Computação Matemática Elementar em Maxima

Delfim F. M. Torres Rita M. R. Pereira

Departamento de Matemática
Universidade de Aveiro
Janeiro de 2005

Conteúdo

Capítulo 1

Breve apresentação do Maxima

1.1 Introdução

O Maxima faz parte de uma família de ambientes computacionais apelidados de *Sistemas de Computação Algébrica*. Foi criado, na década de sessenta, pelo grupo de Matemática Aplicada e Computação do MIT (Massachusetts Institute of Technology). Inicialmente chamava-se Macsyma (MACs SYmbolic MAnipulator) e era um produto comercial. Mais tarde William Shelter, Professor na Universidade do Texas, conseguiu obter uma autorização para estudar e desenvolver o código do programa original, do qual fez um programa open-source com o nome de Maxima . Shelter manteve e melhorou o programa durante 15 anos. Após a sua morte, em 2001, um grupo de utilizadores entusiastas e programadores juntaram-se para dar continuidade ao projecto. Para uma breve história sobre o Maxima veja-se [4].

O Maxima é uma ferramenta matemática que pode ser usada para a manipulação de expressões algébricas envolvendo constantes, variáveis e funções. Permite também diferenciar, integrar, determinar soluções de equações lineares ou polinomiais, factorizar polinómios, expandir funções em séries de Laurent ou Taylor, fazer manipulação de matrizes, desenhar gráficos a duas e três dimensões, etc, etc. O Maxima vem também integrado com uma linguagem de programação de muito alto nível, que pode ser usada pelo utilizador para expandir as suas capacidades para além do conhecimento matemático já disponibilizado. Neste trabalho abordamos apenas algumas questões da matemática elementar. Para uma utilização mais avançada e técnica do Maxima remetemos o leitor interessado para [2].

Nesta secção mostramos, através de exemplos, algumas das facilidades disponibilizadas pelo Maxima. O objectivo é dar, desde já, uma ideia ao leitor das potencialidades da ferramenta e motivá-lo. Nas seguintes secções começamos verdadeiramente, passo a passo.

Depois de se iniciar uma sessão Maxima, o sistema oferece-nos uma "linha de comandos":

```
(C1)
```

O Maxima encontra-se então à espera de ordens. Comecemos por um exemplo simples:

```
(C1) 2+2;
(D1)                    4
(C2)
```

As etiquetas C e D , que correspondem aos valores de entrada e de saída, respectivamente, podem ser usadas para referir anteriores entradas ou resultados (na versão 5.9.1, as letras C

e D foram alteradas para `%i` e `%o`, respectivamente). As instruções em Maxima são lidas linha a linha, e terminam com ; ou $, caso se pretenda que o resultado seja imprimido ou não. Note-se que mesmo que o resultado não seja imprimido no écran, este é sempre atribuído à correspondente etiqueta de saída D. Vejamos algumas potencialidades do Maxima.

Façamos n tomar o valor de 70! (setenta factorial), isto é, o número que resulta do produto dos primeiros 70 inteiros positivos ($70! = 1 \cdot 2 \cdot 3 \cdots 69 \cdot 70$):

```
(C1) n : 70!;
(D1) 11978571669969891796072783721689098736458938142546425857555362864628009 5#
827898453196800000000000000000
```

Outra maneira de determinar o factorial é usando a função `factorial`:

```
(C2) factorial(70);
(D2) 11978571669969891796072783721689098736458938142546425857555362864628009 5#
827898453196800000000000000000
```

Decomponhamos agora 70! em factores primos (atenção, o Maxima é sensível às minúsculas-maiúsculas para o nome das variáveis):

```
(C3) factor(n);
(D3)
```

$$2^{67}\,3^{32}\,5^{16}\,7^{11}\,11^{6}\,13^{5}\,17^{4}\,19^{3}\,23^{3}\,29^{2}\,31^{2}\,37\,41\,43\,47\,53\,59\,61\,67$$

Vejamos quais os primeiros 200 dígitos de π:

```
(C4) ev(%PI,BFLOAT,FPPREC:200);
(D4) 3.14159265358979323846264338327950288419716939937510582097494459230781 64#

062862089986280348253421170679821480865132823066470938446095505822317253594081#

28481117450284102701938521105559644622948954930381 9B0
```

Podemos achar a expansão de expressões como $(a + \sqrt{b})^{15}$:

```
(C5) expand((a + sqrt(b))^15);
(D3)
```

$$b^{\frac{15}{2}} + 15\,a\,b^{7} + 105\,a^{2}\,b^{\frac{13}{2}} + 455\,a^{3}\,b^{6} + 1365\,a^{4}\,b^{\frac{11}{2}} + 3003\,a^{5}\,b^{5}$$
$$+5005\,a^{6}\,b^{\frac{9}{2}} + 6435\,a^{7}\,b^{4} + 6435\,a^{8}\,b^{\frac{7}{2}} + 5005\,a^{9}\,b^{3} + 3003\,a^{10}\,b^{\frac{5}{2}}$$
$$+1365\,a^{11}\,b^{2} + 455\,a^{12}\,b^{\frac{3}{2}} + 105\,a^{13}\,b + 15\,a^{14}\,\sqrt{b} + a^{15}$$

calcular o valor de somatórios

```
(C6) s: sum(2^i + i^2,i,0,k);
(D6)
```

$$\sum_{i=0}^{k} 2^{i} + i^{2}$$

```
(C7) ev(%,simpsum);
(D7)
```

$$2^{k+1} + \frac{2\,k^3 + 3\,k^2 + k}{6} - 1$$

ou então:

```
(C8) s: i^3*7^i$
(C9) sum(s,i,0,k);
(D9)
```

$$\sum_{i=0}^{k} i^3\,7^i$$

```
(C10) nusum(s,i,0,k);
(D10)
```

$$\frac{7\,\left(36\,k^3 - 18\,k^2 + 24\,k - 13\right)\,7^k}{216} + \frac{7\,13}{2^3\,3^3}$$

Muitas outras operações são possíveis. Vamos dar apenas mais alguns exemplos ilustrativos. Podemos determinar o limite

$$\lim_{x\to\infty} \frac{3x - 1}{x\arctan(x) + \ln(x)}$$

```
(C11) tlimit((3*x-1)/(x*atan(x)+log(x)),x,inf);
(D11)
```

$$\frac{6}{\pi}$$

a derivada de $\cos\left(x^3 \ln(1 - x^5)\right)$

```
(C12) diff(cos(x^3*log(1-x^5)),x);
(D12)
```

$$-\left(3\,x^2\,\log\left(1 - x^5\right) - \frac{5\,x^7}{1 - x^5}\right)\,\sin\left(x^3\,\log\left(1 - x^5\right)\right)$$

primitivas

```
(C13) p:integrate(x^3 * sqrt(x^4 - a^4),x)$
(C14) p;
(D14)
```

$$\frac{\left(x^4 - a^4\right)^{\frac{3}{2}}}{6}$$

ou então:

```
(C15) q:x^3 * sqrt(x^4 - a^4)$
(C16) 'integrate(q,x);
(D16)
```

$$\int x^3\,\texttt{SQRT(x\^{}4-a\^{}4)}\;dx$$

```
(C17) integrate(q,x);
(D17)
```

$$\frac{\left(x^4-a^4\right)^{\frac{3}{2}}}{6}$$

Podemos igualmente resolver sistemas de equações lineares. Por exemplo,

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

```
(C18) equacoes: [5*x-3*y=2*z+1,-x+4*y=7*z,3*x+5*y=z]$
(C19) solve(equacoes,[x,y,z]);
(D19)
```

