{2}\}$. Vemos que em Maxima a raíz quadrada se calcula com a função `sqrt`.

Os rótulos `(@%ix)` e `(%ox)`, sendo `x` um número inteiro, podem ser utilizadas ao longo de uma sessão a fim de evitar ter que voltar a escrever expressões; assim, se queremos calcular o quociente $\frac{xy}{7}x - y$, com $x = 34578$ y $y = 984003$, podemos aproveitar os resultados `(%o7)` y `(%o8)` e fazer

```
(%i12) %o7/%o8;
                         11341618578
(%o12)                 - -----------
                           316475
```

Os rótulos que indicam as entradas e saídas podem ser modificadas à vontade fazendo uso das variáveis `inchar` e `outchar`,

```
(%i13) inchar;
(%o13)                          %i
(%i14) outchar;
(%o14)                          %o
(%i15) inchar: "entrada";
(%o15)                        entrada
(menda15) outchar: "saída";
(lerenda15)                  "saída"
(menda16) 2+6;
(lerenda16)                     8
(menda17) inchar:"%i"$ outchar:"%o"$
(%i17)
```

No caso de que se pretenda realizar novamente um cálculo já indicado deverá escrever-se dois apóstrofos (`''`), não aspas, junto com o rótulo da instrução desejada

```
(%i18) ''%i9;
(%o18)                      x - 984003
```

Ôbserve que ao usar `x` sem valor numérico, agora o resultado é outro. Quando deseja fazer referência ao último resultado calculado pelo Maxima pode ser mais simples fazer uso do símbolo `%`, de forma que se queremos subtrair `x` da última expressão podemos fazer

```
(%i19) %-x;
(%o19)                       - 984003
```

Certas constantes matemáticas de uso freqüente possuem um símbolo especial em Maxima: a base dos logarítmos naturais (e), o quociente entre o comprimento de uma circunferência e seu diámetro (π), a unidade imaginária ($i = \sqrt{-1}$) e a constante de Euler-Mascheroni ($\gamma = -\int_0^\infty e^{-x} \ln x dx$), se representarão por `%e`, `%pi`, `%i` e `%gamma`, respectivamente.

É muito simples pedir ao Maxima que nos informe sobre alguma função de sua linguagem; a técnica nos servirá para ampliar informação sobre qualquier instrução que se faça referência neste manual, por exemplo,

```
(%i20) describe(sqrt);

 0: isqrt :(maxima.info)Definitions for Operators.
 1: sqrt :Definitions for Operators.
 2: sqrtdispflag :Definitions for Operators.
```

```
Enter space-separated numbers, àll' or 'none': 1

Info from file /usr/local/info/maxima.info:
 - Function: sqrt (<x>)
     The square root of <x>. It is represented internally by
     '<x>^(1/2)'.  See also 'rootscontract'.

     'radexpand' if 'true' will ...roots of factors of a product
     which are powers of n to be ...outside of the radical, e.g.
     'sqrt(16*x^2)' will ...'4*x' only if 'radexpand' is 'true'.

(%o20)                     false
```

Inicialmente mostra um simples menu no qual devemos selecionar que informação concreta desejamos; nesta ocasião se optou por `1` que é o número associado à função que interessava.

A fim de não perder os resultados de uma sessão com Maxima, talvez com o objetivo de continuar o trabalho em próximas jornadas, podemos guardar em um arquivo aqueles valores que nos podem interessar; suponhamos que necesitamos armazenar o conteúdo da variável `y`, tal como o definimos na entrada `(%i5)`, assim como o resultado da saída `(%o11)`, junto com a instrução que a gerou, a entrada `(%i11)`,

```
(%i21) save("minha.sessao",y,resultado=%o11,equacao=%i11)$
```

Note que o primeiro argumento de `save` é o nome do arquivo, junto com seu caminho se se considera necessário, que a variável `y` se escreve tal qual, porém as referências às entradas e saídas devem ser acompanhadas de um nome que as faça posteriormente reconhecíveis.

Por último, a forma correta de abandonar a sessão de Maxima é mediante

```
(%i22) quit();
```

Abramos agora uma nova sessão com Maxima e coloquemos na memória os resultados da anterior sessão,

```
(%i1) load("minha.sessao")$
(%i2) y;
(%o2)                        984003
(%i3) equacao;
                             2
```

```
(%o3)            solve(x  - 3 x + 1 = 0, x)
(%i4) resultado;
                   sqrt(5) - 3      sqrt(5) + 3
(%o4)       [x = - -----------, x = -----------]
                         2                2
```

Se agora quiséssemos recalcular as soluções da equação, modificando-a de maneira que o coeficiente de primeiro grau seja substituído por outro número, simplemente faríamos

```
(%i5) subst(5,3,equacao);
                          2
(%o5)           solve(x  - 5 x + 1 = 0, x)
(%i6) ev(%);
                  sqrt(21) - 5      sqrt(21) + 5
(%o6)      [x = - ------------, x = ------------]
                        2                 2
```

4 Operações aritméticas

Os operadores aritméticos mais comuns são:

+	adição
-	subtração
*	produto
/	divisão
^ ou **	potência

Maxima devolve os resultados de forma exata, sem aproximações decimais; assim temos que, para realizar o cálculo da expressão

$$\left[\left(\frac{3^7}{4+\frac{3}{5}}\right)^{-1}+\frac{7}{9}\right]^3$$

fazermos

```
(%i1) ((3^7/(4+3/5))^(-1)+7/9)^3;
                        620214013952
(%o1)                   -------------
                        1307544150375
```

obtendo o resultado em forma de fração simplificada.

De outra forma podemos solicitar o resultado em forma decimal; por exemplo, se queremos a expressão decimal do último resultado,

```
(%i2) %,numer;
(%o2)                .4743350454163436
```

Maxima pode trabalhar com precissão arbitrária. Para calcular o valor do quociente $\frac{e}{\pi}$ com 100 casas decimais, deveremos especificar primeiro a precissão requerida atribuindo à variável `fpprec` o valor 100 e em seguida realizar o cálculo, solicitando a expressão decimal com uma chamada à função `bfloat`:

```
(%i3) fpprec:100$  bfloat(%e/%pi);
(%o4) 8.652559794322650872177747896460896174287446239085l5#
539454330288948045044570677058631924662516l845173B-1
```

Note que quando um resultado é devolvido no formato `bfloat` se escreve a letra `B` em lugar da clássica `E` para indicar o expoente.

Na instrução `%i3` se estabelece a precisão desejada e logo se transforma o valor simbólico `%e/%pi` mediante a função `bfloat`. Lembre-se que o símbolo `$` serve como delimitador entre instruções.

A fatoração de um número inteiro em fatores primos é um problema difícil de resolver para quantidades grandes; de fato, alguns algorítmos de encriptação (ocultação/embaralhamento de informações) se baseiam nesta dificultade.

```
(%i5) factor(1303948800000);
                       12  3  5  3
(%o5)                 2   3  5  7  11
```

Na seguinte fatoração fazemos uso da variável global `showtime` para saber o tempo que leva executar o cálculo,

```
(%i6) showtime:true$    /* ativamos o contador de tempo */
Êvaluation took 0.00 seconds. (0.00 elapsed) using 80 bytes.
(%i7) factor(2^300-1);
Êvaluation took 349.35 sec. (352.10 elap.) using 21866187.852 KB.
        2  3
(%o7) 3  5  7 11 13 31 41 61 101 151 251 331 601 1201 1321
 1801 4051 8101 63901 100801 268501 10567201 13334701
 1182468601 1133836730401
(%i8) showtime:false$  /* desativamos o contador */
```

O texto que se escreve entre `/*` e `*/` são comentários que Maxima ignora.

Em relação aos números primos, para saber se um número é ou não primo,

```
(%i9) primep(3^56-1);
(%o9)                       false
```

E para solicitar ao Maxima que comprove se um número é par ou impar necessitamos lançar mão das funções `evenp` ou `oddp`, respectivamente,

```
(%i10) evenp(42);
(%o10)                       true
(%i11) oddp(31);
(%o11)                       true
```

Solicitamos ao Maxima que nos faça uma lista com todos os divisores de um número,

```
(%i12) divisors(100);
(%o12)       {1, 2, 4, 5, 10, 20, 25, 50, 100}
```

No que concerne à divisão, pode ser necessário conhecer o quociente inteiro e o resto correspondente; para isso se dispõe das funções `quotient` e `remainder`,

```
(%i13) quotient(5689,214);
(%o13)                       26
(%i14) remainder(5689,214);
(%o14)                      125
```

5 Números complexos

Como já se comentou mais acima, a unidade imaginária $\sqrt{-1}$ se representa em Maxima mediante o símbolo `%i`,

```
(%i1) %i^2;
(%o1)                         - 1
```

Está disponível as operações básicas com números complexos,

```
(%i2) z1:3+5*%i$ z2:1/2-4*%i$  /*z1 y z2*/
(%i4) z1 + z2;     /*adição*/
                                7
(%o4)                        %i + -
                                2
(%i5) z1 - z2;     /*subtração*/
                                 5
(%o5)                       9 %i + -
                                 2
(%i6) z1 * z2;     /*multiplicação*/
                     1
(%o6)               (- - 4 %i) (5 %i + 3)
                     2
(%i7) z1 / z2;     /* divisão */
                           5 %i + 3
(%o7)                      --------
                           1
                           - - 4 %i
                           2
```

É possível que estes dois últimos resultados nos pareçam frustrantes e os desejemos em outro formato, resultado das operações indicadas; com esse objetivo podemos pedir a Maxima que nos devolva `(%o6)` y `(%o7)` em forma cartesiana,

```
(%i8) rectform(%o6);
                          43   19 %i
(%o8)                     -- - -----
                          2      2
(%i9) rectform(%o7);

                          58 %i   74
(%o9)                     ----- - --
                           65     65
```

As funções `realpart` e `imagpart` extraem do número complexo suas partes real e imaginária, respectivamente. Podemos utilizá-las para comprovar que o resultado obtido em `(%o9)` é o correto,

```
(%i10) realpart(%o7);
                               74
(%o10)                       - --
                               65
(%i11) imagpart(%o7);
                              58
(%o11)                        --
                              65
```

Antes havíamos optado pelos resultados em formato cartesiano; mas também é possível solicitar na forma polar,

```
(%i12) polarform(%o7);
                          %i (%pi - atan(29/37))
            2 sqrt(34) %e
(%o12)      ----------------------------------
                          sqrt(65)
(%i13) %,numer;   /*radio y argumento reales*/
                               2.4768181587724474 %i
(%o13)   1.4464811413591583 %e
```

A norma (módulo) de um número complexo se calcula com a função `abs` e admite o argumento tanto em forma cartesiana como em forma polar,

```
(%i14) abs(%o9);
                          2 sqrt(34)
(%o14)                    ----------
                           sqrt(65)
(%i15) abs(%o12);
                          2 sqrt(34)
(%o15)                    ----------
                           sqrt(65)
```

Por último, o conjugado de um número complexo se calcula com a função `conjugate`, porém seu uso requer a colocação na memória do arquivo `eigen`. Como se vê neste exemplo, o resultado dependerá do formato do argumento, cartesiano ou polar,

```
(%i16) load(eigen)$
(%i17) conjugate(%o9);       /*forma cartesiana*/
                          58 %i   74
(%o17)                  - ----- - --
                            65     65
(%i18) conjugate(%o12);      /*forma polar*/
                             %i atan(29/37)
                 2 sqrt(34) %e
(%o18)         - ---------------------------
                           sqrt(65)
```

6 Manipulações algébricas

Sem dúvida uma das capacidades mais importantes do Maxima é sua habilidade para manipular expressões algébricas. Desenvolvamos um exemplo que começa por atribuir à variável `q` uma expressão literal:

```
(%i1) q: (x+3)^5-(x-a)^3+(x+b)^(-1)+(x-1/4)^(-5);
             1            3          5      1
(%o1)      ----- - (x - a)  + (x + 3)  + --------
           x + b                              1 5
                                          (x - -)
                                              4
```

Se observa que em principio Maxima não realiza nenhum cálculo. A função `expand` se encarrega de desenvolver as potências,

```
(%i2) expand(q);
                        1                          1      5
(%o2)  ------------------------------------ + ----- + x
                 4      3      2                 x + b
         5   5 x    5 x    5 x    5 x    1
        x  - ---- + ---- - ---- + --- - ----
              4      8      32     256   1024
        4       3        2        2      2              3
 + 15 x  + 89 x  + 3 a x  + 270 x  - 3 a  x + 405 x + a
 + 243
```

Não obstante é possível que não nos interesse expandir toda a expressão, entre outras coisas para evitar uma resposta grande e difícil de interpretar; em tal caso podemos utilizar `expand` adicionando dois argumentos e fazer da seguinte maneira

```
(%i3) q,expand(3,2);
          1            5    3        2      2        1
(%o3)   ----- + (x + 3)  - x  + 3 a x  - 3 a  x + --------
        x + b                                        1 5
                                                 (x - -)
                                                     4
                                                         3
                                                      + a
```

Com o primeiro argumento indicamos que queremos a expansão de todas aquelas potências com exponente positivo menor ou igual a 3 e das que tendo o exponente negativo sejam menores ou iguais a 2.

Dada uma expressão com valores literais, podemos desejar sustituir alguma letra por outra expressão; por exemplo, se queremos fazer as trocas $a = 2$, $b = 2c$ no último resultado obtido,

```
(%i4) %,a=2,b=2*c;
           1              5    3      2              1
(%o4)   ------- + (x + 3)  - x  + 6 x  - 12 x + -------- + 8
        x + 2 c                                        1 5
                                                 (x - -)
                                                       4
```

Nestes últimos exemplos (`%i3` e `%i4`) apareceram sentenças com elementos separados por vírgula (`,`). Esta é uma forma simplificada de utilizar a função `ev`, que avalia a primera expressão atribuindo os valores que se lhe vão indicando em seguida; por exemplo, `(%i3)` se podia ter escrito da forma `ev(q,expand(3,2))` e `(%i4)` como `ev(%,a=2,b=2*c)`. Outra forma é usar da variante com `ev` esta mais indicada para ser utilizada dentro de expressões mais amplas. Observe-se o resultado seguinte

```
(%i5) 3*x^2 + ev(x^4,x=5);
                              2
(%o5)                      3 x  + 625
```

onde a substitução $x = 5$ foi realizada exclusivamente dentro do ambiente delimitado pela função `ev`.