$$\left[\left[x=\frac{31}{255},y=-\frac{22}{255},z=-\frac{1}{15}\right]\right]$$

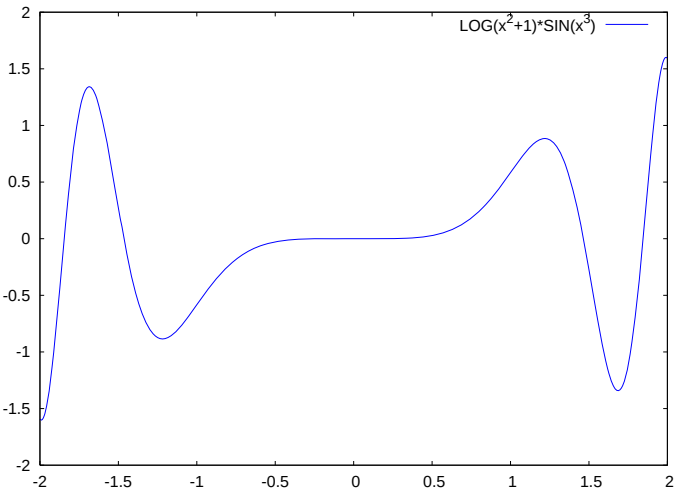
ou ainda:

```
(C20) linsolve(equacoes,[x,y,z]);
(D20)
```

$$\left[x=\frac{31}{255},y=-\frac{22}{255},z=-\frac{1}{15}\right]$$

Por último esboçamos o gráfico da função $\sin(x^3)=\ln(1+x^2)$, $x\in[-2,2]$. Antes de fazer um gráfico é conveniente escolher a opção 'Separate', para que o gráfico apareça numa janela à parte. Esta opção selecciona-se indo ao menu `Options` > `Plot Windows`.

```
(C21) plot2d(sin(x^3)*log(1+x^2),[x,-2,2]);
```



Pode-se fazer também:

```
(C22) plot2d(sin(x^3)*log(1+x^2),[x,-2,2],[plot_format,openmath]);
```

1.2 Expressões Aritméticas

A aritmética em $\mathbb{Z}$ é trivial com o Maxima.

```
(C1) 5+3;
(D1)                          8

(C2) 5
                     +
        3
            ;
(D2)                          8
```

São possíveis várias operações em simultâneo:

```
(C3) 1+10+100+1000;
(D3)                        1111
```

Os caracteres `+ - * /` denotam as quatro operações aritméticas básicas: adição, subtracção, multiplicação e divisão, respectivamente.

Uma expressão pode conter um número arbitrário de operadores, o que pode criar um problema de representação. Por exemplo o significado das expressões Maxima

```
(C4) 8+4+2$
(C5) 8*4*2$
```

é claro. Mas o que significa a seguinte expressão Maxima?

```
(C6) 8/4/2$
```

Traduzindo literalmente para a notação matemática, obtemos um objecto bizarro:

$$\frac{8}{\frac{4}{2}}$$

A ambiguidade surge porque a divisão, ao contrário da adição e multiplicação, e tal como a subtracção, não é associativa: $(a/b)/c$ é, em geral, diferente de $a/(b/c)$.

```
(C7) (8/4)/2;
(D7)                          1

(C8) 8/(4/2);
(D8)                          4
```

O Maxima realiza sempre a divisão da esquerda para a direita:

```
(C9) 8/4/2;
(D9)                          1
```

O Maxima é muito tolerante a parêntesis redundantes:

```
(C10) (((((8/((((4))))))))))/(((2)));
(D10)                          1
```

Apenas parêntesis curvos são permitidos. Os parêntesis rectos e as chavetas têm, como veremos mais tarde, um significado especial em Maxima e, por isso, não podem ser usados para colocar expressões entre parêntesis.

O operador exponenciação é representado pelo caracter `^` ou `**` . Por exemplo `2^14` significa 2^{14}. A exponenciação não é associativa:

```
(C11) 2^2^3;
(D11)                         256
(C12) 2^(2^3);
(D12)                         256
(C13) (2^2)^3;
(D13)                         64
```

O Maxima não é uma calculadora normal. Isso torna-se claro quando calculamos algo como

```
(C13) 2^1279-1;
```

e recebemos como resposta um número com 386 dígitos. Como o resultado não cabe em apenas uma linha, o Maxima usa o caracter `#` no resultado para indicar a continuação na linha seguinte. Para o valor de entrada, não existe em Maxima caracter de continuação de linha:

```
(C14) 100
00+1;
Incorrect syntax: 0 is not an infix operator
100\n00+
      ^
(C14)
(D14)                          1
```

mas podemos separar operações por várias linhas:

```
(C15) 10000
+1;
(D15)                        10001
```

A expressão `2^1279-1` acima contém dois operadores distintos: exponenciação e subtracção. Qual das duas possibilidades é considerada pelo Maxima: $2^{1279}-1$ ou $2^{1279-1} = 2^{1278}$? O Maxima dá prioridade à exponenciação:

```
(C16) 3-2^5;
(D16)                        - 29
(C17) (3-2)^5;
(D17)                          1
```

No caso de existir mais do que um operador, a ordem de cálculo da expressão é:

(1) exponenciação;

(2) multiplicação, divisão;

(3) adição, subtracção.

Se aparecer mais do que um operador com a mesma prioridade, o cálculo é realizado da esquerda para a direita para as operações dos grupos (2) e (3). Sequências do operador exponenciação são realizadas da direita para a esquerda. Uma sub-expressão entre parêntesis é sempre realizada em primeiro lugar e, se existir mais do que uma, com prioridade da esquerda para a direita.

Exemplo 1. *Uma expressão Maxima correspondente à expressão matemática*

$$\frac{2^{3\times 4-5}+6}{7\times 8} \tag{1.1}$$

é obtida usando, pelo menos, três parêntesis:

```
(C18) (2^(3*4-5)+6)/(7*8);
(D18)
```

$$\frac{67}{28}$$

1.3 Variáveis e Comentários

O caracter `%` pode ser usado em qualquer expressão para representar o valor do resultado do último comando (tem, por isso, um significado completamente distinto da percentagem das máquinas de calcular!):

```
(C1) 51^2+80^2-1;
(D1)                          9000
(C2) %/1000;
(D2)                             9
```

Reparar que no último comando não é necessário parêntesis: é o valor que é substituído e não a expressão. Uma outra maneira de obter resultados anteriores é através das etiquetas `D`. Por exemplo:

```
(C3) D1/1000;
(D3)                             9
```

Exemplo 2. *O valor da expressão* $(12^3+1^3)-(10^3+9^3)$ *pode ser calculado com recurso às etiquetas D da seguinte maneira (repare que não é usado qualquer parêntesis):*

```
(C4) 12^3+1^3$
(C5) 10^3+9^3$
(C6) D4-D5;
(D6)                             0
```

Exemplo 3. *O valor da expressão aninhada*

$$((((1+1)\times 2+1)\times 3+1)\times 4+=1)\times 5$$

pode ser calculado da seguinte maneira:

```
(C7) 1$ (%+1)*2$ (%+1)*3$ (%+1)*4$ (%+1)*5;
(C8)
(C9)
(C10)
(C11)
(D11)                        325
```

O exemplo anterior mostra que são possíveis vários comandos numa mesma linha de entrada.

Os comentários são inseridos entre os caracteres `/*` e `*/`.

```
(C12) 1+1; /* isto deve dar dois */
(D12)                         2
```

Pode-se inseridos comentários na saída usando aspas:

```
(C13) "um mais um são dois"; 1+1; /* isto é um comentário*/
(D13)                um mais um são dois
(C14)
(D14)                         2
```

Em Maxima, os nomes das variáveis devem começar por uma letra ou `_`, e podem depois ser seguidos por letras, `_` ou `%`. O Maxima é sensível a minúsculas-maiúsculas, pelo que `AA` e `aa` são nomes de variáveis diferentes.

Vejamos alguns exemplos:

```
(C15) primeiro_quadrado : 15140424455100^2$
(C16) SegundoQuadrado : 158070671986249^2$
(C17) 109 * primeiro_quadrado - SegundoQuadrado;
(D17)                        - 1
```

Não podem ser usadas algumas variáveis e funções cujo nome coincide com uma palavra reservada do Maxima (e.g., `and from for else unless`):

```
(C18) and: 3;Incorrect syntax: AND is not a prefix operator
and:
  ^
```

Em Maxima, a constante π é representada por `%PI`.

```
(C19) %PI:3;
%PI improper value assignment to a numerical quantity
 -- an error.  Quitting.  To debug this try DEBUGMODE(TRUE);
(C20) pi:3;  /* neste caso, 'pi' é o nome de uma variável qualquer */
(D20)                         3

(C21) ev(bfloat(%PI),fpprec=5);
(D21)                    3.1416B0
```

O comando `kill` permite que uma dada variável deixe de ter um valor atribuído. Outro comando útil é `values` que permite ver a que variáveis foram atribuídos valores:

```
(C22) a:10$
(C23) b:6$
(C24) c:a+b;
(D24)                          16
(C25) values;
(D25)                     [a, b, c]
(C26) kill(c)$
(C27) values;
(D27)                      [a, b]
```

Uma variável também volta ao seu estado indeterminado através do uso de aspas:

```
(C28) a:3$
(C29) 2-a^a;
(D29)                          - 25
(C30) a : "a"$
(C31) 2-a^a;
(D31)
```

$$2 - a^a$$

```
(C32) y:3$
(C33) y + "y";
(D33)                          y + 3
```

Atenção ao uso de definições recursivas. O comando seguinte não causa problemas

```
(C34) perigosa: perigosa + 1$
(C35) perigosa;
(D35)                   perigosa + 1
```

mas o próximo sim!

```
(C36) perigosa(x):= perigosa(x) + 1$
(C37) perigosa(x);
Maxima encountered a Lisp error:

 Error in CATCH [or a callee]: Bind stack overflow.

Automatically continuing.
To reenable the Lisp debugger set *debugger-hook* to nil.
```

1.4 Substituições

A função do Maxima `subst` permite substituir sub-expressões numa expressão:

```
(C1) s: x+y+x^2+y^2+x*y;
(D1)
```

$$y^2 + x\,y + y + x^2 + x$$

```
(C2) subst([x=100,y=99],s);
(D2)                         29900
```

```
(C3) s;
(D3)
```

$$y^2 + x\,y + y + x^2 + x$$

Outra maneira de fazer substituições é através da função `ev` (a atribuição dos valores das variáveis x e y pode ser feita com `:` ou `=`):

```
(C4) ev(s,x=100,y=99);
(D4)                        29900

(C5) subst(x=1,s) - subst(y=z^2,s);
(D5)
```

$$-z^4 - x\,z^2 - z^2 + y^2 + 2\,y - x^2 - x + 2$$

ou:

```
(C6) ev(s,x=1) - ev(s,y=z^2);
```

Existem dois tipos de substituições: as substituições sequenciais (em que a ordem importa)

```
(C7) subst([a=b,b=c],a+b+c);
(D7)                          3 c
(C8) subst([b=c,a=b],a+b+c);
(D8)                        2 c + b
```

e as substituições múltiplas em que a ordem é irrelevante. Vejamos a diferença:

```
(C9) subst([a=b,b=c,c=a],a^3+b^2+c); /* substituição simultânea */
(D9)
```

$$a^3 + a^2 + a$$

```
(C10) sublis([a=b,b=c,c=a],a^3+b^2+c); /* substituição múltipla */
(D10)
```

$$c^2 + b^3 + a$$

ou

```
(C11) ev(a^3+b^2+c,[a=b,b=c,c=a]); /* substituição múltipla */
(D11)
```

$$c^2 + b^3 + a$$

Exemplo 4. *O valor de*

$$4\left(\frac{1}{a^2} - a\right)\left(\frac{1}{b^4} + b^2\right) - = \frac{9a}{\frac{1}{(b-1)^2} - b + 1}$$

para $a = 2$ *e* $b = -1$ *pode ser calculado da seguinte maneira:*

```
(C12) 1/a^2-a$
(C13) 4*%*subst(a=-b^2,%) - 9*a/subst(a=b-1,%)$
(C14) subst([a=2,b=-1],%);
(D14)                        - 22
```

Capítulo 2

Expressões Relacionais

2.1 Operadores Relacionais

Problema 1. *Qual dos dois é maior:* 2^{1000} *ou* 3^{600}*?*

Uma maneira óbvia de resolver o problema é através da função `SIGN(exp)`, que devolve uma das seguintes respostas: `POS` (positivo), `NEG` (negativo), `ZERO`, `PZ` (positivo ou zero), `NZ` (negativo ou zero), `PN` (positivo ou negativo), ou `PNZ` (positivo, negativo, ou zero, i.e., nada conhecido).

```
(C1) sign(2^1000 - 3^600);
(D1)                       POS
```

Como o resultado é positivo (POS), concluímos que $2^{1000} > 3^{600}$. O Maxima oferece a possibilidade de obter respostas directas a questões deste tipo, através da função `ev`:

```
(C2) ev(2^1000 > 3^600, pred);
(D2)                      TRUE
```

Também se pode usar a função `is`:

```
(C2) is(2^1000 > 3^600);
(D2)                      TRUE
```

A expressão `2^1000 > 3^600` é um exemplo de uma *expressão relacional*: uma expressão que liga dados numéricos e lógicos. De uma maneira geral, uma expressão relacional consiste de duas expressões aritméticas que são comparadas por meio de um *operador relacional*. Existem duas possibilidades:

1. A relação é satisfeita: o Maxima retorna *true*;
2. A relação não é satisfeita: o Maxima retorna $false$;

Os operadores relacionais do Maxima mais usados são:

```
=    #(diferente)
<    >
<=   >=
```

Para calcular o valor de uma expressão relacional é necessário usar a função `ev`

```
(C3) ev(-3 < -2,pred); ev(-3 = -2,pred);
(D3)                    TRUE
(C4)
(D4)                   FALSE
```

ou

```
(C5) is(-3 < -2); is(-3 = -2);
(D5)                    TRUE
(C6)
(D6)                   FALSE
```

Para calcular distâncias usamos a função *valor absoluto* que é implementada em Maxima por `abs`

```
(C7) abs(-12); abs(53*9100^2 - 66249^2);
(D7)                    12
(C8)
(D8)                     1
```

Exemplo 5. *Seja* $n = 100^{100}$. *Qual dos números* $n_1 = 101^{99}$ *ou* $n_2 = 99^{101}$ *está mais próximo de* n*?*

```
(C9)  n: 100^100$
(C10) n1: 101^99$
(C11) n2: 99^101$
(C12) ev(abs(n-n1)<abs(n-n2),pred);
(D12)                   TRUE
```

ou, usando a função `orderlessp`*:*

```
(C13) orderlessp(abs(n-n1),abs(n-n2));
(D12)                   TRUE
```

Concluímos então que n_1 *está mais próximo de* n *do que* n_2.

2.2 Divisibilidade

Dados dois inteiros d e n dizemos que d divide n (ou que d é divisor de n; ou que n é múltiplo de d) se existir um inteiro q tal que $n = dq$. A q chamamos quociente da divisão de n por d. Quando d divide n escrevemos

$$d|n$$

Podemos então escrever, por exemplo, que $3|21$. Um inteiro é par se é divisível por 2; ímpar se não.

Se $d|n$ então $-d|n$. Por esta razão é usual considerar apenas os divisores positivos. Um *divisor próprio* d de n é um divisor diferente de 1 e diferente de n.

Exemplo 6. *O* 0 *tem infinitos divisores; 1 tem 1 divisor; 12 tem 6 divisores (1,2,3,4,6,12) – quatro dos quais são próprios; 11 tem dois divisores (nenhum divisor próprio).*

A divisibilidade tem uma interpretação geométrica simples: se $d|n$ então podemos organizar n pontos no plano de forma a formar um array rectangular com d linhas. Se mudarmos linhas e colunas (se rodarmos o array de 90^o) obtemos um novo array com $q = \frac{n}{d}$ linhas. Desta interpretação geométrica concluímos que os divisores vêm aos pares: $d|n \Rightarrow \left(\frac{n}{d}\right)|n$.