De forma mais geral, a função `subst` substitui subexpressões inteiras. no seguinte exemplo, introducimos uma expressão algébrica e a seguir substituimos todos os binômios `x+y` pela letra `k`,

```
(%i6) 1/(x+y)-(y+x)/z+(x+y)^2;
                    y + x           2     1
(%o6)             - ----- + (y + x)  + -----
                      z                y + x
(%i7) subst(k,x+y,%);
                           k    2   1
(%o7)                    - - + k  + -
                           z        k
```

Não obstante, o seguinte resultado nos sugere que devemos ser cuidadosos com o uso desta função, já que Maxima algumas vezes não interpretará como subexpressão aquela que para nós é uma subexpressão:

```
(%i8) subst(sqrt(k),x+y,(x+y)^2+(x+y));
(%o8)                    y + x + k
```

Como uma aplicação prática de `subst`, vejamos como podemos utilizá-la para obter o conjugado de um número complexo,

```
(%i9) subst(-%i,%i,a+b*%i);
(%o9)                     a - %i b
```

A operação inversa da expansão é a fatoração. Façamos a expansão e a fatoração sucessivamente de um polinômio para comprovar os resultados,

```
(%i10) expand((a-2)*(b+1)^2*(a+b)^5);
           7      7      2  6        6      6       3  5
(%o10)  a b  - 2 b  + 5 a  b  - 8 a b  - 4 b  + 10 a  b
       2  5         5      5       4  4       2  4         4
 - 10 a  b  - 19 a b  - 2 b  + 10 a  b  - 35 a  b  - 10 a b
      5  3       4  3       3  3       2  3    6  2
 + 5 a  b  + 10 a  b  - 30 a  b  - 20 a  b  + a  b
      5  2       4  2       3  2      6      5         4
 + 8 a  b  - 10 a  b  - 20 a  b  + 2 a  b + a  b - 10 a  b
    6      5
 + a  - 2 a
(%i11) factor(%);
                                  2        5
(%o11)           (a - 2) (b + 1)  (b + a)
```

A função `ratsimp` simplifica qualquer expressão racional, assim como as subexpressões racionais que são argumentos de funções quaisquer. O resultado é retornado como o quociente de dois polinômios. Em algumas ocasiões não é suficiente com uma única execução de `ratsimp`, pelo que será necesário aplicá-la mais vezes, isto é o que faz precisamente a função `fullratsimp`; solidifiquemos isto com um exemplo:

```
(%i12) (x^(a/2)-1)^2*(x^(a/2)+1)^2 / (x^a-1);
```

```
                  a/2     2   a/2     2
                 (x    - 1)  (x    + 1)
(%o12)           ----------------------
                           a
                          x  - 1
(%i13) ratsimp(%);  /* simplificamos uma vez */
                        2 a      a
                       x    - 2 x  + 1
(%o13)                 ---------------
                            a
                           x  - 1
(%i14) ratsimp(%);  /* simplificamos outra vez */
                            a
(%o14)                     x  - 1
(%i15) fullratsimp(%o12);   /* simplificamos tudo de uma vez! */
                            a
(%o15)                     x  - 1
```

Dada uma fração algébrica, podemos obter separadamente o numerador e o denominador,

```
(%i16) fr: (x^3-4*x^2+4*x-2)/(x^2+x+1);
                     3      2
                    x  - 4 x  + 4 x - 2
(%o16)              -------------------
                         2
                        x  + x + 1
(%i17) num(fr);
                     3      2
(%o17)              x  - 4 x  + 4 x - 2
(%i18) denom(fr);
                           2
(%o18)                    x  + x + 1
```

O máximo divisor comum de um conjunto de polinômios se calcula com a função `gcd` e o mínimo multiplo comum com `lcm`

```
(%i19) p1: x^7-4*x^6-7*x^5+8*x^4+11*x^3-4*x^2-5*x;
         7      6      5      4       3      2
(%o19)  x  - 4 x  - 7 x  + 8 x  + 11 x  - 4 x  - 5 x
```

```
(%i20) p2: x^4-2*x^3-4*x^2+2*x+3;
                  4      3      2
(%o20)           x  - 2 x  - 4 x  + 2 x + 3
(%i21) gcd(p1,p2);
                       3    2
(%o21)                 x  + x  - x - 1
(%i22) load(functs)$
(%i23) lcm(p1,p2);
                                    2          3
(%o23)      (x - 5) (x - 3) (x - 1)  x (x + 1)
```

Em `(%i19)` e `(%i20)` definimos os polinômios `p1` y `p2`, a seguir calculamos seu máximo divisor comum (mdc) em `(%i21)` e antes de pedir o mínimo múltiplo comum em `(%i23)` colocamos na memória o arquivo `functs.mac` no qual se encontra definida a função `lcm`. É possível que desejemos dispor do mdc fatorado, pelo que fazemos

```
(%i24) factor(%o21);
                                     2
(%o24)                (x - 1) (x + 1)
```

7 Equacões

Atribuir um rótulo a uma equação é tão simples como fazer

```
(%i1) eq: 3 * x = 1 + x;
(%o1)                    3 x = x + 1
```

A partir daí, aquelas operações nas quais usava-se um rótulo serão realizadas a ambos os membros da igualdade; subtraimos `x` nos dois lados e a seguir dividimos o que resta por `2`,

```
(%i2) %-x;
(%o2)                      2 x = 1
(%i3) %/2;
                                1
(%o3)                       x = -
                                2
```

obtendo desta maneira a solução da equação como se tivéssemos feito manualmente.

Tem-se que dizer que a equação anterior pode ser resolvida de um modo mais imediato,

```
(%i4) solve(eq);
                                1
(%o4)                      [x = -]
                                2
```

A instrução `solve` pode admitir como segundo argumento a incógnita em relação à qual se pretende calcular o valor, o que resultará de utilidade quando na equação aparecerem constantes literais,

```
(%i5) solve((2-a)/x-3=b*x+1/x,x);
                sqrt((4 - 4 a) b + 9) + 3
(%o5)  [x = - ------------------------,
                         2 b
                                  sqrt((4 - 4 a) b + 9) - 3
                              x = ------------------------]
                                           2 b
```

As soluções das equações serão provavelmente utilizadas em cálculos posteriores, pelo que nos interesará poder extraí-las da lista anterior; a seguir tomamos o primeiro resultado calculado por Maxima mediante a função `part` e depois atribuimos à variável `sol` o resultado numérico,

```
(%i6) part(%,1);
                     sqrt((4 - 4 a) b + 9) + 3
(%o6)          x = - -------------------------
                                2 b
(%i7) sol: rhs(%);
                    sqrt((4 - 4 a) b + 9) + 3
(%o7)             - -------------------------
                                2 b
```

A função `rhs` retorna o lado direito da igualdade, enquanto que `lhs` faria a mesma coisa com o lado esquerdo.

É possível resolver equações polinomiais de grau ≤ 4, porém desgraçadamente, como é de se esperar, Maxima não dispõe de um método algébrico que permita resolver equações polinomiais de grau maior que quatro,

```
(%i8) solve(x^5 - 3*x^4 + 2*x^3 -2*x^2 - x + 4 = 0);
                  5      4      3      2
(%o8)       [0 = x  - 3 x  + 2 x  - 2 x  - x + 4]
```

pelo que `solve` devolverá a mesma equação sem resolver.

Maxima dispõe de outra função para resolver equações e sistemas, que em muitas ocasiões será mais útil que `solve`. Vejamos como trata a instrução `algsys` a equação polinomial anterior,

```
(%i9) algsys([x^5 - 3*x^4 + 2*x^3 -2*x^2 - x + 4 = 0],[x]);
(%o9) [[x = 2.478283086356668],
[x = .1150057557117294 - 1.27155810694299 %i],
[x = 1.27155810694299 %i + .1150057557117294],
[x = - .8598396689940523], [x = 1.151545166402536]]
```

Como se vê, ao não ser capaz de resolvê-la algebricamente, nos brinda a oportunidade de conhecer uma aproximação numérica da solução. A função `algsys` reconhece a variável global `realonly`, que quando toma o valor `true`, fará com que se ignorem as soluções complexas,

```
(%i10) realonly:true$
(%i11) ''%i9;  /* recalcula a entrada %i9 */
(%o12) [[x = 2.478283086356668], [x = 1.151545166402536],
                                  [x = - .8598396689940523]]
(%i13) realonly:false$  /* lhe devolvemos o valor padrão */
```

Tratando-se de equações polinomiais, para forçar o cálculo numérico de suas raízes se pode fazer uso também da instrução `allroots`, que obtém tanto as raízes reais como as raízes complexas,

```
(%i14) allroots(x^5 - 3*x^4 + 2*x^3 -2*x^2 - x + 4 = 0);
(%o14) [x = 1.151545173424091, x = - .8598397137271315,
x = 1.27155810694299 %i + .1150057557117294,
x = .1150057557117294 - 1.27155810694299 %i,
x = 2.478283028879582]
```

donde se recordará que o símbolo `%i` representa a unidade imaginária $i = \sqrt{-1}$.

Vemos a seguir como resolver um sistema linear de equações mediante `algsys`. Sejam as equações

$$\begin{cases} 2x - 4y + 2z = -2 \\ \frac{1}{3}x + 2y + 9z = x+y \\ -4x + \sqrt{2}y + z = 3y \end{cases}$$

se fará

```
(%i15) algsys([ 2 * x - 4 * y + 2 * z = -2,
               1/3* x + 2 * y + 9 * z = x + y,
               -4 * x + sqrt(2) * y + z = 3 * y],
         [x,y,z]);
               27 sqrt(2) - 84                106
(%o15) [[x = - ---------------, y = - ----------------,
               29 sqrt(2) - 314        29 sqrt(2) - 314
                                       575 sqrt(2) - 2884
                                 z = - --------------------]]
                                       9106 sqrt(2) - 50139
```

note-se que as equações estão entre colchetes e se separaram por vírgulas e que o mesmo se deve fazer com os nomes das incógnitas.

Também a função `algsys` permite a resolução de sistemas de equações não lineares como

$$\begin{cases} 3x^2 - y = 2y^2 + 4 \\ 5x + 7y = 1 \end{cases}$$

Podemos proceder como se indica a seguir,

```
(%i16) algsys([3*x^2-y=2*y^2+4, 5*x+7*y=1],[x,y]);
```

```
                7 sqrt(1685) + 55
(%o16) [[x = - -----------------,
                       194
    5 sqrt(5) sqrt(337) + 67
y = ------------------------],
               194
     7 sqrt(1685) - 55        5 sqrt(5) sqrt(337) - 67
[x = -----------------, y = - ------------------------]]
            194                          194
```

cujo resultado é uma lista com dois pares ordenados, soluções ambos do sistema proposto. Uma maneira alternativa de proceder neste caso podería consistir em solicitar primeiro ao Maxima que nos eliminasse uma das variáveis e resolver a seguir para a outra, tal como indica o seguinte código,

```
(%i17) eliminate([3*x^2-y=2*y^2+4, 5*x+7*y=1],[y]);
                           2
(%o17)               [97 x  + 55 x - 205]
(%i18) algsys(%,[x]);
             7 sqrt(1685) - 55          7 sqrt(1685) + 55
(%o18) [[x = -----------------], [x = - -----------------]]
                     194                          194
```

Em `(%i17)` pedimos que nos elimine a incógnita *y*, obtendo como resultado uma equação de segundo grau, a qual se resolve a seguir. Note-se que na expressão `(%o17)` falta uma igualdade, o que haverá de interpretar-se como igualada a zero, sendo equivalente a $97x^2 + 55x - 205 = 0$.

Já para terminar, resolvamos uma equação indeterminada obtendo a solução em forma paramétrica,

```
(%i19) algsys([3*x^2-5*y=x],[x,y]);
                                    2
                               3 %r1  - %r1
(%o19)          [[x = %r1, y = ------------]]
                                    5
(%i20) %, %r1:1;
                                    2
(%o20)                 [[x = 1, y = -]]
                                    5
```

Maxima nomeia os parm̂etros seguindo o esquema `%Rx`, sendo `x` um número inteiro positivo. Na entrada `%i20` pedimos que em `%o19` se substitua o parâmetro pela unidade.

8 Matrizes

A definição de uma matriz é extremadamente simples em Maxima,

```
(%i1) m1: matrix([3,4,0],[6,0,-2],[0,6,a]);
                       [ 3  4   0  ]
                       [           ]
(%o1)                  [ 6  0  - 2 ]
                       [           ]
                       [ 0  6   a  ]
```

Também é possível definir uma matriz de forma interativa tal como mostra o seguinte exemplo,

```
(%i2) entermatrix(2,3);
Row 1 Column 1:
4/7;
Row 1 Column 2:
0;
Row 1 Column 3:
%pi;
Row 2 Column 1:
sqrt(2);
Row 2 Column 2:
log(3);
Row 2 Column 3:
-9;

Matrix entered.
                  [    4                 ]
                  [    -       0     %pi ]
(%o2)             [    7                 ]
                  [                      ]
                  [ sqrt(2)  log(3)  - 9 ]
```

Existe um terceiro método para construir matrizes que é útil quando o elemento (i, j)-ésimo da mesma é função de sua posição dentro da matriz. A seguir, se fixa em primeiro lugar a regra que permite definir um elemento qualquer e logo com base nela se constrói uma matriz de dimensões 2×5

```
(%i3) a[i,j]:=i+j$
```

```
(%i4) genmatrix(a,2,5);
                      [ 2  3  4  5  6 ]
(%o4)                 [                ]
                      [ 3  4  5  6  7 ]
```

Observe-se que o símbolo de atribuição para o elemento genérico é `:=`.

Podemos acessar aos diferentes elementos da matriz fazendo referência a seus subíndices, indicando primeiro a linha e depois a coluna:

```
(%i5) m1[3,1];
(%o5)                          0
```

Se pode extrair uma submatriz com a função `submatrix`, tendo em conta que os inteiros que precedem ao nome da matriz original são as linha a eliminar e os que se colocam por detrás indicam as colunas que não interesam; no seguinte exemplo, queremos a submatriz que nos resta de `m1` depois de extrair a primeira linha e a segunda coluna,

```
(%i5) submatrix(1,m1,2);
                           [ 6  - 2 ]
(%o5)                      [        ]
                           [ 0   a  ]
```

Outro exemplo é o seguinte,

```
(%i6) submatrix(1,2,m1,3);
(%o6)                        [ 0  6 ]
```

na qual eliminamos as duas primeiras linhas e a última coluna.