Uma vez que a cada divisor d de n corresponde o *divisor gémeo* $\frac{n}{d}$, podemos concluir que o número de divisores de um inteiro é par? A resposta é *não*. Isto pelo simples facto que d e $\frac{n}{d}$ podem coincidir. Isto acontece quando n é um quadrado.

Teorema 1. *Um inteiro é um quadrado se, e somente se, tem um número ímpar de divisores.*

O emparelhamento dos divisores tem uma implicação importante. Suponhamos que $d|n$ com $d^2 \leq n \Leftrightarrow d \leq \sqrt{n}$. Então $\frac{n}{d^2} \geq 1$ e concluímos que $\left(\frac{n}{d}\right)^2 \geq n \Leftrightarrow \frac{n}{d} \geq \sqrt{n}$. Isto significa que para encontrarmos todos os divisores de n apenas precisamos de testar a divisibilidade dos inteiros d para os quais $1 \leq d \leq \sqrt{n}$.

Exemplo 7. *Para encontrar todos os divisores de 30 testamos a divisibilidade para* $d = 1, 2, 3, 4, 5$. *Os divisores são* $(1, 30)$, $(2, 15)$, $(3, 10)$ *e* $(5, 6)$. *Para encontrar os divisores de 36 testamos a divisibilidade para* $d = 1, 2, 3, 4, 5, 6$. *Os divisores de 36 são* $(1, 36)$, $(2, 18)$, $(3, 12)$, $(4, 9)$ *e 6 (número ímpar de divisores porque* $36 = 6^2$*).*

2.3 Quociente e Resto

Para todo o $d \in \mathbb{N}$ e $n \in \mathbb{N}_0$ existe um único q e um único r tais que

$$n = dq + r\,, \qquad 0 \leq r < d\,.$$

O q é chamado de *quociente* e r de *resto*.

Um divisor comum a dois inteiros é um inteiro que divide ambos. Por exemplo, como 3|12 e 3|21 então 3 é divisor comum a 12 e 21. Tem especial interesse o *máximo divisor comum* entre dois inteiros x e y, denotado usualmente por $gcd(x, d)$ ("greatest common divisor"), que é o maior número entre os divisores comuns de x e y. Por definição, $gcd(x, y) = gcd(|x|, |y|)$; $gcd(x, 0) = |x|$; $gcd(0, 0) = 0$. Existe uma relação íntima entre o conceito de divisibilidade e o conceito de *gcd*.

Teorema 2. *Se* $0 < a \leq b$, *então* $a|b \Leftrightarrow gcd(a, b) = a$.

A definição de máximo divisor comum é extendida a mais do que dois inteiros da maneira óbvia. Dizemos que os inteiros $x_1, \ldots, x_n$ são *primos entre si* quando o $gcd(x_1, \cdots, x_n) = 1$.

Muito importante também, é o conceito de *mínimo múltiplo comum* entre x e y, denotado usualmente por $lcm(x, y)$ ("least common multiple"): o menor inteiro positivo divisível por x e y.

2.4 Funções do Maxima associadas à divisibilidade

O máximo divisor comum e mínimo múltiplo comum estão implementados em Maxima pelas seguintes funções:

`gcd(p1, p2, var1,...)` - calcula o máximo divisor comum de p1 e p2;

`ezgcd(p1, p2,...)` - devolve uma lista cujo primeiro elemento é o m.d.c. dos polinómios p1, p2,..., e os restantes elementos são os polinómios divididos pelo m.d.c.;

`lcm(exp1, exp2,...)` - calcula o mínimo múltiplo comum dos seus argumentos. Fazer `LOAD(FUNCTS);` para aceder a esta função.

```
(C1) gcd(-15,21);
(D1)                          3
(C2) gcd(1,-7);
(D2)                          1
(C3) gcd(0,7);
(D3)                          7
```

Os números 14 e 15 são primos entre si:

```
(C4) gcd(14,15);
(D4)                          1
```

Repare que 10, 15 e 18 são primos entre si embora nenhum par formado entre eles seja primo entre si:

```
(C5) ezgcd(10,15,18);
(D5)                      [1, 10, 15, 18]
(C6) gcd(10,15);
(D6)                          5
(C7) gcd(10,18);
(D7)                          2
(C8) gcd(15,18);
(D8)                          3
```

A função `ezgcd` devolve uma lista, em que o primeiro elemento é o m.d.c. Se pretendermos obter apenas este valor, podemos usar a função `part` para nos devolver um elemento da lista, ou seja:

```
(C9) part(ezgcd(10,15,18),1);
(D9)                          1
```

Pelo Teorema 2 podemos usar o `gcd` para testar a divisibilidade.

Exemplo 8. $2^{191}-1$ *é divisível por 383? Uma maneira de responder à questão é através do seguinte comando Maxima:*

```
(C10) ev(gcd(383,2^191-1) = 383,pred);
(D10)                        TRUE
```

ou, através do comando:

```
(C11) is(gcd(383,2^191-1) = 383);
(D11)                        TRUE
```

O mínimo múltiplo comum entre 12 e 21 é 84. Não esquecer de escrever o comando `LOAD(FUNCTS)` para poder aceder à função `lcm`.

```
(C12) LOAD(FUNCTS)$
(C13) lcm(12,21);
(D13)                        84
```

O mínimo múltiplo comum entre x (x inteiro arbitrário) e zero é zero:

```
(C14) lcm(x,0);
(D14)                         0
```

O quociente e resto da divisão inteira são obtidos, respectivamente, pelas funções `quotient` e `remainder`. As funções do Maxima também aceitam argumentos negativos. A definição é:

$$n = dq + r\,, \qquad 0 \leq |r| \leq |d|\,, \quad nr \geq 0\,.$$

O resto é negativo sempre que n é negativo; o quociente é negativo se n e d têm sinais opostos:

```
(C14) quotient(23,7);
(D14)                         3
(C15) quotient(23,-7);
(D15)                        - 3
(C16) quotient(-23,7);
(D16)                        - 3
(C17) quotient(-23,-7);
(D17)                         3

(C18) remainder(23,7);
(D18)                         2
(C19) remainder(23,-7);
(D19)                         2
(C20) remainder(-23,7);
(D20)                        - 2
(C21) remainder(-23,-7);
(D21)                        - 2
```

Outra forma de obter o quociente e o resto é através da função `divide` cujo resultado é uma lista em que o primeiro elemento é o quociente e o segundo elemento é o resto da divisão.

```
(C22) divide(23,7);
(D22)                      [3, 2]
(C23) divide(23,-7);
(D23)                     [- 3, 2]
(C24) divide(-23,7);
(D24)                    [- 3, - 2]
(C25) divide(-23,-7);
(D25)                     [3, - 2]
```

Podemos também usar a função `remainder` para resolver o problema do Exemplo 8:

```
(C26) ev(remainder(2^191-1,383) = 0,pred);
(D26)                       TRUE
```

ou

```
(C26) is(remainder(2^191-1,383) = 0);
(D26)                       TRUE
```

2.5 Racionais

O conjunto $\mathbb{Z}$ não é fechado sob a divisão: dados $a, b \in \mathbb{Z}$, $b \neq 0$, em geral não é verdade que $\frac{a}{b} \in \mathbb{Z}$. Somos assim levados à introdução do corpo $\mathbb{Q}$ dos números racionais:

$$\mathbb{Q} = \left\{ x = \frac{b}{a} : a, b \in \mathbb{Z} \text{ e } a \neq 0 \right\} .$$

A definição de número racional cria um problema de representação: existem infinitos pares de inteiros (a, b) que representam o mesmo racional. Por exemplo,

$$x = \frac{-2}{3} = \frac{2}{-3} = \frac{-4}{6} = \frac{4}{-6} = \frac{-6}{9} = \frac{6}{-9} = \cdots$$

Para resolver este problema de representação, o Maxima usa a chamada *forma reduzida*: dizemos que $\frac{b}{a}$ está na forma reduzida quando a é positivo e a e b são primos entre si. Por exemplo a forma reduzida do racional $\frac{4}{-6}$ é $\frac{-2}{3}$:

```
(C1) 4/(-6);
(D1)                          -2
                              --
                              3
```

As funções `num` e `denom` permitem-nos aceder, respectivamente, ao numerador e denominador de um racional:

```
(C2) 22/8$
(C3) num(C2);
(D3)                        11
(C4) denom(C2);
(D4)                         4
```

Todo o inteiro é racional.

```
(C5) denom(3);
(D5)                         1
```

Os racionais constituem um novo tipo de objecto em Maxima – um novo tipo de dados.

```
(C6) integerp(2);
(D6)                       TRUE
(C7) ratnump(2/3);
(D7)                       TRUE
```

Exemplo 9. *Pretendemos responder à seguinte questão:* 1111|111111111111*? Começamos por definir os números inteiros em jogo:*

```
(C8) n: 111111111111$
(C9) d: 1111$
```

Vejamos três maneiras de resolver o problema:

```
(C10) ev(remainder(n,d)=0,pred); /* maneira 1 */
(D10)                     TRUE

(C11) integerp(n/d); /* maneira 2 */
(D11)                     TRUE

(C12) ev(denom(n,d)=1,pred); /* maneira 3 */
(D12)                     TRUE
```

Dado $x = \frac{b}{a} \in \mathbb{Q}$ podemos encontrar q e r tal que

$$b = qa + r$$

Dividindo por a obtemos:

$$\frac{b}{a} = q + \frac{r}{a}$$

A q chamamos *parte inteira de* x e a $\frac{r}{a}$ *parte fraccionária de* x. A parte fraccionária de x é denotada por $\{x\}$. Por exemplo,

$$\frac{23}{7} = 3 + \frac{2}{7} \Rightarrow \left\{\frac{23}{7}\right\} = \frac{2}{7}$$

A função Maxima `entier(X)` devolve-nos o maior inteiro menor ou igual a X:

```
(C13) entier(23/7);
(D13)                      3
```

Se pretendermos determinar a parte fraccionária de um racional podemos fazer:

```
(C13) (23/7) - entier(23/7);
(D13)                      2
                           -
                           7
```

Este comando pode ser definido como função da forma `frac(x) := x - entier(x)`, para se usar mais facilmente.

Exemplo 10. *Por definição, a função* `frac(x)` *pode ser definida pelo comando Maxima* `remainder(num(x),denom(x))/denom(x);`

Exemplo 11. *Seja* $r_1 = \frac{21}{34}$, $r_2 = \frac{55}{89}$, $r_3 = \frac{34}{55}$. *Pretende-se mostrar que* r_2 *está entre* r_1 *e* r_3. *Isso é conseguido por intermédio dos seguintes comandos Maxima:*

```
(C14) r1:21/34$
(C15) r2:55/89$
(C16) r3:34/55$
(C17) ev(abs(r1-r3) = abs(r1-r2) + abs(r2-r3),pred);
(D17)                     TRUE
```

ou

```
(C17) is(abs(r1-r3) = abs(r1-r2) + abs(r2-r3));
(D17)                        TRUE
```

A distância mínima entre dois inteiros é um. Em particular, 1 é o inteiro positivo mais pequeno. Não existe distância mínima entre dois racionais e, como consequência, não existe "o racional positivo mais pequeno".

Problema 2. *Determinar um racional a uma distância de* $x = \frac{a}{b}$ *(racional dado) inferior a* ε*.*

Vejamos uma maneira de abordar o Problema 2. Para qualquer $m > 0$, $x = \frac{a}{b} = \frac{am}{bm}$ e os racionais $\frac{am+1}{bm}$ e $\frac{am-1}{bm}$ estão a uma distância $\frac{1}{bm}$ de x. Escolhendo m suficientemente grande, podemos fazer esta distância tão pequena quanto queiramos.

Exemplo 12. *Pretendemos encontrar um racional a uma distância de* $x = \frac{96}{145}$ *inferior a* 10^{-4}*. Para isso encontramos o menor* m *para o qual* $bm > 10^4$*.*

```
(C18) a:96$
(C19) b:145$
(C20) m:quotient(10^4,b)+1$
(C21) (a*m+1)/(b*m);
(D21)
```

$$\frac{1325}{2001}$$

2.6 Primos

Um inteiro positivo $n > 1$ diz-se primo se ele tem precisamente 2 divisores: 1 e n (ou seja, se não tiver divisores próprios). Um número não primo diz-se composto. O *Teorema Fundamental da Aritmética* diz que qualquer inteiro n maior que 1 pode ser expresso como um produto

$$n = p_1^{e_1} \times p_2^{e_2} \times \cdots \times p_k^{e_k} \tag{2.1}$$

onde os p_i's são números primos distintos e os e_i's são inteiros positivos. Mais, afirma que esta factorização é única a menos da ordem dos factores. Por exemplo, $12 = 2^2 \times 3$. Por conveniência, não consideramos 1 como número primo (de outro modo, $12 = 1 \times 2^2 \times 3$ era uma decomposição em primos diferente!). O número de divisores de n é dado por

$$\sigma(n) = (e_1 + 1) \times (e_2 + 1) \times \cdots \times (e_k + 1)$$

Por exemplo, $84 = 2^2 \times 3^1 \times 7^1$ ($p_1 = 2$, $p_2 = 3$, $p_3 = 7$, $e_1 = 2$, $e_2 = 1$, $e_3 = 1$). Concluímos então que 84 tem 12 divisores: $\sigma(84) = 3 \times 2 \times 2 = 12$.

Exemplo 13. *Pretende-se encontrar a estrutura de todos os inteiros* n *com 10 divisores. De modo a que* $(e_1 + 1) \times (e_2 + 1) \times \cdots \times (e_k + 1) = 10 = 2 \times 5$ *temos duas possibilidades: ou* $k = 1 \wedge e_1 = 9 \Rightarrow n = p_1^9$*; ou* $k = 2 \wedge e_1 = 1 \wedge e_2 = 4 \Rightarrow n = p_1 \times p_2^4$*.*

Em Maxima a factorização (2.1) é obtida através da função `factor`:

```
(C1) factor(3^52-2^52);
(D1)
```

$$5\ 13^2\ 53\ 79\ 1093\ 4057\,29927\ 13761229$$

Exemplo 14. *Pretende-se determinar se 31418506212244678577 é, ou não, primo. Uma maneira muito ineficiente de saber a resposta consiste em usar o* `factor`*:*

```
(C2) factor(31418506212244678577);
(D2)
```

$$7919\ 7927\ 7933\ 7937\ 7949$$

```
(C3) expand(%);
(D3)                  31418506212244678577
```

É possível decidir se um número é primo ou não sem calcular a sua factorização! (A factorização (2.1) *é computacionalmente muito exigente, razão pela qual é muito usada nos métodos de criptografia.) Em* Maxima *usamos a função* `primep`*:*

```
(C4) primep(31418506212244678577);
(D4)                      FALSE
```

2.7 Funções aritméticas do Maxima

Não deixe de explorar os vários comandos com a ajuda dos manuais online. Boas digressões com o Maxima!

```
abs(x)
n! ou factorial(n)
binomial(x,y)
isqrt(n)
max(x1,x2,...)
min(x1,x2,...)
quotient(p1,p2,var1,...,varn)
remainder(p1,p2,var1,...)
gcd(p1,p2,var1,...) / ezgcd(p1,p2,...)
lcm(exp1,exp2,...) (fazer LOAD(FUNCTS);)
factor(exp)
primep(int)
num(exp)
denom(exp)
entier(x)
```

Capítulo 3

Conjuntos e Funções em Maxima

3.1 Conjuntos

Um conjunto é uma colecção de objectos. A ordem na qual os elementos são listados é irrelevante e, por conseguinte, dois conjuntos são iguais se contêm os mesmos elementos.

Para se poder trabalhar com conjuntos em Maxima, é preciso instalar-se o "pacote" `nset`. Este pode ser obtido em [5].