Da mesma forma que se extrai submatrizes, é possível adicionar linhas e colunas a uma matriz dada; por exemplo,

```
(%i7) addrow(m1,[1,1,1],[2,2,2]);
```

```
                        [ 3  4   0  ]
                        [           ]
                        [ 6  0  - 2 ]
                        [           ]
(%o7)                   [ 0  6   a  ]
                        [           ]
                        [ 1  1   1  ]
                        [           ]
                        [ 2  2   2  ]
(%i8) addcol(%,[7,7,7,7,7]);
                       [ 3  4   0   7 ]
                       [               ]
                       [ 6  0  - 2  7 ]
                       [               ]
(%o8)                  [ 0  6   a   7 ]
                       [               ]
                       [ 1  1   1   7 ]
                       [               ]
                       [ 2  2   2   7 ]
```

A matriz identidade é mais fácil construí-la mediante a função `ident`,

```
(%i9) ident(3);
                         [ 1  0  0 ]
                         [         ]
(%o9)                    [ 0  1  0 ]
                         [         ]
                         [ 0  0  1 ]
```

e a matriz com todos seus elementos iguais a zero,

```
(%i10) zeromatrix(2,4);
                       [ 0  0  0  0 ]
(%o10)                 [            ]
                       [ 0  0  0  0 ]
```

Também, uma matriz diagonal com todos os elementos da diagonal principal iguais pode construir-se com uma chamada à função `diagmatrix`,

```
(%i11) diagmatrix(4,%e);
```

```
                    [ %e  0   0   0  ]
                    [                ]
                    [ 0   %e  0   0  ]
(%o11)              [                ]
                    [ 0   0   %e  0  ]
                    [                ]
                    [ 0   0   0   %e ]
```

Em todo caso, deve ter-se cuidado em que se a matriz não for construída de forma apropriada, Maxima não a reconhece como tal. Para saber se uma expressão é reconhecida como uma matriz se utiliza a função `matrixp`; a seguinte sequência permite aclarar o que se pretende dizer,

```
(%i12) matrix([[1,2,3],[4,5,6]]);  /* construçào correta */
(%o12)            [ [1, 2, 3]  [4, 5, 6] ]
(%i13) matrixp(%);  /* é a anterior realmente uma matriz? */
(%o13)                       TRUE
(%i14) [[7,8,9],[0,1,2]];    /* outra matriz */
(%o14)             [[7, 8, 9], [0, 1, 2]]
(%i15) matrixp(%);   /* sera uma matriz? */
(%o15)                      FALSE
```

Casos particulares de submatrizes são as matrizes linha e as matrizes coluna; os exemplos se explicam por si só:

```
(%i16) col(m1,3);
                          [  0  ]
                          [     ]
(%o16)                    [ - 2 ]
                          [     ]
                          [  a  ]
(%i17) row(m1,2);
(%o17)                [ 6  0  - 2 ]
```

Com as matrizes se pode realizar múltiplas operações. Começemos pelo cálculo da potência de uma matriz:

```
(%i18) m1^^2;
```

```
                    [ 33  12     - 8    ]
                    [                   ]
(%o18)              [ 18  12    - 2 a   ]
                    [                   ]
                    [           2       ]
                    [ 36  6 a   a  - 12 ]
```

Note-se que se utiliza duas vezes o símbolo ^ antes do exponente; em caso de escrevê-lo uma só vez se calcularíam as potências de cada um dos elementos da matriz independentemente, como se indica no seguinte exemplo,

```
(%i19) m2:m1^2;
                       [ 9   16  0  ]
                       [            ]
(%o19)                 [ 36  0   4  ]
                       [            ]
                       [          2 ]
                       [ 0   36  a  ]
```

Para a adição, subtração e produto matriciais se utilizam os operadores +, − e ., respectivamente,

```
(%i20) m1+m2;
                      [ 12  20    0    ]
                      [                ]
(%o20)                [ 42  0     2    ]
                      [                ]
                      [          2     ]
                      [ 0   42  a  + a ]
(%i21) m1-m2;
                    [ - 6   - 12    0    ]
                    [                    ]
(%o21)              [ - 30   0     - 6   ]
                    [                    ]
                    [                  2 ]
                    [  0    - 30  a - a  ]
(%i22) m1.m2;
```

```
                    [ 171   48     16    ]
                    [                    ]
                    [                 2  ]
(%o22)              [ 54    24   - 2 a   ]
                    [                    ]
                    [             3      ]
                    [ 216  36 a   a  + 24 ]
```

Sem impedimentos, tanto o produto elemento a elemento de duas matrizes, como a multiplicação por um escalar se realizam mediante o operador `*`, como indicam os seguintes dois exemplos,

```
(%i23) m1*m2;
                      [ 27   64    0  ]
                      [               ]
(%o23)                [ 216   0   - 8 ]
                      [               ]
                      [            3  ]
                      [  0   216  a   ]
(%i24) 4*m1;
                       [ 12  16   0  ]
                       [             ]
(%o24)                 [ 24  0   - 8 ]
                       [             ]
                       [ 0   24  4 a ]
```

Outros cálculos freqüentes com matrizes são a transposição, o determinante, a inversão, o polinômio característico, assim como os autovalores (eigenvalues) e autovetores (eigenvectors) próprios; para todos eles existe funções em Maxima:

```
(%i25) transpose(m1);    /*a transposta*/
                        [ 3   6   0 ]
                        [           ]
(%o25)                  [ 4   0   6 ]
                        [           ]
                        [ 0  - 2  a ]
(%i26) determinant(m1);  /*o determinante*/
(%o26)                  36 - 24 a
(%i27) invert(m1);       /*a inversa*/
```

```
         [     12            4 a           8     ]
         [  ---------   - ---------  - --------- ]
         [  36 - 24 a     36 - 24 a    36 - 24 a ]
         [                                       ]
         [      6 a         3 a           6      ]
(%o27)   [ - ---------   ---------    ---------  ]
         [   36 - 24 a   36 - 24 a    36 - 24 a  ]
         [                                       ]
         [     36            18           24     ]
         [  ---------   - ---------  - --------- ]
         [  36 - 24 a     36 - 24 a    36 - 24 a ]
(%i28) invert(m1),detout;  /*a inversa, com o determinante fora*/
                  [  12    - 4 a  - 8  ]
                  [                    ]
                  [ - 6 a   3 a    6   ]
                  [                    ]
                  [  36    - 18   - 24 ]
(%o28)            ----------------------
                        36 - 24 a
(%i29) charpoly(m1,x); /*pol.  caract. com variável x*/
(%o29)     (3 - x) (12 - (a - x) x) - 24 (a - x)
(%i30) expand(%);     /*pol. caract. expandido*/
          3      2      2
(%o30) - x  + a x  + 3 x  - 3 a x + 12 x - 24 a + 36
```

Vamos supor agora que a variável a em (%o30) vale zero e calculemos os valores próprios (autovalores) da matriz,

```
(%i31) m1,a:0;
                      [ 3  4   0  ]
                      [           ]
(%o31)                [ 6  0  - 2 ]
                      [           ]
                      [ 0  6   0  ]
(%i32) eigenvalues(%o31);
           sqrt(15) %i + 3  sqrt(15) %i - 3
(%o32) [[- ---------------, ---------------, 6], [1, 1, 1]]
                  2                2
```

O resultado que se obtém é uma lista formada por duas sublistas, na primeira se encontram os valores próprios, que neste caso são $\lambda_1 = -\frac{3}{2} - \frac{\sqrt{15}}{2}i$, $\lambda_2 = -\frac{3}{2} + \frac{\sqrt{15}}{2}i$

e $\lambda_3 = 6$, enquanto que na segunda sublista se encontram as multiplicidades de cada uma das λ_i.

Para o cálculo dos vetores próprios (autovetores),

```
(%i33) eigenvectors(%o31);

            sqrt(15) %i + 3  sqrt(15) %i - 3
(%o33) [[[- ---------------, ---------------, 6],
                   2                2

                  sqrt(15) %i + 9
[1, 1, 1]], [1, - ---------------,
                         8

  3 sqrt(3) sqrt(5) %i - 21
- -------------------------],
              8

    sqrt(15) %i - 9  3 sqrt(3) sqrt(5) %i + 21       3  3
[1, ---------------, -------------------------], [1, -, -]]
           8                     8                   4  4
```

O que se obtém é, em primeiro lugar, os valores próprios (autovalores) junto com suas multiplicidades, o mesmo resultado que se obtém com a função `eigenvalues`; a seguir os vetores próprios (autovetores) da matriz associados a cada um dos valores próprios (autovalores). As vezes interessa que os vetores sejam unitários, de norma (módulo) 1, para o que será de utilidade a função `uniteigenvectors`, que se encontra definida no pacote `eigen.lisp`, o que significa que antes de fazer uso dele teremos que executar o comando `load(eigen)`.

Outros cálculos possíveis com matrizes,

```
(%i34) minor(%o31,2,1);  /* ou menor complementar de um elemento */
                          [ 4  0 ]
(%o34)                    [      ]
                          [ 6  0 ]
(%i35) rank(%o31);   /* ou posto da matriz*/
(%o35)                        3
(%i36) matrixmap(sin,%o31);
```

```
                [ sin(3)  sin(4)     0     ]
                [                          ]
(%o36)          [ sin(6)    0     - sin(2) ]
                [                          ]
                [   0     sin(6)     0     ]
```

Neste último exemplo, aplicamos a função seno a todos os elementos da matriz.

9 Funções matemáticas

Em Maxima estão definidas um grande número de funções, algumas das quais se apresentam na Figura 5. A seguir se desenvolvem alguns exemplos sobre seu uso.

As funções exponenciais sempre são positivas, independentemente do argumento *x*,

```
(%i1) limit(1/(x-1),x,1,minus);
(%o1)                        minf
(%i2) abs(3^-x);
                              1
(%o2)                         --
                               x
                              3
(%i3) signum(3^-x);
(%o3)                         1
```

A função `genfact(m,n,p)` é o fatorial generalizado, de forma que `genfact(m,m,1)` coincide com *m*!,

```
(%i4) genfact(5,5,1)-5!;
(%o4)                         0
```

e `genfact(m,m/2,2)` é igual a *m*!!,

```
(%i5) genfact(5,5/2,2)-5!!;
(%o5)                         0
```

Maxima sempre devolve resultados exatos, que podemos solicitar em formato decimal,

```
(%i6) asin(1);
                              %pi
(%o6)                         ---
                               2
(%i7) %,numer;
(%o7)                1.570796326794897
```

Recordemos que o formato decimal podemos pedir com precissão arbitrária,

```
(%i8) fpprec:50$ bfloat(%o6);
(%o9) 1.5707963267948966192313216916397514420985846996876B0
```

`abs(x)`	$\text{abs}(x)$
`min(x1,x2,...)`	$\min(x_1,x_2,\ldots)$
`max(x1,x2,...)`	$\max(x_1,x_2,\ldots)$
`signum(x)`	$\text{sinal}(x)=\begin{cases}-1 & \text{si } x<0\\ 0 & \text{si } x=0\\ 1 & \text{si } x>0\end{cases}$
`x!`	$x!$
`x!!`	$x!!$
`binomial(m,n)`	$\binom{m}{n}=\frac{m(m-1)\ldots[m-(n-1)]}{n!}$
`genfact(m,n,p)`	$m(m-p)(m-2p)\ldots[m-(n-1)p]$
`sqrt(x)`	$\sqrt{x}$
`exp(x)`	e^x
`log(x)`	$\ln(x)$
`sin(x)`	$\sin(x)$
`cos(x)`	$\cos(x)$
`tan(x)`	$\tan(x)$
`csc(x)`	$\csc(x)$
`sec(x)`	$\sec(x)$
`cot(x)`	$\cot(x)$
`asin(x)`	$\arcsin(x)$
`acos(x)`	$\arccos(x)$
`atan(x)`	$\arctan(x)$
`atan2(x,y)`	$\arctan(\frac{x}{y})\in(-\pi,\pi)$
`sinh(x)`	$\sinh(x)=\frac{1}{2}(e^x-e^{-x})$
`cosh(x)`	$\cosh(x)=\frac{1}{2}(e^x+e^{-x})$
`tanh(x)`	$\tanh(x)=\frac{\sinh(x)}{\cosh(x)}$
`asinh(x)`	$\text{arcsinh}(x)$
`acosh(x)`	$\text{arccosh}(x)$
`atanh(x)`	$\text{arctanh}(x)$
`gamma(x)`	$\Gamma(x)=\int_0^\infty e^{-u}u^{x-1}du,\forall x>0$
`beta(x,y)`	$\beta(x,y)=\frac{\Gamma(x)\Gamma(y)}{\Gamma(x+y)}$
`erf(x)`	$\text{erf}(x)=\int_0^x\frac{2}{\sqrt{\pi}}e^{-u^2}du$

Figura 5: Algumas funções do Maxima.

A função *error* está relacionada com a função de distribuição da variável aleatória normal $X \sim \mathcal{N}(0,1)$ da forma

$$\Phi(x) = \Pr(X \leq x) = \frac{1}{2} + \frac{1}{2}\text{erf}\left(\frac{x}{\sqrt{2}}\right),$$

pelo que a probabilidade de que esta variável tome um valor menor que 1.5 é da ordem de

```
(%i10) 0.5+0.5*erf(1.5/sqrt(2)),numer;
(%o10)               0.9331927987311419
```

Uma forma mais elegante de fazer o anterior é definir nossa própria função de distribução a partir da de *error*, para o que se faz uso do símbolo `:=`,

```
(%i11) F(x):=1/2+erf(x/sqrt(2))/2$
(%i12) F(1.5),numer;
(%o12)               0.9331927987311419
```

Não terminam aqui as funções do Maxima; junto às já expostas teríamos que incluir as funções de Airy, elípticas e de Bessel, sobre as quais se poderá obter mais informação executando a instrucção `describe` e utilizando como argumento `airy`, `elliptic` ou `bessel`, conforme o caso. Por exemplo,

```
(%i13) describe(airy);

 0: airy :(maxima.info)Definitions for Special Functions.
 1: airy_ai :Definitions for Special Functions.
 2: airy_bi :Definitions for Special Functions.
 3: airy_dai :Definitions for Special Functions.
 4: airy_dbi :Definitions for Special Functions.
Enter space-separated numbers, àll' or 'none': 1

Info from file /usr/local/info/maxima.info:
 - Function: airy_ai (<x>)
     The Airy function Ai, as defined in Abramowitz and Stegun,
     Handbook of Mathematical Functions, Section 10.4.

     The Airy equation 'diff (y(x), x, 2) - x y(x) = 0' has two
     linearly independent solutions, 'y = Ai(x)' and 'y = Bi(x)'.  The
     derivative 'diff (airy_ai(x), x)' is àiry_dai(x)'.
```

```
     If the argument `x' is a real or complex floating point number,
     the numerical value of àiry_ai' is returned when possible.

     See also 'airy_bi', 'airy_dai', 'airy_dbi'.
(%o13)                     false
```

10 Limites

Sem mais preâmbulos, vejamos alguns exemplos de como calcular limites com a assistência do Maxima. Em primeiro lugar vemos que é possível fazer com que a variável se aproxime ao infinito ($x \to \infty$) fazendo uso do símbolo `inf`, ou que se aproxime ao menos infinito ($x \to -\infty$) fazendo uso de `minf`,

```
(%i1) limit(1/sqrt(x),x,inf);
(%o1)                         0
(%i2) limit((exp(x)-exp(-x))/(exp(x)+exp(-x)),x,minf);
(%o2)                        - 1
```

que nos permite calcular $\lim_{x\to\infty} \frac{1}{\sqrt{x}}$ e $\lim_{x\to-\infty} \frac{e^x-e^{-x}}{e^x+e^{-x}}$, respectivamente.