Depois de instalado, deve-se fazer:

```
(C1) load(nset)$
```

Para se construir um conjunto com elementos $a1$, $a2$, ..., an, usa-se o comando

```
set(a1,a2,...,an)
```

Por exemplo,

```
(C2) T: set(1,3,-4)$
(C3) setp(T);
(D3)                          TRUE
```

Os conjuntos são representados no resultado da maneira usual:

```
(C4) T;
(D4)                    {- 4, 1, 3}
```

Se se pretender trabalhar, nos valores de entrada, com conjuntos usando a representação `{ }`, pode-se fazer escrevendo os seguintes comandos (ver [5]):

```
(C5)  matchfix("{","}")$
(C6)  "{"([a]) := apply(set,a)$
```

Desta forma, já podemos definir conjuntos usando chavetas, isto é

```
(C7) {1,3,-4};
(D7)                    {- 4, 1, 3}
```

O Maxima elimina os elementos repetidos num conjunto

```
(C8) {7,7,0,3,7};
(D8)                    {0, 3, 7}
```

e pode reordenar os elementos (num conjunto não existe a noção de ordem):

```
(C9) {-4,3,1};
(D9)                   {- 4, 1, 3}
```

Para verificar quando um elemento pertence, ou não, a um conjunto, usamos a função `elementp`, que é uma função Booleana.

```
(C10) U : {a,{a,b}}$
(C11) elementp(a,U);
(D11)                         TRUE
(C12) elementp(b,U);
(D12)                        FALSE
(C13) elementp({b,a},U);
(D13)                         TRUE
```

O conjunto vazio $\emptyset$ é representado em Maxima através de `set()`, ou com o abrir e fechar de chavetas.

```
(C14) vazio : {}$
```

Para contar o número de elementos de um conjunto, usamos a função `cardinality`, que devolve o número de elementos (distintos) de um conjunto

```
(C15) cardinality(U);
(D15)                          2
(C16)cardinality(vazio);
(D16)                          0
```

ou, a função `length`

```
(C15) length(U);
(D15)                          2
```

O Maxima permite achar a reunião, intersecção e diferença de conjuntos por intermédio, respectivamente, dos operadores `union`, `intersect` ou `intersection` e `setdifference`:

```
(C16) A : {1,{1,2}}$
(C17) B : {1,{1,3}}$
(C18) union(A,B);
(D18)                   {1, {1, 2}, {1, 3}}
(C19) intersect(A,B);
(D19)                          {1}
(C20) intersection(A,B);
(D20)                          {1}
(C21) setdifference(A,B);
(D21)                       {{1, 2}}
(C22) setdifference(B,A);
(D22)                       {{1, 3}}
```

O Maxima permite também determinar um subconjunto de um dado conjunto, satisfazendo um determinado predicado. Por exemplo, se se pretender determinar o subconjunto dos números pares de um dado conjunto, faz-se:

```
(C23) subset({1,2,7,8,9,14},evenp);
(D23)                  {2, 8, 14}
```

Para vermos se um conjunto é subconjunto de outro conjunto, usamos a função `subsetp`

```
(C23) subsetp({2, 8, 14},{1,2,7,8,9,14});
(D23)                      TRUE
```

Os conjuntos são muitas vezes definidos por operações sobre outros conjuntos. Por exemplo, sejam a e b reais tais que $a < b$. O intervalo $[a, b[$ é a intersecção dos dois conjuntos A e B definidos por

$$A = \{x \in \mathbb{R} : x \geq a\} \ , \quad B = \{x \in \mathbb{R} : x < b\} \ .$$

Da definição de intersecção de dois conjuntos, resulta que $x \in [a, b[\Leftrightarrow x \in A \cap B$. A condição $x \in [a, b[$ é testada em Maxima através do valor (lógico) da expressão lógica

```
x >= a and x < b;
```

A reunião de dois conjuntos requer uma construção idêntica com o `and` substituído por `or`; enquanto a diferença entre A e B é lida como $x \in A$ `and` (`not` $x \in B$).

Os operadores lógicos `and` e `or`, e o operador lógico unário `not`, relacionam expressões cujo valor é do tipo Booleano. As seguintes propriedades são satisfeitas:

- `x and y` $\Leftrightarrow$ `y and x`
- `x or y` $\Leftrightarrow$ `y or x`
- `not (x and y)` $\Leftrightarrow$ `not(x) or not(y)`
- `not (x or y)` $\Leftrightarrow$ `not(x) and not(y)`

Ao existirem vários operadores lógicos numa mesma expressão, eles são considerados pela seguinte ordem: primeiro o `not`, depois o `and`, a seguir o `or`. Os parêntesis são usados para alterar as prioridades.

As expressões contendo operadores lógicos são identificadas pelo Maxima como sendo do tipo lógico. Neste caso elas são calculadas automaticamente, não sendo necessário o uso do `ev`.

```
(C24) not(true and false) = (not true) or (not false);
(D24)                      TRUE
```

Proposição 1. *Se* $A = \{1, 2, 3\}$ *e* $B = \{2, 3, 4\}$ *então* $4 \in B \setminus A$.

Demonstração.

```
(C25) A : {1,2,3}$
(C26) B : {2,3,4}$
(C27) elementp(4,setdifference(B,A)); /* 1a demonstração */
(D27)                      TRUE
(C28) elementp(4,B) and not elementp(4,A); /* 2a demonstração */
(D28)                      TRUE
```

□

Proposição 2. $83|(3^{41}-1)$ *e* $83 \nmid (2^{41}-1)$.

Demonstração.

```
(C29) remainder(3^41-1,83)=0 and not remainder(2^41-1,83)=0;
(D29)                        TRUE
```

□

Reparar que na demonstração da Proposição 2 o comando Maxima não necessita do `ev`: embora o operador relacional `=` esteja envolvido, a presença dos operadores lógicos tornam o tipo Booleano da expressão inequívoco e o Maxima calcula o valor lógico automaticamente.

Proposição 3. *O número* $n = 1004^2 + 1$ *é um número primo da forma* $7k+3$ *ou* $7k+4$ *para algum inteiro* k.

Demonstração. Temos que testar que n é primo e verificar que ao dividirmos n por 7 obtemos resto 3 ou 4.

```
(C30) 1004^2 + 1$
(C31) primep(%) and (remainder(%,7)=3 or remainder(%,7)=4);
(D31)                        TRUE
```

O último comando Maxima envolve duas vezes o mesmo cálculo. Podemos evitar o cálculo repetido. Uma demonstração alternativa seria:

```
(C32) 1004^2 + 1$
(C33) primep(%) and elementp(remainder(%,7),{3,4});
(D33)                        TRUE
```

□

3.2 Funções definidas pelo utilizador

É possível definir em Maxima novas funções, a ser usadas em pé de igualdade com as funções pré-definidas. A construção de funções em Maxima usa o operador `:=`. Vejamos um exemplo. A função

$$\begin{aligned} f : \mathbb{Z} \setminus \{0\} &\to \mathbb{Q} \\ x &\mapsto \frac{x+1}{x} \end{aligned}$$

é construída em Maxima como se segue:

```
(C1) f(x) := (x+1)/x;
(D1)
```

$$f(x) := \frac{x+1}{x}$$

Outra forma de definir funções é através do comando `define`:

```
(C2) define(f(x),(x+1)/x);
```

Para sabermos que funções foram definidas pelo utilizador, fazemos

```
(C3) functions;
(D3)                          [f(x)]
```

A partir do momento que a função é definida, passa a poder ser usada como qualquer uma das funções pré-definidas, substituindo qualquer expressão válida no seu argumento. No nosso exemplo, são válidas expressões aritméticas e algébricas:

```
(C4) f(2/3);
(D4)
```

$$\frac{5}{2}$$

```
(C5) f(a);
(D5)
```

$$\frac{a+1}{a}$$

```
(C6) f(ola);
(D6)
```

$$\frac{ola+1}{ola}$$

```
(C7) f(b^2-1);
(D7)
```

$$\frac{b^2}{b^2-1}$$

Resultados análogos podem ser obtidos por via de substituições, embora isso seja, a maior parte das vezes, muito menos "elegante".

```
(C8) f : (x+1)/x$
(C9) subst(x=2/3,f);
(D9)
```

$$\frac{5}{2}$$

```
(C10) subst(x=a,f);
(D10)
```

$$\frac{a+1}{a}$$

```
(C11) subst(x=ola,f);
(D11)
```

$$\frac{ola+1}{ola}$$

```
(C11) subst(x=b^2-1,f);
(D11)
```

$$\frac{b^2}{b^2-1}$$

O nome do argumento da função (a variável que aparece entre parêntesis) é uma variável muda e, em particular, não está relacionada com nenhuma variável do mesmo nome que possa ter sido previamente definida (em linguagem de programação, diz-se que é uma *variável local*).

```
(C12) g(n) := 2*n$
(C13) h(oQueSeja) := 2*oQueSeja$
(C14) oQueSeja : 3$
(C15) g(2);
(D15)                          4
(C16) h(2);
(D16)                          4
(C17) g(oQueSeja);
(D17)                          6
(C18) h(oQueSeja);
(D18)                          6
(C19) g(n);
(D19)                        2 n
(C20) h(n);
(D20)                        2 n
```

Erros frequentes na definição de funções:

```
(C21) y:=x^2;
Improper function definition:
y
 -- an error.  Quitting.  To debug this try DEBUGMODE(TRUE);
```

```
(C22) y=x^2;
(D22)
```

$$y = x^2$$

```
(C23) y(2);
(D23)                        y(2)
```

```
(C24) y(x)=x^2;
(D24)
```

$$y(x) = x^2$$

```
(C25) y(2);
(D25)                        y(2)
```

```
(C26) y[x]=x^2;
(D26)
```

$$y_x = x^2$$

```
(C27) y[2];
(D27)
```

$$y_2$$

3.3 A função característica e a estrutura `if`

Seja A um conjunto e C um seu subconjunto. A função característica de C em A, denotada por χ_C, é definida como se segue:

$$\begin{aligned} \chi_C &: A \to \{0,1\} \\ x &\mapsto \begin{cases} 1 & \text{se } x \in C \\ 0 & \text{se } x \notin C \end{cases} \end{aligned} \tag{3.1}$$

Uma variante é a função característica Booleana, dada por

$$\begin{aligned} \chi_C &: A \to \{true, false\} \\ x &\mapsto \begin{cases} true & \text{se } x \in C \\ false & \text{se } x \notin C \end{cases} \end{aligned} \tag{3.2}$$

O cálculo do valor da função característica (3.1) ou (3.2) para um determinado $x \in A$ envolve um processo de decisão: é preciso decidir se x pertence, ou não, a C e afectar o valor à função χ de acordo. O caso Booleano é mais simples, uma vez que os valores `true` ou `false` podem ser obtidos por intermédio de uma expressão Booleana. Se o conjunto C for dado explicitamente, então $\chi_C(x)$ é representado em Maxima pela expressão `elementp(x,C)`. Se C é definido por uma certa propriedade, então traduzimo-la numa expressão lógica em Maxima e adoptamos uma construção do tipo

```
chi(x) := expressaoLogica$
```

que pode, ou não, requerer o uso do `ev`.

Exemplo 15. *Seja $n \in \mathbb{Z}$. Vamos construir a função* ***`par(n)`*** *cujo valor é* ***`true`*** *se n é par e* ***`false`*** *se n é ímpar.*

```
(C2) par(n) := ev(remainder(n,2)=0,PRED)$
(C3) par(1000);
(D3)                        TRUE
(C4) par(555);
(D4)                       FALSE
```

Em Maxima, existe a função `evenp` para avaliar se uma expressão é par, e a função `oddp` para avaliar se uma expressão é ímpar. Para o exemplo anterior ficaria

```
(C5) evenp(1000);
(D5)                        TRUE
(C6) evenp(555);
(D6)                       FALSE
(C7) oddp(555);
(D7)                        TRUE
```

Exemplo 16. *Consideremos o intervalo $[0,1[$. A sua função característica é dada por*

```
(C8) intervalo(x):= x >= 0 and x < 1$
(C9) intervalo(1/2);
(D9)                     TRUE
(C10) intervalo(4/3);
(D10)                    FALSE
```

Notar que o `ev` *não é necessário aqui, uma vez que está presente o operador lógico* `and`*.*

Para implementarmos em Maxima a função característica não-Booleana (3.1), recorremos à estrutura `if`. A sua forma mais simples é:

```
IF condicao THEN expressao1 ELSE expressao2;
```

Se `condicao` resulta `true`, então a `expressao1` é executada; se ela resulta `false`, então a `expressao2` é executada. O valor da `condicao` é determinado automaticamente, i.e., não há necessidade de se usar o `ev`. A estrutura `if` corresponde a um único comando Maxima e por isso as expressões não necessitam ser terminadas por `$` ou `;`

Exemplo 17. *Implemente em Maxima a função característica*

$$\chi_{\{0\}} : x \mapsto \begin{cases} 1 & se\ x = 0 \\ 0 & se\ x \neq 0 \end{cases}$$

```
(C11) chi(x):= if x = 0 then 1 else 0 $
(C12) chi(0);
(D12)                        1
(C13) chi(1);
(D13)                        0
```

A versão Booleana da mesma função não requer o uso da estrutura `if`*:*

```
(C14) chiBooleana(x):= ev(x = 0,pred)$
(C15) chiBooleana(0);
(D15)                      TRUE
(C16) chiBooleana(1);
(D16)                     FALSE
```

Exemplo 18. *A função característica Booleana do conjunto dos números primos é dada por* `primep`*. A versão não-Booleana é construída com a ajuda do* `if`*:*

```
(C17) chiPrimos(n):= if primep(n) then 1 else 0 $
(C18) chiPrimos(4);
(D18)                        0
(C19) chiPrimos(7);
(D19)                        1
```

Exemplo 19. *Vamos definir em Maxima a função característica de* $\mathbb{Z}$ *em* $\mathbb{Q}$*:*

$$\chi : \mathbb{Q} \to \{0, 1\}$$
$$x \mapsto \begin{cases} 1 & se\ x \in \mathbb{Z} \\ 0 & se\ x \notin \mathbb{Z}\,. \end{cases}$$

Para isso notamos que um inteiro é um racional com denominador igual a 1.

```
(C20) chi(x):= if denom(x) = 1 then 1 else 0 $
(C21) chi(8/4);
(D21)                          1
(C22) chi(8/3);
(D22)                          0
(C23) chi(10000);
(D23)                          1
```

Em Maxima, existe a função **integerp**, *que devolve* **true** *se é um número inteiro, e* **false** *caso contrário. Podemos então fazer*

```
(C24) chi(x):= if integerp(x) then 1 else 0 $
```

A versão Booleana será apenas

```
(C25) integerp(8/4);
(D25)                         TRUE
(C26) integerp(8/3);
(D26)                        FALSE
(C27) integerp(10000);
(D27)                         TRUE
```

Exemplo 20. *Pretende-se agora construir a função* **nint(x)** *que retorna o inteiro mais próximo do racional positivo* $x = \frac{a}{b}$. *Usamos o seguinte raciocínio: se a parte fraccionária de* x *não excede* $\frac{1}{2}$, *então o inteiro mais próximo é dado pelo quociente da divisão de a por b; senão, é a mesma quantidade mais um. Para melhorar a legibilidade da definição da função* **nint**, *subdividimos o comando Maxima por várias linhas de entrada.*

```
(C28) frac(x):= x - entier(x)$
(C29) nint(x):= if frac(x) <= 1/2 then
                  quotient(num(x),denom(x))
                else
                  quotient(num(x),denom(x)) + 1
                $
(C30) nint(1/3);
(D30)                          0
(C31) nint(2/3);
(D31)                          1
```