Os seguintes exemplos mostram limites nos quais a variável x se aproxima de pontos de descontinuidade,

```
(%i3) limit((x^2-x/3-2/3)/(5*x^2-11*x+6),x,1);
                                5
(%o3)                         - -
                                3
(%i4) limit(1/(x-1)^2,x,1);
(%o4)                        inf
```

de onde obtivemos os resultados

$$\lim_{x\to 1} \frac{x^2-\frac{x}{3}-\frac{2}{3}}{5x^2-11x+6} = -\frac{5}{3}$$

e

$$\lim_{x\to 1} \frac{1}{(x-1)^2} = \infty.$$

Sem impedimentos, certos limites não podem ser resolvidos sem informação adicional, tal é o caso de $\lim_{x\to 1} \frac{1}{x-1}$, para o qual podemos fazer

```
(%i5) limit(1/(x-1),x,1);
(%o5)                        und
```

de onde Maxima nos responde com o símbolo `und` de *undefined* ou indefinido. Em tais situações podemos indicar ao assistente que a variável x se aproxima de 1 pela dereita ($x \to 1^+$) ou pela esquerda ($x \to 1^-$),

```
(%i6) limit(1/(x-1),x,1,plus);
(%o6)                        inf
(%i7) limit(1/(x-1),x,1,minus);
(%o7)                        minf
```

11 Derivadas

Maxima controla o cálculo de derivadas mediante a instrução `diff`. A seguir se apresentam alguns exemplos sobre seu uso,

```
(%i1) diff(x^log(a*x),x);
                log(a x)  log(a x)   log(x)
(%o1)          x         (-------- + ------)
                                x         x
(%i2) diff(x^log(a*x),x,2); /*derivada segunda*/
       log(a x)  log(a x)   log(x) 2
(%o2) x         (-------- + ------)
                    x         x
                         log(a x)    log(a x)   log(x)   2
                   + x         (- -------- - ------ + --)
                                        2         2      2
                                       x         x      x
(%i3) factor(%);
       log(a x) - 2     2
(%o3) x             (log (a x) + 2 log(x) log(a x)
                                          2
                         - log(a x) + log (x) - log(x) + 2)
```

de onde se calculadou a primeira e segunda derivadas da função $y = x^{\ln(ax)}$. Note-se que na entrada `(%i3)` se pediu ao asistente que simplificasse a saída `(%o2)`.

Também se pode calcular derivadas parciais, tal como se mostra a seguir,

```
(%i4) diff(x*cos(y)+y*sin(x*y),x,1,y,2);
                          2  2
(%o4) - 4 x y sin(x y) - x  y  cos(x y) + 2 cos(x y)
                                                    - cos(y)
(%i5) diff(exp(x)*sin(y)*tan(z),x,3,y,5,z,2);
                   x           2
(%o5)           2 %e  cos(y) sec (z) tan(z)
```

de onde se obteve os resultados

$$\frac{\vartheta^3}{\vartheta x \vartheta y^2}\left(x\cos(y)+y\sin(xy)\right) = -4xy\sin(xy) - x^2y^2\cos(xy) + 2\cos(xy) - \cos(y)$$

e

$$\frac{\vartheta^{10}}{\vartheta x^3 \vartheta y^5 \vartheta z^2}\left(e^x \sin(y)\tan(z)\right) = 2e^x\cos(y)\sec^2(z)\tan(z).$$

Maxima também nos pode ajudar à hora de aplicar a regra da cadeia no cálculo de derivadas de funções vetoriais com variável também vetorial. Suponha-se que certa variável *z* depende de outras duas *x* e *y*, às quais por sua vez dependem de *u* e *v*. Vejamos como se aplica a regra da cadeia para obter $\frac{\vartheta z}{\vartheta v}$, $\frac{\vartheta z^2}{\vartheta y \vartheta v}$ ou $\frac{\vartheta z^2}{\vartheta u \vartheta v}$.

```
(%i6) depends(z,[x,y],[x,y],[u,v]);
(%o6)           [z(x, y), x(u, v), y(u, v)]
(%i7) diff(z,v,1);
                       dy dz   dx dz
(%o7)                  -- -- + -- --
                       dv dy   dv dx
(%i8) diff(z,y,1,v,1);
                         2         2
                     dy d z   dx  d z
(%o8)                -- --- + -- -----
                     dv   2   dv dx dy
                        dy
(%i9) diff(z,u,1,v,1);
               2         2        2
      dy  dy d z   dx  d z      d y  dz
(%o9)  -- (-- --- + -- -----) + ----- --
      du  dv   2   dv dx dy    du dv dy
             dy
                                  2         2        2
                            dx  dx d z   dy  d z      d x  dz
                          + -- (-- --- + -- -----) + ----- --
                            du  dv   2   dv dx dy    du dv dx
                                   dx
```

A qualquer momento podemos solicitar ao Maxima que nos recorde o quadro de dependências,

```
(%i10) dependencies;
(%o10)          [z(x, y), x(u, v), y(u, v)]
```

ou também podemos eliminar dependências,

```
(%i11) remove(x,dependency);
(%o11)                       done
(%i12) dependencies;
(%o12)               [z(x, y), y(u, v)]
```

```
(%i13) diff(z,y,1,v,1);
                                 2
                            dy d z
(%o13)                      -- ---
                            dv   2
                               dy
```

Vejamos como Maxima deriva funções definidas implícitamente. No seguinte exemplo, para evitar que `y` seja considerada uma constante, a declararemos dependente de `x`,

```
(%i14) depends(y,x)$
(%i15) diff(x^2+y^3=2*x*y,x);
                    2 dy              dy
(%o15)          3 y  -- + 2 x = 2 x -- + 2 y
                      dx              dx
```

Quando se solicita o cálculo de uma derivada sem especificar a variável em relação à qual se deriva, Maxima utilizará o símbolo `del` para representar as diferencias,

```
(%i16) diff(x^2);
(%o16)                      2 x del(x)
```

O que se interpretará como $2xdx$. Se na expressão a derivar existe mais de uma variável, teremos diferencias para todas,

```
(%i17) diff(x^2+y^3=2*x*y);
          2              2 dy
(%o17) 3 y  del(y) + (3 y  -- + 2 x) del(x) =
                          dx
                                                dy
                          2 x del(y) + (2 x -- + 2 y) del(x)
                                                dx
```

Recorde-se que durante este cálculo estava todavia ativa a dependência declarada na entrada `(%i14)`.

Finalmente, para acabar esta seção, façamos referência ao desenvolvimento de Taylor de terceiro grau da função

$$y = \frac{x \ln x}{x^2 - 1}$$

em torno de $x = 1$,

```
(%i18) taylor((x*log(x))/(x^2-1),x,1,3);
                          2          3
              1   (x - 1)    (x - 1)
(%o18)/T/     - - -------- + -------- + . . .
              2      12         12
(%i19) expand(%);
                      3    2
                     x    x    5 x   1
(%o19)               -- - -- + --- + -
                     12   3    12    3
```

A seguir um exemplo de desenvolvimento multivariante da função $y = \exp(x^2 \sin(xy))$ ao redor do ponto $(2,0)$ até grau 2 com relação a cada variável,

```
(%i20) taylor(exp(x^2*sin(x*y)),[x,2,2],[y,0,2]);
                         2
(%o20)/T/ 1 + 8 y + 32 y  + . . .
               2
 + (12 y + 96 y  + . . .) (x - 2)
              2                  2
 + (6 y + 120 y  + . . .) (x - 2)  + . . .
(%i21) expand(%);
           2  2          2        2      2
(%o21) 120 x  y  - 384 x y  + 320 y  + 6 x  y - 12 x y + 8 y
                                                          + 1
```

12 Integrais

A função do Maxima que controla o cálculo de integrais é `integrate`, tanto para as definidas como para as indefinidas; comecemos por estas últimas,

```
(%i1) integrate(cos(x)^3/sin(x)^4,x);
                              2
                       3 sin (x) - 1
(%o1)                  -------------
                              3
                         3 sin (x)
(%i2) integrate(a[3]*x^3+a[2]*x^2+a[1]*x+a[0],x);
                    4       3       2
                a  x    a  x    a  x
                 3       2       1
(%o2)           ----- + ----- + ----- + a  x
                  4       3       2      0
```

que nos devolve os resultados

$$\int \frac{\cos^3 x}{\sin^4 x} dx = \frac{3\sin^2 x - 1}{3\sin^3 x}$$

e

$$\int (a_3 x^3 + a_2 x^2 + a_1 x + a_0) dx = \frac{a_3 x^4}{4} + \frac{a_2 x^3}{3} + \frac{a_1 x^2}{2} + a_0 x.$$

Além disso, este último exemplo nos oferece a oportunidade de ver como escrever coeficientes com subíndices.

Agora um par de exemplos sobre a integral definida,

```
(%i3) integrate(2*x/((x-1)*(x+2)),x,3,5);
           2 log(7) + log(4)   2 log(5) + log(2)
(%o3)   2 (---------------- - ----------------)
                  3                   3
(%i4) %,numer;   /*aproximação decimal*/
(%o4)                0.91072776920158
(%i5) integrate(asin(x),x,0,u);
Is  u  positive, negative, or zero?

positive;
                                      2
(%o5)           u asin(u) + sqrt(1 - u ) - 1
```

isto é,

$$\int_3^5 \frac{2x}{(x-1)(x+2)}dx \approx 0.91072776920158$$

e

$$\int_0^u \arcsin(x)dx = u\arcsin(u) + \sqrt{1-u^2} - 1, \forall u > 0.$$

Note-se neste último exemplo como antes de dar o resultado Maxima pergunta se u é positivo, negativo ou nulo; respondemos escrevendo `positive;` (ponto e vírgula incluído) obtemos finalmente o resultado.

A transformada de Laplace de uma função $f(x)$ se define como a integral

$$L(p) = \int_0^\infty f(x)e^{-px}dx,$$

sendo p um número complexo. Assim, a transformada de Laplace de $f(x) = ke^{-kx}$ é

```
(%i6) laplace(k*exp(-k*x),x,p);
                              k
(%o6)                       -----
                            p + k
```

e calculando a transformada inversa voltamos ao ponto de partida,

```
(%i7) ilt(%,p,x);
                                - k x
(%o7)                     k %e
```

A transformada de Fourier de uma função se reduz à de Laplace quando o argumento $p = -it$, sendo i a unidade imaginária e $t \in \mathbb{R}$,

$$F(t) = \int_0^\infty f(x)e^{itx}dx.$$

Desta maneira, a transformada de Fourier de $f(x) = ke^{-kx}$ é

```
(%i8) laplace(k*exp(-k*x),x,-%i*t);
                                k
(%o8)                       --------
                            k - %i t
```

Note-se que se $x > 0$, a $f(x)$ anterior é precisamente a função de densidade de uma variável aleatória exponencial de parâmetro k, pelo que este último resultado coincide precisamente com a função característica desta mesma distribuição. Tenha-se em conta que Maxima calcula a transformada de Laplace integrando na semi-reta positiva, pelo que o cálculo anterior não é válido para calcular funções características de variáveis aleatórias que possam tomar valores negativos, como por exemplo a distribuição normal. Uma possível solução a esta situação é integrar diretamente para calcular a função característica da distribuição normal $X \sim \mathcal{N}(0,1)$,

```
(%i9) integrate(1/sqrt(2*%pi)*exp(-x^2/2)*exp(%i*t*x),x,minf,inf);
                                2  2
                              %i  t
                              ------
                                2
(%o9)                        %e
(%i10) ratsimp(%);  /* Ojo: %i^2=-1 */
                                  2
                                 t
                               - --
                                 2
(%o10)                        %e
```

13 Equações diferenciais

Com Maxima se pode resolver analíticamente algumas equações diferenciais ordinárias de primeira e segunda ordem mediante a instrução `ode2`.

Uma equação diferencial de primeira ordem possui a forma geral $F(x,y,y') = 0$, de onde $y' = \frac{dy}{dx}$. Para expressar uma destas equações se faz uso de `diff`,

```
(%i1) /* equação de variáveis separadas */
      eq:(x-1)*y^3+(y-1)*x^3*'diff(y,x)=0;
                 3         dy            3
(%o1)           x  (y - 1) -- + (x - 1) y = 0
                           dx
```

sendo obrigatorio o uso do apóstrofo (') antes de `diff` com o objetivo evitar o cálculo da derivada, que por outro lado daria zero ao não haver-se declarado a variável `y` como dependente de `x`. Para a resolução desta equação tão somente teremos que fazer

```
(%i2) ode2(ec,y,x);
                    2 y - 1          2 x - 1
(%o2)               ------- = %c - -------
                        2              2
                     2 y            2 x
```

de onde `%C` representa uma constante, que se ajustará de acordo à condição inicial que se lhe imponha à equação. Suponha-se que se sabe que quando $x = 2$, deve verificar-se que $y = -3$, o qual faremos saber ao Maxima a través da função `ic1`,

```
(%i3) ic1(%o2,x=2,y=-3);
                                 2
                    2 y - 1     x  + 72 x - 36
(%o3)               ------- = - --------------
                        2               2
                     2 y             72 x
```

Vejamos exemplos de outros tipos de equações diferenciais que o Maxima pode resolver,

```
(%i4) /* equação homogênia */
      ode2(x^3+y^3+3*x*y^2*'diff(y,x),y,x);
```

```
                              3    4
                           4 x y  + x
(%o4)                      ---------- = %c
                              4
```

Neste caso, quando não se inclui o símbolo de igualdade, se dá por certo que a expressão é igual a zero.

```
(%i5) /* redutível a homogênea */
      ode2('diff(y,x)=(x+y-1)/(x-y-1),y,x);
             2    2                      x - 1
        log(y  + x  - 2 x + 1) + 2 atan(-----)
                                          y
(%o5)   ------------------------------------ = %c
                          4
(%i6) /* equação exata */
       ode2((4*x^3+8*y)+(8*x-4*y^3)*'diff(y,x),y,x);
                      4            4
(%o6)               - y  + 8 x y + x  = %c
(%i7) /* Bernoulli */
       ode2('diff(y,x)-y+sqrt(y),y,x);
(%o7)           2 log(sqrt(y) - 1) = x + %c
(%i8) solve(%,y);
                   x + %c       x/2 + %c/2
(%o8)       [y = %e       + 2 %e           + 1]
```

Neste último caso, optamos por obter a solução em sua forma explícita.

Uma equação diferencial ordinária de segunda ordem possui a forma general $F(x,y,y',y'')=0$, sendo y'' a segunda derivada de y com relação a x. Como exemplo,

```
(%i9) 'diff(y,x)=x+'diff(y,x,2);
                               2
                          dy   d y
(%o9)                     -- = --- + x
                          dx     2
                               dx
(%i10) ode2(%,y,x);
```

```
                                  2
                             x    x  + 2 x + 2
(%o10)          y = %k1 %e   + ------------ + %k2
                                    2
```

Maxima nos devolve um resultado que depende dos parâmetros, `%k1` e `%k2`, que para ajustá-los necessitaremos proporcionar certas condições iniciais; se sabemos que quando $x = 1$ então $y = -1$ e $y' = \left.\frac{dy}{dx}\right|_{x=1} = 2$, faremos uso da instrução `ic2`,

```
(%i11) ic2(%,x=1,y=-1,diff(y,x)=2);
                              2
                             x  + 2 x + 2   7
(%o11)                  y = ------------ - -
                                   2         2
```

No caso das equações de segunda ordem, também é possível ajustar os parámetros da solução especificando condições de contorno, isto é, fixando dois pontos do plano pelos quais passe a solução; assim, se a solução obtida em `(%o10)` deve pasar pelos pontos $(-1,3)$ e $(2,\frac{5}{3})$, fazemos

```
(%i12) bc2(%o10,x=-1,y=3,x=2,y=5/3);
                    x + 1    2                    3
              35 %e         x  + 2 x + 2   15 %e  + 20
(%o12) y = - ---------- + ------------ + -----------
                  3             2              3
              6 %e  - 6                    6 %e  - 6
```

Note-se que este cálculo se solicita ao Maxima com `bc2`.

A resolução de sistemas de equações diferenciais se faz com chamadas à função `desolve`. Neste contexto é preciso ter em conta que se deve utilizar notação funcional dentro da expressão `diff`; um exemplo aclarará este ponto, resolvendo o sistema

$$\begin{cases} \frac{df(x)}{dx} &= 3f(x) - 2g(x) \\ \frac{dg(x)}{dx} &= 2f(x) - 2g(x) \end{cases}$$

```
(%i13) desolve(['diff(f(x),x)=3*f(x)-2*g(x),
                'diff(g(x),x)=2*f(x)-2*g(x)],
               [f(x),g(x)]);
```

```
                                    - x
                  (2 g(0) - f(0)) %e
(%o13) [f(x) = ---------------------
                           3
                         2 x
   (2 g(0) - 4 f(0)) %e
 - ----------------------, g(x) =
               3
                     - x                       2 x
(4 g(0) - 2 f(0)) %e       (g(0) - 2 f(0)) %e
---------------------- - ---------------------]
           3                          3
```

Como se vê, as referências às funções devem incluir a variável independente e as equações estarão entre colchetes, assim como os nomes das funções. Observamos na resposta que nos dá Maxima a presença de `f(0)` e `g(0)`, o que é devido ao fato de se desconhecerem as condições de contorno do sistema.

Neste último exemplo, suponhamos que queremos resolver o sistema de equações diferenciais

$$\begin{cases} \frac{df(x)}{dx} &= f(x)+g(x)+3h(x) \\ \frac{dg(x)}{dx} &= g(x)-2h(x) \\ \frac{dh(x)}{dx} &= f(x)+h(x) \end{cases}$$

sob as condições $f(0)=-1, g(0)=3$ y $f(0)=1$. em primeiro lugar introduziremos estas condições com a função `atvalue`, para posteriormente solicitar a resolução do sistema,

```
(%i14) atvalue(f(x),x=0,-1)$
(%i15) atvalue(g(x),x=0,3)$
(%i16) atvalue(h(x),x=0,1)$
(%i17) desolve(['diff(f(x),x)=f(x)+g(x)+3*h(x),
         'diff(g(x),x)=g(x)-2*h(x),
         'diff(h(x),x)=f(x)+h(x)], [f(x),g(x),h(x)]);
                   2 x     2 x       - x
(%o17) [f(x) = x %e    + %e    - 2 %e   ,
                 2 x       2 x     - x
g(x) = - 2 x %e    + 2 %e    + %e   ,
           2 x     - x
h(x) = x %e    + %e   ]
```

14 Números aleatórios

Se seu argumento é um número inteiro positivo, a função `random(n)` gera um número pseudoaleatório com distribuição uniforme discreta entre 0 e $n-1$, ambos inclusive; assim, uma simulação do lançamento de um dado seria

```
(%i1) random(6)+1;
(%o1)                         5
```

y uma serie de 100 lanzamientos de uma moneda,

```
(%i2) makelist(random(2),i,1,100);
(%o2) [0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1,
1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0,
1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0,
0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1,
0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1,
1, 1, 1]
```

Quando o argumento é um número decimal positivo, a variável aleatória que se simula é a uniforme contínua, dando como resultado um número real pertencente ao intervalo $[0, r)$,

```
(%i3) random(6.0);
(%o3)                1.171107706786732
```

O algorítmo gerador dos números pseudoaleatórios é determinista, de maneira que partindo de uma mesma semente ou valor inicial, se gerará a mesma seqüência de números. Para controlar o valor desta semente dispomos das funções `make_random_state` e `set_random_state`; por exemplo, para definir uma semente que se gere a partir do estado atual do relógio do sistema faremos

```
(%i4) nova_semente: make_random_state(true)$
```

Sem impedimentos, para que tal semente se ative no gerador, devemos indicar expressamente fazendo

```
(%i5) set_random_state(nova_semente)$
```

O argumento da função `make_random_state` pode ser também um número inteiro, como se faz no exemplo de mais abaixo.

Vejamos um caso de aplicação de tudo isto. Suponhamos que queremos simular diferentes séries estocásticas, de forma que todas elas sejam iguais. Se fazemos

```
(%i6) makelist(random(6),i,1,10);
(%o6)          [0, 2, 2, 4, 3, 0, 3, 3, 5, 3]
(%i7) makelist(random(6),i,1,10);
(%o7)          [5, 4, 4, 5, 0, 1, 3, 1, 3, 4]
(%i8) makelist(random(6),i,1,10);
(%o8)          [4, 4, 3, 4, 5, 2, 5, 5, 2, 3]
```

o mais provável é que obtenhamos três seqüências distintas, como no exemplo. Se fazemos

```
(%i9) semente: make_random_state(123456789)$
(%i10) set_random_state(semilla)$ makelist(random(6),i,1,10);
(%o11)         [4, 4, 0, 1, 0, 3, 2, 5, 4, 4]
(%i12) set_random_state(semilla)$ makelist(random(6),i,1,10);
(%o13)         [4, 4, 0, 1, 0, 3, 2, 5, 4, 4]
(%i14) set_random_state(semilla)$ makelist(random(6),i,1,10);
(%o15)         [4, 4, 0, 1, 0, 3, 2, 5, 4, 4]
```

se verá que as três seqüências são iguais, já que antes de gerar cada amostra aleatória reiniciamos o estado do gerador.

Outra variável aleatória que interessa simular é a normal ou gaussiana com média $\mu = m \in \mathbb{R}$ e desvio padrão $\sigma = s > 0$, para o qual se dispõe da função `gauss(m,s)`,

```
(%i16) gauss(31.45,1.23);
(%o16)                32.2873298461951
```

O gerador de números aleatórios normais opera independentemente do mecanismo de semente comentado mais acima.

15 Gráficos

Maxima não está habilitado para realizar gráficos, pelo que necesitará de um programa externo que realize esta tarefa. Nós a partir do Maxima nos encarregaremos de ordenar que tipo de gráfico queremos e Maxima se encarregará de comunicá-lo à aplicação gráfica que está ativa nesse momento, que por padrão será o Gnuplot.

Veremos em primeiro lugar alguns exemplos de como gerar gráficos a partir do Maxima com Gnuplot e logo trataremos brevemente sobre como podemos modificar algumas das opções padrão do ambiente gráfico.

O mais simples é desenhar uma função no plano, por exemplo $y = e^{-x^2}$, em um intervalo tal como $[-2,5]$,

```
(%i1) plot2d(exp(-x^2),[x,-2,5])$
```

cujo resultado se pode observar na região *a)* da Figura 6. Na região *b)* da mesma figura se pode ver como é possível representar varias funções de uma só vez,

```
(%i2) plot2d([-x^2,1+x,7*sin(x)],[x,-2,5])$
```

Para a realização de funções definidas na forma paramétrica necessitamos fazer uso do símbolo `parametric`, Figura 7,

```
(%i3) plot2d([parametric,t,t*sin(1/t),[t,0.01,0.2]])$
```

O resultado que se obtém é o da região *a)*. Maxima calcula por padrão um número fixo de pontos da função que logo utilizará para representá-la; como esta é uma função que varia muito perto da orígem, pediremos que nos faça o desenho com um maior número de pontos, o que se fará mediante a opção `nticks`, tal como se indica a seguir,

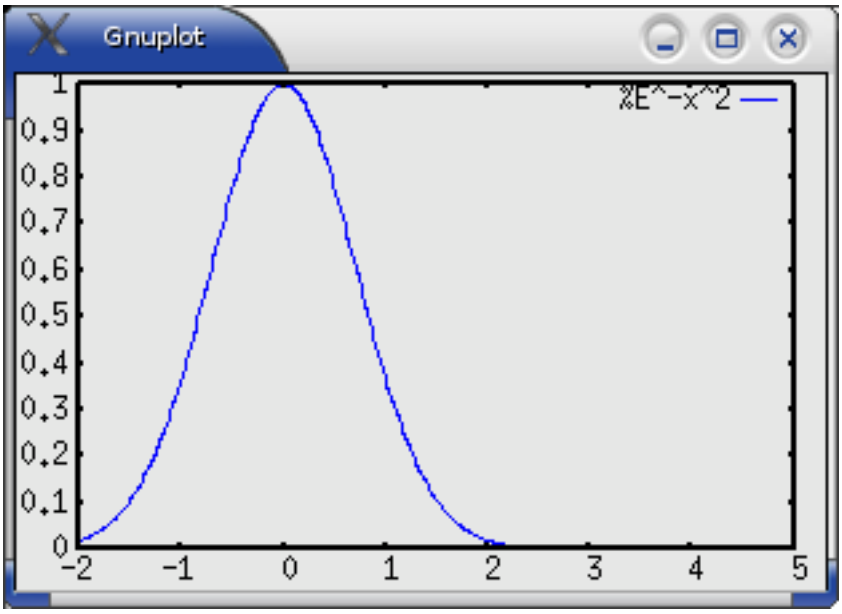
```
(%i4) plot2d([parametric,t,t*sin(1/t),[t,0.01,0.2],[nticks,500]])$
```

Se comprova que o aspecto mudou apreciadamente.

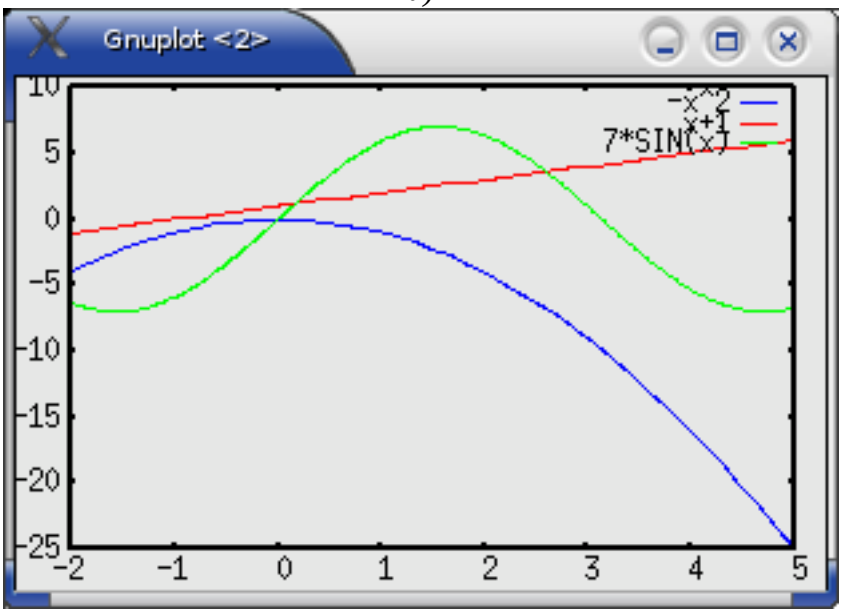
O seguinte exemplo mostra a presença em um mesmo gráfico da função $y = x^3 + 2$ junto com a circunferência de raio unitário, expressa na forma paramétrica, que na região *a)* da Figura 8 se vê como uma elipse devido à diferença de escalas nos eixos,

```
(%i5) plot2d([x^3+2,[parametric,cos(t),sin(t),[t,-5,5]]],
(%i6)        [x,-1.3,1.3],[nticks,500])$
```

Também é possível a geração de superficies em 3D definidas da forma $z = f(x,y)$, região *b)* da Figura 8,

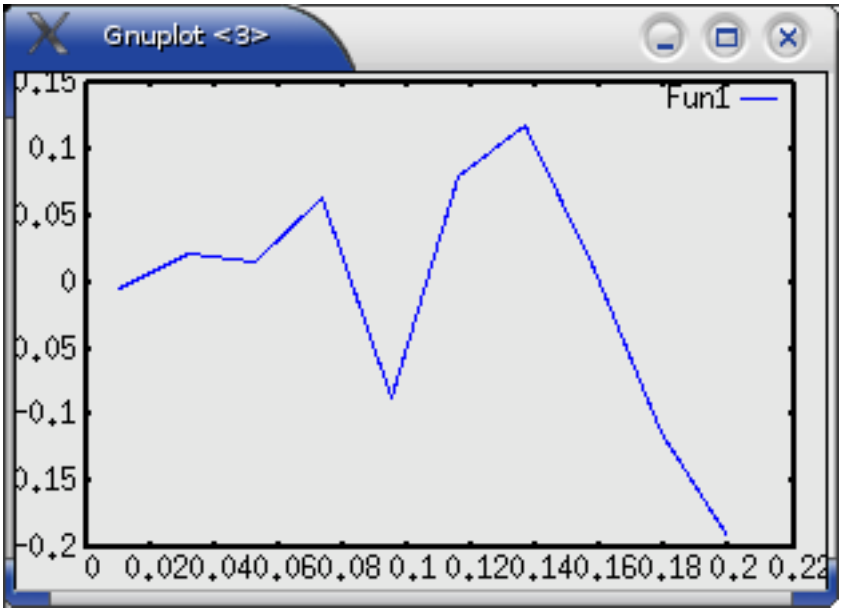


a)

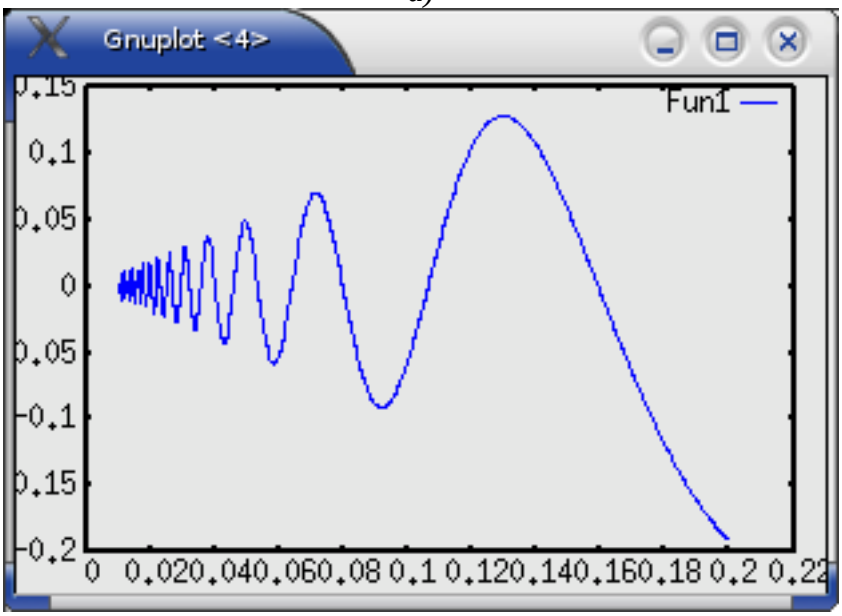


b)

Figura 6: Funções em coordenadas cartesianas: *a)* uma única função; *b)* uma família de funções.

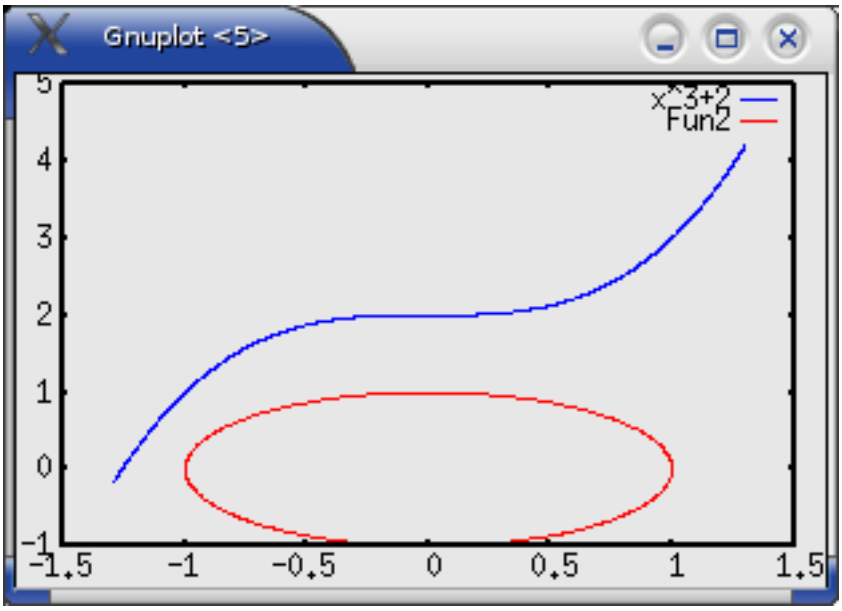


a)

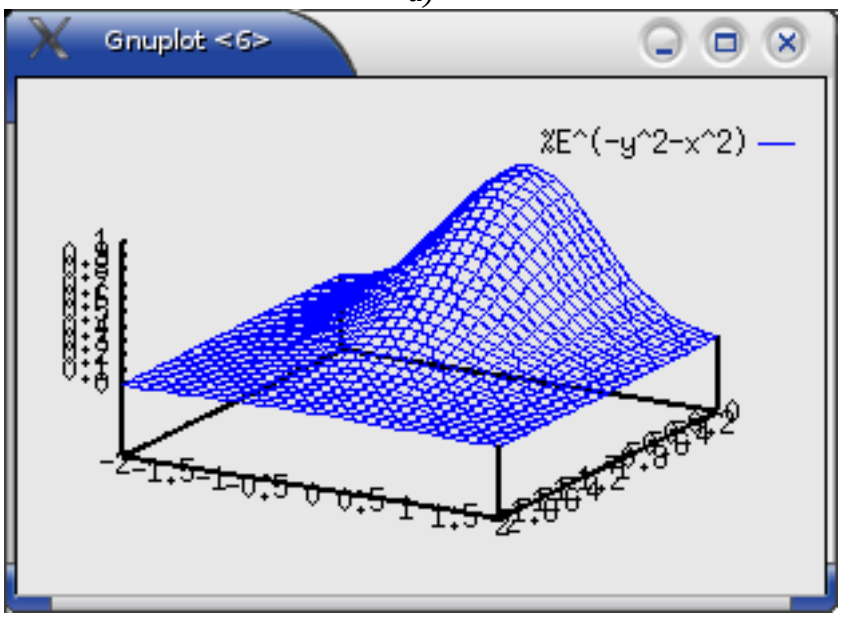


b)

Figura 7: Controle do número de pontos: *a)* número de pontos padrão; *b)* a mesma função com 500 pontos.



a)



b)

Figura 8: Mais tipos de gráficos: *a)* combinando funções paramétrica e não paramétrica; *b)* superfície em 3D.

```
(%i7) plot3d(exp(-x^2-y^2),[x,-2,2],[y,-2,0])$
```

Finalmente, do manual de Maxima se extraiu a superfície (cinta) de Möbius como um exemplo de superfície paramética, região *a)* da Figura 9,

```
(%i8) plot3d([cos(x)*(3+y*cos(x/2)),
              sin(x)*(3+y*cos(x/2)),y*sin(x/2)],
              [x,-%pi,%pi],[y,-1,1],['grid,50,15]);
```

A função `plot3d` é capaz de gerar uma curva paramétrica em três dimensões, como o prova esta hélice, que se pode ver na região *b)* da Figura 9,

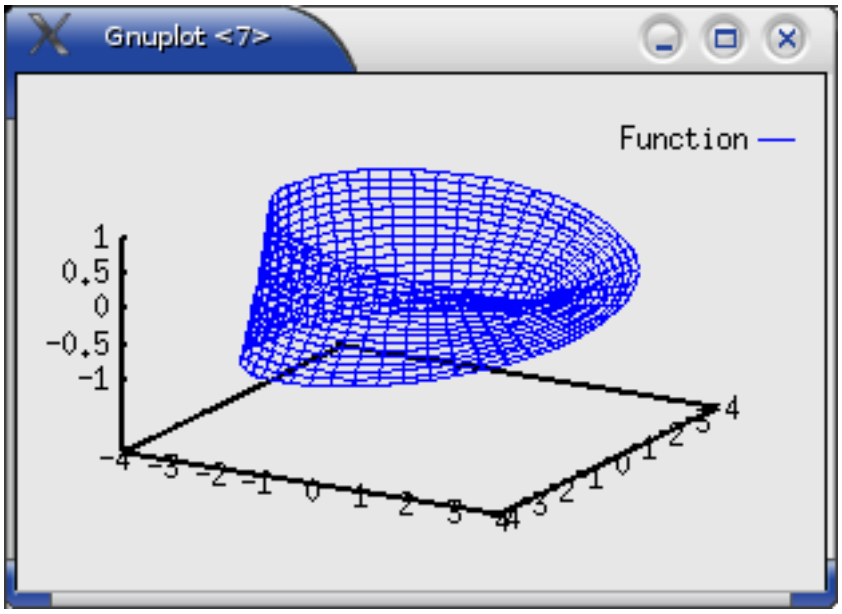
```
(%i9) plot3d([cos(t),sin(t),2*t],[t,-%pi,%pi],[u,0,1]);
```

O controle das opções gráficas se consegue manipulando a variável global `plot_options`, cujo estado por padrão é

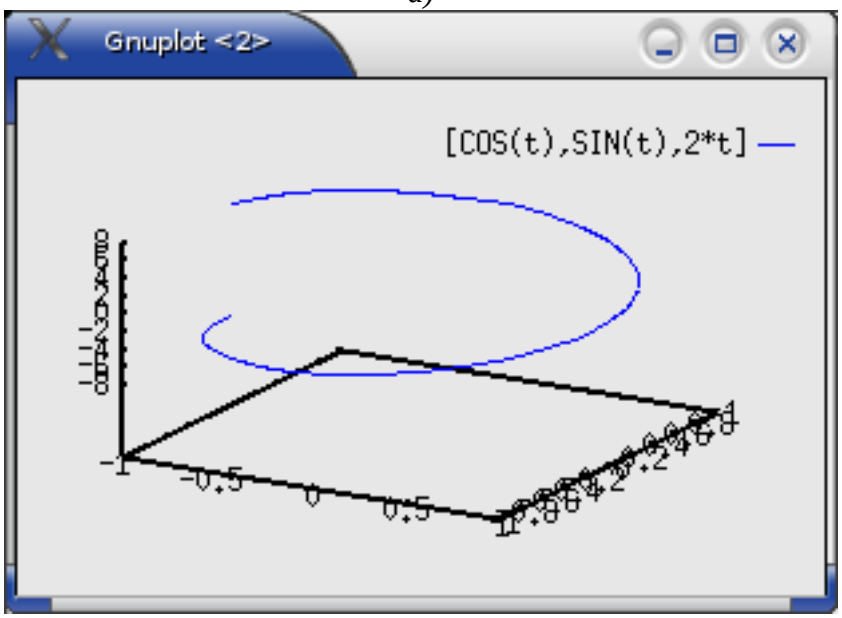
```
(%i10) plot_options;
(%o10) [[x, - 1.755559702014e+305, 1.755559702014e+305],
[y, - 1.755559702014e+305, 1.755559702014e+305],
[t, - 3, 3], [GRID, 30, 30], [VIEW_DIRECTION, 1, 1, 1],
[COLOUR_Z, FALSE], [TRANSFORM_XY, FALSE],
[RUN_VIEWER, TRUE], [PLOT_FORMAT, GNUPLOT],
[GNUPLOT_TERM, DEFAULT], [GNUPLOT_OUT_FILE, FALSE],
[NTICKS, 10], [ADAPT_DEPTH, 10], [GNUPLOT_PM3D, FALSE],
[GNUPLOT_PREAMBLE, ], [GNUPLOT_CURVE_TITLES, [DEFAULT]],
[GNUPLOT_CURVE_STYLES, [with lines 3, with lines 1,
with lines 2, with lines 5, with lines 4, with lines 6,
with lines 7]], [GNUPLOT_DEFAULT_TERM_COMMAND, ],
[GNUPLOT_DUMB_TERM_COMMAND, set term dumb 79 22],
[GNUPLOT_PS_TERM_COMMAND, set size 1.5, 1.5;set term postsc#
ript eps enhanced color solid 24]]
```

Para maior informação sobre o significado de cada um dos elementos desta lista sería aconselhável executar o comando `describe(plot_options)`.

Já se comentou que a menos que se lhe indique o contrário, Maxima invocará o programa Gnuplot para a representação de um gráfico, pode ser que prefiramos o programa Openmath, que faz parte da distribução de Maxima; em tal caso teríamos que modificar previamente as opções guardadas em `plot_options` e a seguir solicitar o gráfico desejado, como neste caso no que se representa a função gamma e sua inversa, cujo resultado se observa na região *a)* da Figura 10



a)



b)

Figura 9: Gráficos paramétricos em 3D: *a)* uma superficie; *b)* uma curva.

```
(%i11) set_plot_option([plot_format, openmath])$
(%i12) plot2d([gamma(x),1/gamma(x)],[x,-4.5,5],[y,-10,10])$
```

Openmath também é capaz de realizar gráficos em três dimensões definidas em coordenadas polares e que se representa na região *b)* da Figura 10. Note-se como a partir da própria função é possível alterar as opções gráficas; sem impedimentos, feito desta maneira as trocas só possuem efeito para a realização do gráfico presente, sem alterar as opções globais do Maxima.

```
(%i13) plot3d(r^0.33*cos(t/3),[r,0,1],[t,0,6*%pi],
              ['grid,20,80],['transform_xy,polar_to_xy])$
```

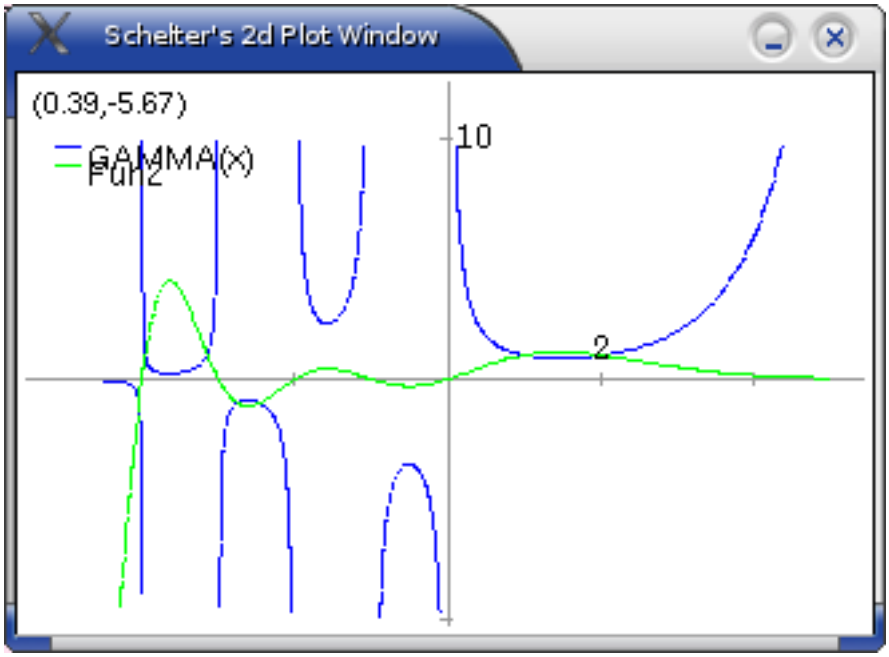
As Figuras que foram representadas até agora nesta seção são capturadas da janela do monitor gerada por Gnuplot ou por Openmath e guardadas no disco em formato PNG. Sem impedimentos, podemos fazer com que o programa gráfico gere diretamente um arquivo PNG em lugar de mostrar no monitor. Voltemos ao Gnuplot e obtenhamos diretamente deste programa um arquivo com este formato (png) de um dos gráficos que já realizamos anteriormente,

```
(%i14) set_plot_option([plot_format, gnuplot])$
(%i15) plot3d(exp(-x^2-y^2),[x,-2,2],[y,-2,0],
              [gnuplot_preamble,"set terminal png size 420,320;
                                 set out 'grafico.png'"])$
```

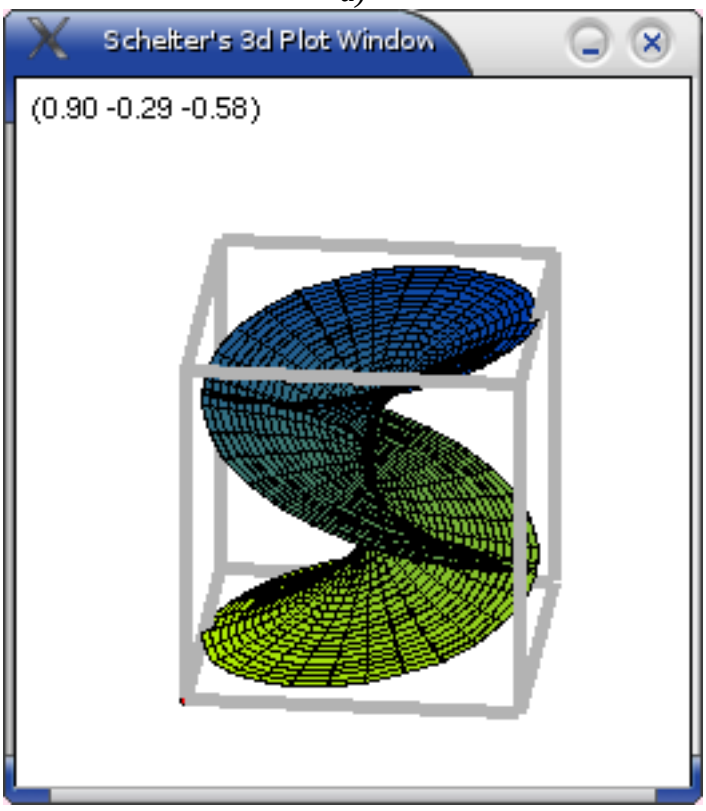
Depois de alterar globalmente o programa gráfico, voltemos a representar a superficie tridimensional que já vimos na região *b)* da Figura 8. Veja-se como se mudou o parâmetro gráfico `gnuplot_preamble` para indicar ao Gnuplot que gere um arquivo em formato PNG de certas dimensões dadas e que o guarde com o nome `grafico.png`. O resultado o vemos na região *a)* da Figura 11. Quando se trabalha em Gnuplot, o parâmetro `gnuplot_preamble` permite passar a este programa uma série de comandos que aperfeiçoam os detalles; estes comandos devem ir separados por ponto e vírgula e devem ser os próprios da linguagem do Gnuplot. Para um melhor domínio destes detalhes é aconselhável recorrer à documentação deste programa (www.gnuplot.info).

Por último, se se quer um arquivo gráfico em Postscript, repitamos o último exemplo solicitando este formato,

```
(%i16) plot3d(exp(-x^2-y^2),[x,-2,2],[y,-2,0],
              [gnuplot_preamble,"set terminal postscript eps;
                                 set out 'grafico.eps'"])$
```

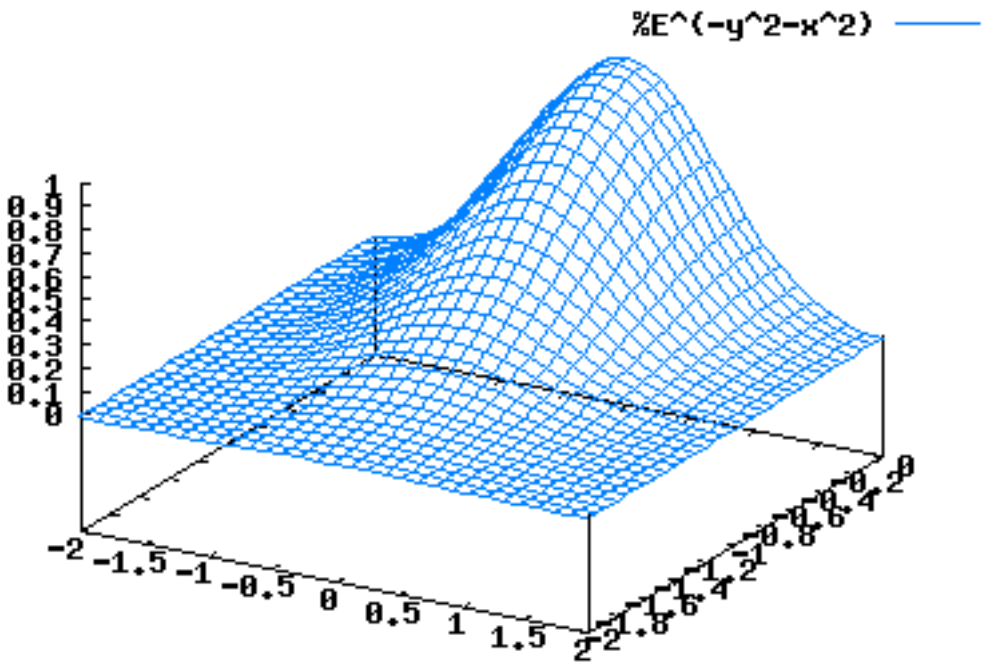


a)

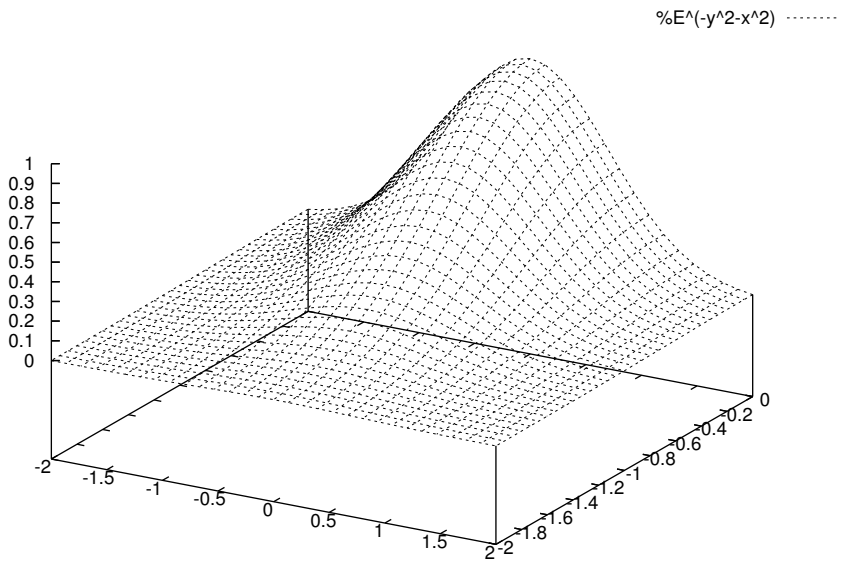


b)

Figura 10: Gráficos em Openmath: *a)* no plano; *b)* em três dimensões.



a)



b)

Figura 11: Formatos de arquivos gráficos: *a)* PNG; *b)* Postscript.

O resultado enontra-se na região *b)* da Figura 11. Os gráficos armazenados em formato Postscript, os que possuem extensão EPS, são adequados quando se planeja criar um documento baseado em TEX-LATEX.

Posto que estamos falando de LATEX, este é um bom lugar para fazer referência a uma função que transforma uma expressão de Maxima para o formato TEX, de maneira que o resultado que se obtenha possa ser incorporado facilmente (copiar e colar) a um arquivo fonte de LATEX; a modo de exemplo, calculemos uma derivada para posteriormente transformá-la,

```
(%i17) 'diff(sin(x^x)*sqrt(log(x)),x)=
           diff(sin(x^x)*sqrt(log(x)),x);
       d                       x
(%o17) -- (sqrt(log(x)) sin(x )) =
       dx

              x
         sin(x )          x                                    x
     ---------------- + x  sqrt(log(x)) (log(x) + 1) cos(x )
     2 x sqrt(log(x))
(%i18) tex(%);

$${{d}\over{d\,x}}\,\left(\sqrt{\log x}\,\sin x^{x}\right)={{\sin x^{
 x}}\over{2\,x\,\sqrt{\log x}}}+x^{x}\,\sqrt{\log x}\,\left(\log x+1
 \right)\,\cos x^{x}$$
(%o18)                       false
```

O apóstrofo que se coloca na entrada `(%i17)` antes da função `diff` serve para devolver a expressão sem avaliá-la, tal como aparece no membro esquerdo da igualdade `(%o17)`. Uma vez copiada e colada a resposta da entrada `(%i18)` em um fonte LATEX, sua compilação dará como resultado a expressão

$$\frac{d}{dx}\left(\sqrt{\log x}\,\sin x^x\right)=\frac{\sin x^x}{2x\sqrt{\log x}}+x^x\,\sqrt{\log x}\,(\log x+1)\,\cos x^x$$

mais fácil de interpretar por um humano.

16 Listas

As listas são objetos muito potentes à hora de representar estruturas de dados; de fato, toda expressão de Maxima se representa internamente como uma lista, o que não é de estranhar tomando-se em conta que Maxima está programado em Lisp (*List Processing*). Vejamos como podemos ver a representação interna, isto é em Lisp, de uma simples expressão tal como $1+3a$,

```
(%i1) :lisp #$1+3*a$
((MPLUS SIMP) 1 ((MTIMES SIMP) 3 $a))
```

Note-se que o formato geral é da forma `:lisp #$expr$`, sendo `expr` uma expressão qualquer na linguagem do Maxima.

Todavia a nível de usuário que não está interesado nas interioridades de Maxima, também se pode trabalhar com listas como as definidas a seguir, sempre encerradas entre colchetes,

```
(%i2) q:[b,5,a,d,1,3,7]$
(%i3) r:[1,[a,3],sqrt(3)/2,"Don Quijote"];
                          sqrt(3)
(%o3)        [1, [a, 3], -------, Don Quijote]
                             2
```

Vemos que os elementos de uma lista podem por sua vez serem também listas, expressões matemáticas ou cadeias de caracteres incluídas entre aspas duplas, o que pode ser aproveitado para a construção e manipulação de estruturas mais ou menos complexas. Extraiamos a seguir alguma informação das listas anteriores,

```
(%i4) listp(r); /* é r uma lista? */
(%o4)                          true
(%i5) first(r); /* primeiro elemento */
(%o5)                           1
(%i6) second(r); /* segundo elemento */
(%o6)                         [a, 3]
(%i7) third(r); /* ...até o décimo somente */
                           sqrt(3)
(%o7)                      -------
                              2
(%i8) last(r); /* o último da lista */
(%o8)                     Don Quijote
(%i9) rest(r); /* todos menos o primeiro */
```

```
                        sqrt(3)
(%o9)          [[a, 3], -------, Dom Quixote]
                           2
(%i10) part(r,3); /* peço o que quero */
                         sqrt(3)
(%o10)                   -------
                            2
(%i11) length(r); /* quantos existem? */
(%o11)                       4
(%i12) reverse(r); /* invertemos */
                          sqrt(3)
(%o12)        [Dom Quixote, -------, [a, 3], 1]
                             2
(%i13) member(a,r); /* é a um elemento?*/
(%o13)                      false
(%i14) member([a,3],r); /* ou é [a,3]? */
(%o14)                       true
(%i15) sort(q); /* ordeno */
(%o15)             [1, 3, 5, 7, a, b, d]
(%i16) delete([a,3],r); /* apago o elemento */
                       sqrt(3)
(%o16)           [1, -------, Dom Quixote]
                          2
```

Note-se que em todo este tempo a lista `r` não foi alterada,

```
(%i17) r;
                         sqrt(3)
(%o18)        [1, [a, 3], -------, Dom Quixote]
                            2
```

Algumas funções de Maxima permitem adicionar novos elementos a uma lista, tanto ao início como ao final,

```
(%i19) cons(1+2,q);
(%o19)             [3, b, 5, a, d, 1, 3, 7]
(%i20) endcons(x,q);
(%o20)             [b, 5, a, d, 1, 3, 7, x]
```

Neste exemplo observamos também que a lista `q` não foi alterada; se o que queremos é atualizar seu conteúdo,

```
(%i26) q: endcons(x,cons(1+2,q))$
(%i27) q;
(%o27)          [3, b, 5, a, d, 1, 3, 7, x]
```

É possível unir duas listas,

```
(%i28) append(r,q);
                    sqrt(3)
(%o28) [1, [a, 3], -------, Dom Quixote, 3, b, 5, a, d, 1,
                       2

                                                        3, 7, x]
```

Quando os elementos de uma lista obedecem a um certo critério de construção, podemos utilizar a função `makelist`,

```
(%i29) s:makelist(2+k*2,k,0,10);
(%o29)    [2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22]
```

de onde indicamos ao Maxima que nos construa uma lista com elementos da forma `2+2*k`, de modo que `k` tome valores inteiros de 0 a 10.

A função `apply` permite fornecer a outra função todos os elementos de uma lista como argumentos, assim podemos adicionar ou multiplicar todos os elementos da lista `s` recém criada,

```
(%i30) apply("+",s);
(%o30)                      132
(%i31) apply("*",s);
(%o31)                  81749606400
```

ainda que estas duas operações tivessem sido melhor realizadas com as funções `sum` e `product`.

Às vezes interessará aplicar uma mesma função a varios elementos de uma lista de forma independente, para o que faremos uso de `map`; a seguir um exemplo de cálculo de fatoriais e outro trigonométrico,

```
(%i32) map("!",s);
(%o32) [2, 24, 720, 40320, 3628800, 479001600, 87178291200,
20922789888000, 6402373705728000, 2432902008176640000,
112400072777607680000]
(%i33) map(sin,s);
(%o33) [sin(2), sin(4), sin(6), sin(8), sin(10), sin(12),
                sin(14), sin(16), sin(18), sin(20), sin(22)]
```

Por último, as listas também podem ser utilizadas em operações aritméticas,

```
(%i34) [1,2,3]+[a,b,c];
(%o34)             [a + 1, b + 2, c + 3]
(%i35) [1,2,3]*[a,b,c];
(%o35)                  [a, 2 b, 3 c]
(%i36) [1,2,3]/[a,b,c];
                          1  2  3
(%o36)                   [-, -, -]
                          a  b  c
(%i37) [1,2,3]-[a,b,c];
(%o37)             [1 - a, 2 - b, 3 - c]
(%i38) [1,2,3].[a,b,c]; /* produto escalar */
(%o38)                 3 c + 2 b + a
(%i39) [a,b,c]^3;
                          3   3   3
(%o39)                   [a , b , c ]
(%i40) 3^[a,b,c];
                          a   b   c
(%o40)                   [3 , 3 , 3 ]
```

Para que estas operações possam realizar-se sem problemas, a variável global `listarith` deve tomar o valor `true`, caso contrário o resultado será bem diferente,

```
(%i41) listarith:false$
(%i42) [1,2,3]+[4,5,6];
(%o42)             [4, 5, 6] + [1, 2, 3]
(%i43) listarith:true$
```

Como já se viu no começo desta seção, uma lista pode ser elemento de outra lista, se queremos desfazer todas as listas interiores para que seus elementos passem a fazer parte da exterior,

```
(%i44) flatten([1,[a,b],2,3,[c,[d,e]]]);
(%o44)            [1, a, b, 2, 3, c, d, e]
```

17 Operações com conjuntos

Se define a seguir um conjunto mediante a função `set`,

```
(%i1) c1:set(a,[2,k],b,sqrt(2),a,set(a,b),
             3,"Sancho",set(),b,sqrt(2),a);
(%o1)  {3, sqrt(2), {}, [2, k], a, {a, b}, b, Sancho}
```

Como se vê, se admitem objetos de mui diversa natureza como elementos de um conjunto: números, expressões, o conjunto vazio (`{}`), listas, outros conjuntos ou cadeias de caracteres. Quando se trabalha com listas, pode ser de utilidade considerar seus componentes como elementos de um conjunto, logo se necessita uma função que nos transforme uma lista em conjunto,

```
(%i2) [[2,k],sqrt(2),set(b,a),[k,2],"Pança"];
(%o2)     [[2, k], sqrt(2), {a, b}, [k, 2], Pança]
(%i3) c2:setify(%);
(%o3)     {sqrt(2), [2, k], {a, b}, [k, 2], Pança}
```

a mudança na natureza destas duas coleções de objetos se aprecia na presença de chaves frente aos colchetes. De igual maneira, podemos transformar um conjunto em lista,

```
(%i4) listify(%o1);
(%o4)  [3, sqrt(2), {}, [2, k], a, {a, b}, b, Sancho]
```

Comprovemos rapidamente que `{}` representa o conjunto vazio,

```
(%i5) emptyp(%[3]);
(%o5)                         true
```

Recorde-se que `%` substitui a última resposta dada por Maxima, que neste caso havia sido uma lista, pelo que `%[3]` faz referência à sua terceira componente.

Para comprovar se um certo objeto faz parte de um conjunto fazemos uso da instrução `elementp`,

```
(%i6) elementp(sqrt(2),c1);
(%o6)                         true
```

É possível extrair um elemento de um conjunto e logo adicionar-lhe outro diferente

```
(%i7) c1: disjoin(sqrt(2),c1);  /* sqrt(2) retirado */
(%o7)     {3, {}, [2, k], a, {a, b}, b, Sancho}
(%i8) c1: adjoin(sqrt(3),c1);  /* sqrt(3) dentro */
(%o8) {3, sqrt(3), {}, [2, k], a, {a, b}, b, Sancho}
```

A substituição que se acaba de realizar se podia ter feito com a função `subst`,

```
(%i9) /* novamente coloca-se sqrt(2) */
      subst(sqrt(2),sqrt(3),c1);
(%o9) {3, sqrt(2), {}, [2, k], a, {a, b}, b, Sancho}
```

A comprovação de que um conjunto é subconjunto de outro se faz com a função `subsetp`,

```
(%i10) subsetp(set([k,2],"Pança"),c2);
(%o10)                      true
```

A seguir alguns exemplos de operações com conjuntos,

```
(%i11) union(c1,c2);
(%o11) {3, sqrt(2), sqrt(3), {}, [2, k], a, {a, b}, b,
                                      [k, 2], Pança, Sancho}
(%i12) intersection(c1,c2);
(%o12)                {[2, k], {a, b}}
(%i13) setdifference(c1,c2);
(%o13)         {3, sqrt(3), {}, a, b, Sancho}
(%i14) cardinality(%);
(%o14)                       6
```

Vemos aqui também como pedir o cardinal de um conjunto.

Igual ao que se viu em como aplicar uma função a todos os elementos de uma lista, podemos fazer o mesmo com os elementos de um conjunto,

```
(%i15) map(sin,set(1,2,3,4,5));
(%o15)    {sin(1), sin(2), sin(3), sin(4), sin(5)}
```

Por último, o produto cartesiano de três conjuntos,

```
(%i16) cartesian_product(set(1,2),set(a,b,c),set(x,y));
(%o16) {[1, a, x], [1, a, y], [1, b, x], [1, b, y],
[1, c, x], [1, c, y], [2, a, x], [2, a, y], [2, b, x],
[2, b, y], [2, c, x], [2, c, y]}
```

18 Operações lógicas

Como qualquer outro interpretador ou ambiente de programação, Maxima dispõe também da capacidade de avaliar predicados lógicos, aqueles que podem ser ou verdadeiros (`true`), ou falsos (`false`). Os predicados mais simples são funções que devolvem um destes dois valores lógicos; exemplos dessas já apareceram em seções anteriores,

```
(%i1) listp([[1,2],[a,b]]);
(%o1)                         true
(%i2) matrixp([[1,2],[a,b]]);
(%o2)                        false
(%i3) evenp(2^3);
(%o3)                         true
```

Posto que Maxima é um ambiente de processamento matemático, é freqüênte ter que comparar duas quantidades. Os operadores de comparação se resumem na seguinte tabela,

=	...igual a...
#	...diferente de...
>	...maior que...
<	...menor que...
>=	...maior ou igual a...
=<	...menor ou igual a...

Se se quer saber se um número é menor que outro podemos fazer uso da função `is`,

```
(%i4) is(sqrt(7895)<85);
(%o4)                        false
(%i5) is(4^12#16777216);
(%o5)                        false
```

No primeiro caso comprovamos se $\sqrt{7895}$ é estritamente menor que 85 e no segundo comprovamos se a potência 4^{12} é diferente de 16777216.

Junto com estes predicados simples, os conectores `and`, `or` e `not` permitem construir formas mais complexas. a tabela adicional resume o comportamento destes operadores,

p	q	p and q	p or q	not p
false	false	false	false	true
false	true	false	true	true
true	false	false	true	false
true	true	true	true	false

Na hora de escrever expressões lógicas convém ter em conta a ordem de precedência; primeiro se avalia `not`, a seguir `and` e, finalmente, `or`; em caso de dúvida sempre se pode recorrer aos parêntesis. Um exemplo,

```
(%i6) is(not 3<4 or 5<5);
(%o6)                         false
```

Aqui, primeiro se avalia `not 3<4`, que dá como resultado `false`; a seguir, se avalia se é certo ou não que `5<5`, o qual é também falso, finalmente se obtém o resultado correspondente a `false or false`.

O leitor devería ser capaz de justificar as seguintes respostas,

```
(%i7) is(3#4 and not 7<=2);
(%o7)                          true
(%i8) is((5<8 or 8<2) and oddp(3) and evenp(sqrt(16)));
(%o8)                          true
```

19 Programação no Maxima

Com vistas à optimização de tempo e esforço será interessante poder definir de uma só vez nossas próprias funções e poder logo reutilizá-las quantas vezes seja necessário.

A programação de funções requer certas sentenças de controle que são comuns, com mais ou menos matrizes (nuances) em sua sintaxe, em todas as linguagens de programação.

Um elemento imprescindível no controle do fluxo é a sentença condicional `if`, cuja estrutura é

```
if <cond> then <expr1> else <expr2>
```

tal como em

```
(%i1) x:30!$ y:exp(30)$
(%i3) if (x>y) then 0 else 1;
(%o3)                         0
```

A condição `<cond>` é uma expressão lógica que admite os operadores `and`, `or` e `not`, sendo seus argumentos predicados lógicos, cujo valor só pode ser *verdadero* ou *falso*,

Outra sentença que nunca falta em uma linguagem de programação é a que controla as iterações e os cíclos. Maxima oferece aqui várias possibilidades. Em primeiro lugar,

```
for <var>:<val1> step <val2> thru <val3> do <expr>
```

O seguinte exemplo escreve, fazendo uso da função `print`, os cinco primeiros cubos inteiros positivos,

```
(%i4) for i:1 thru 5 do print(i^3);
1
8
27
64
125
(%o4)                         done
```

Outra possibilidade de controlar as iterações é com a versão

```
for <var>:<val1> step <val2> while <cond> do <expr>
```

que no seguinte exemplo se utiliza para calcular as quintas potências de todos os números impares menores que 20; note-se como trás a sentença "do"se pode escrever várias expressões separadas por vírgulas (,) e encerradas entre parêntesis,

```
(%i5) for i:1 step 2 while i<20 do(j:i^5,print(j));
1
243
3125
16807
59049
161051
371293
759375
1419857
2476099
(%o5)                         done
```

Também existe uma versão da sentença `for` mui a propósito para trabalhar com listas,

```
for <var> in <lista> do <expr>
```

como se mostra no seguinte exemplo, de onde se imprime todos os elementos de uma amostra simulada de números aleatórios, aumentados em uma unidade,

```
(%i7) m:makelist(random(4),i,1,10);
(%o7)          [2, 0, 3, 2, 2, 3, 0, 1, 1, 3]
(%i8) for i in m do print(i+1);
3
1
4
3
3
4
1
2
2
4
(%o8)                         done
```

Em geral, a definição de uma nova função em Maxima tem a estrutura

```
f(<arg1>,<arg2>,...):=<expr>
```

de onde <argi> são os argumentos e <expr> é uma expressão sintaticamente válida. Por exemplo, já que Maxima não dispõe da função logarítmica em base arbitrária, a podemos definir por nossa conta,

```
(%i9) logb(x,b):=log(x)/log(b)$
(%i10) logb(234,10);
                          log(234)
(%o10)                    --------
                          log(10)
(%i11) %,numer;
(%o11)                    2.3692157
```

Em algumas ocasiões, a função é o suficientemente complexa como para necessitar tanto de variáveis locais que guardem valores temporários, como de expressões que os calculem; em tais casos teremos que lançar mão do ambiente de bloco com a instrução block, cuja estrutura é

```
f(<arg1>,<arg2>,...):=block([<varloc1>,<varloc2>,...],
   <expr1>,
   <expr2>,
     ....
   <exprm> );
```

sendo o resultado devolvido o da última expressão avaliada (<exprm>). As variáveis locais às quais se fizeram referência se declaram entre colchetes (<varloci>) dentro do bloco, de onde podem ser inicializadas e cuja vida se extende durante o tiempo que dure o cálculo do bloco; além do mais, se uma destas variáveis temporárias se chama igual a outra global da sessão do Maxima, não interferirá com ela e qualquer referência à variável se considerará que é à local, e em caso de que esta não estiver declarada, a referência será à externa à função. A seguir, um exemplo no qual se define uma função que calcula a média de uma lista de números,

```
(%i12) media(lista):=block([n:length(lista),suma],
   if not listp(lista)
      then return("Atenção: não é uma lista"),
   suma: sum(lista[k],k,1,n),
   suma/n )$
(%i13) media([45,25,87,23,65,31]);
```

```
(%o13)                         46
(%i14) media([[2,3],[6,3],[4,7],[2,6]]);
                              7  19
(%o14)                       [-, --]
                              2  4
(%i15) media([1/3,sqrt(5),logb(5,2),x]);
                         log(5)             1
                    x + ------ + sqrt(5) + -
                         log(2)             3
(%o15)              ------------------------
                               4
(%i16) solve(%=10,x);  /* quanto vale x para a m\'edia 10? */
                3 log(5) + (3 sqrt(5) - 119) log(2)
(%o16)  [x = - ----------------------------------]
                              3 log(2)
```

Este último cálculo não tem nada a ver com o que se comenta nesta seção, porém mostra como integrar nossa função em uma sessão rotineira. Examinando o código da função `media` reparamos na presença da instrucção `return`, que é uma forma alternativa de sair do contexto marcado por `block`; neste caso, o resultado que devolve a função é o indicado pelo argumento de `return`, é dizer a cadeia `"Atenção: não é uma lista"`. Também se observa que se declaram duas variáveis locais, `n` e `suma`, atribuindo à primeira o número de elementos da lista; junto com estas existe outra variável local, a `k` da instrução `sum`, que só está ativa durante o cálculo desta soma, não sendo necessário declará-la em todo o contexto do bloco.

As vezes é necessário definir funções cujo número de argumentos não se conhece a priori. Por exemplo, tal como está programada a função `gcd` em Maxima, a que calcula o máximo divisor comum, não admite mais de dois argumentos; podemos suprir esta carência desenhando uma função, que chamaremos `mcd`, e que admita um número arbitrário de números,

```
(%i17) mcd(a,b,[c]):=block([n],
   n: length(c),
   r: gcd(a,b),
   if n>0 then
      for i in c do r: gcd(r,i),
   r )$
(%i18) mcd(120,300,480,825);
(%o18)                         15
```

Por último, se adiciona um exemplo no qual se define uma função fazendo uso de algumas das sentenças comentadas nesta seção. Como se vê, à função se lhe atribui o nome de `transafin` e sua ação é a de aplicar uma transformação afim a uma lista de pontos,

```
/*Assim se escrevem os comentários. */
/*pts deve ser uma lista de pares: */
/* [[x1,y1],[x2,y2],[x3,y3],...]   */
transafin(pts, a, b, c, d, e, f):=
  block([pts2, n, i, x, y],
    pts2: copylist(pts),
    n: length(pts2),
    for i:1 thru n do(
      x: pts2[i][1],
      y: pts2[i][2],
      pts2[i][1]: a*x+b*y+c,
      pts2[i][2]: d*x+e*y+f ),
    return(pts2) )$
```

A função `transafin`, e qualquer outra dentro desta seção, se pode escrever em Maxima ee logo chamá-la para que realize sua ação; sem impedimento, talvez seja melhor escrevê-lala diretamente com um editor de texto qualquer e guardá-la em um arquivo, ponhamos o nome `tf.mac`, para fazer uso dela no futuro; assim, uma vez dentro de Maxima executaríamos a instrução `batch("tf.mac")` e logo faríamos

```
(%i19) cuadrado:[[0,0],[1,0],[1,1],[0,1]]$
(%i20) transafin(cuadrado,-1,0,-1,0,1,1);
(%o20)      [[- 1, 1], [- 2, 1], [- 2, 2], [- 1, 2]]
```

obtendo assim o resultado de aplicá-la aos vértices do quadrado unitário uma simetría com relação ao eixho das ordenadas seguida de uma traslação ao longo do vetor $\vec{v} = (-1, 1)$.

20 GNU Free Documentation License

Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document "free"in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The **"Document"**, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as **"you"**. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A **"Modified Version"** of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A **"Secondary Section"** is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The **"Invariant Sections"** are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The **"Cover Texts"** are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A **"Transparent"** copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart ôr discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount ôf text. A copy that is not "Transparent"is called **"Opaque"**.

Êxamples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML ôr XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The **"Title Page"** means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page"means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section **"Entitled XYZ"** means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as **"Acknowledgements"**, **"Dedications"**, **"Endorsements"**, or **"History"**.) To **"Preserve the Title"** ôf such a section when you modify the Document means that it remains a section "Entitled XYZ"according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add não other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Ôpaque copy (directly or through your agents or retailers) of that êdition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy ôf it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is não section Entitled "History"in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History"section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements"or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements"or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain não material copied from the Document, you may at your option designate some or all ôf these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties–for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list ôf Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History"in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Êntitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all ôther respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate"if the copyright resulting from the compilation is not used to limit the legal rights ôf the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the êlectronic equivalent of covers if the Document is in electronic form. Ôtherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the ôriginal versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions ôf those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice ôr disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions ôf the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Êach version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version"applies to it, you have the option of following the terms and conditions either of that specified version or ôf any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright ©YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with não Invariant Sections, não Front-Cover Texts, and não Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts."line with this:

> with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.