3.4 Funções de várias variáveis

O operador := pode ser usado para definir funções de várias variáveis:

```
(C1) f(x,y):=  x * y$
(C2) f(3,z);
(D2)                         3 z
(C3) f(3,5);
(D3)                         15
```

Exemplo 21. *Queremos construir a função característica do conjunto $d\mathbb{Z}$ formado pelos inteiros múltiplos de um dado inteiro não nulo d. Usando o facto que* `irem(x,d)` *é zero precisamente quando x é múltiplo de d, obtemos*

```
(C4) multd(x,d):=  if remainder(x,d) = 0 then 1 else 0 $
(C5) multd(28,7);
(D5)                          1
(C6) multd(29,7);
(D6)                          0
```

Exemplo 22. *Pretendemos agora definir a função característica do conjunto dos divisores (positivos ou negativos) de um dado inteiro não nulo n. Uma possível definição é:*

```
(C7) divs(x,n):=  if remainder(n,x) = 0 then 1 else 0 $
(C8) divs(7,28);
(D8)                          1
(C9) divs(7,29);
(D9)                          0
```

Outra alternativa poderá ser

```
(C10) divs(x,n):=  multd(n,x)$
```

3.5 Mapeamento dos elementos de um conjunto

A imagem $f(A)$ de um conjunto A sob uma função f pode ser construída em Maxima por intermédio de um comando `map`.

No seguinte exemplo construímos a imagem do conjunto $\{-2,-1,0,1,2\}$ sob a função $f : x \mapsto x^2$.

```
(C1) load(nset)$
(C2) A: set(-2,-1,0,1,2)$
(C3) f(x):= x^2$
(C4) map(f,A);
(D4)                       {0, 1, 4}
```

A função `map` tem a seguinte sintaxe:

```
MAP(fun(arg1,..., argn), exp1,..., expn)
```

onde `fun` ou é o nome de uma função de n argumentos ou é uma forma `LAMBDA` (que serve para criar funções anónimas) de n argumentos. Na situação anterior poderíamos ter feito directamente:

```
(C5) map(lambda([x],x^2),set(-2,-1,0,1,2));
(D5)                       {0, 1, 4}
```

Outros exemplos de aplicação da função `map` são:

```
(C6) map(f,x+a*y+b*z);
(D6)
```

$$x^2 + a^2y^2 + b^2z^2$$

```
(C7) map(ratsimp, x/(x^2+x)+(y^2+y)/y);
(D7)
```

$$y + \frac{1}{x+1} + 1$$

```
(C8) map("=",[A,B],[-0.5,3]);
(D8)                [A = - 0.5, B = 3]
```

Proposição 4. *Seja $A = \{0, 1, 2, 3, 4, 5\}$. A função*

$$f : A \to A$$
$$x \mapsto resto\left(\frac{x+5}{6}\right)$$

é uma função sobrejectiva.

Demonstração. Temos de mostrar que $f(A) = A$.

```
(C9) set(0,1,2,3,4,5)$
(C10) ev(map(lambda([x],remainder(x+5,6)),%)=%,pred);
(D10)                     TRUE
```

□

Exemplo 23. *Dado um conjunto de números inteiros A pretende-se construir uma função que devolve* `true` *se, e somente se, A contém um primo.*

```
(C11) contemPrimo(A) :=  elementp(true,map(primep,A))$
(C12) contemPrimo(set(4,6,10));
(D12)                    FALSE
(C13) contemPrimo(set(3,5,7));
(D13)                     TRUE
```

Um comando da mesma família do `map` é o comando `apply`. Vejamos um exemplo.

Exemplo 24. *Seja A um conjunto de inteiros e d um inteiro não nulo. Vamos definir uma função que constrói o conjunto dos racionais obtido dividindo cada elemento de A por d.*

```
(C14) divPor(A,d) := setify(apply(lambda([x,y],x/y),[A,d]))$
(C15) divPor([2,4,8],2);
(D15)                    {1, 2, 4}
```

ou então:

```
(C14) divPor(A,d) := setify(apply(lambda([x,y],x/y),[listify(A),d]))$
(C15) divPor(set(2,4,8),2);
(D15)                    {1, 2, 4}
```

Capítulo 4

Sucessões em Maxima

4.1 Sucessões

Quando o domínio de uma função é o conjunto $\mathbb{N}$ dos números naturais dizemos que estamos perante uma *sucessão*. Deste modo uma sucessão f é uma função que associa a cada inteiro positivo n um elemento único $f(n)$ de um dado conjunto. Quando uma função é uma sucessão, é usual em matemática escrever-se f_n em vez de $f(n)$. Na notação usual, f é uma função; n é um elemento do domínio; $\{f_n\}$ é a imagem.

A restrição do domínio de uma sucessão a $\mathbb{N}$ pode ser relaxada. Por exemplo, pode ser conveniente restringir o domínio a um subconjunto de $\mathbb{N}$; considerar o domínio como sendo o conjunto $\mathbb{Z}$ dos inteiros, etc.

A maneira mais fácil de gerar os elementos da sucessão,

$$f_1\,, f_2\,, f_3\,, \ldots\,, f_k\,, \ldots$$

é através da chamada à função Maxima `makelist`. Outra forma será através de um ciclo `for`.

Vejamos um exemplo. Suponhamos que queremos gerar os primeiros 20 números primos, através da função `ithprime`. Para isso fazemos

```
(C1) makelist(ithprime(i),i,1,20);
(D1) [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71]
```

Podemos também fazer:

```
(C2) block(s:set(),for i from 1 thru 20 do s:adjoin(ithprime(i), s),listify(s));
(D2) [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71]
```

ou ainda:

```
(C3)for i from 1 thru 20 do print(ithprime(i));
2
3
5
7
11
13
17
```

```
19
23
29
31
37
41
43
47
53
59
61
67
71
(D3)                    DONE
```

O comando Maxima anterior é equivalente à sucessão de expressões

```
ithprime(1), ithprime(2), ..., ithprime(20)
```

Exemplo 25. *Consideremos a sucessão definida pela função característica Booleana do conjunto dos números primos em* $\mathbb{N}$*:*

$$p : \mathbb{N} \to \{true, false\}$$
$$n \mapsto \begin{cases} true & \text{se } n \text{ é primo} \\ false & \text{se } n \text{ não é primo} \end{cases}$$

Pretendemos gerar os elementos $p_{100}, \ldots, p_{120}$ *desta sucessão. O problema é facilmente resolvido em Maxima:*

```
(C4) makelist(primep(n),n,100,120);
(D4) [FALSE, TRUE, FALSE, TRUE, FALSE, FALSE, FALSE, TRUE, FALSE, TRUE,

   FALSE, FALSE, FALSE, TRUE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE]
```

Exemplo 26. *Pretendemos gerar os primeiros 15 elementos da sucessão*

$$f : n \mapsto \frac{n+1}{n^2}\,, \quad n \geq 1\,.$$

O problema é resolvido definindo a função f *em Maxima e recorrendo depois a um comando* `makelist`*:*

```
(C5) f(n) := (n+1)/(n^2)$
(C6) makelist(f(n),n,1,15);
(D6)
```

$$\left[2, \frac{3}{4}, \frac{4}{9}, \frac{5}{16}, \frac{6}{25}, \frac{7}{36}, \frac{8}{49}, \frac{9}{64}, \frac{10}{81}, \frac{11}{100}, \frac{12}{121}, \frac{13}{144}, \frac{14}{169}, \frac{15}{196}, \frac{16}{225}\right]$$

Exemplo 27. *Pretende-se definir em Maxima o conjunto* C *formado pelas primeiras 30 potências não negativas de 2.*

```
(C7) C: setify(makelist(2^j,j,0,29));
(D7) {1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384,
      32768, 65536, 131072, 262144, 524288, 1048576, 2097152, 4194304, 8388608,
      16777216, 33554432, 67108864, 134217728, 268435456, 536870912}
```

O comando `makelist` devolve uma lista e pode ter uma das seguintes sintaxes:

`makelist(exp, var, lo, hi)`, onde `lo` e `hi` são inteiros;

`makelist(exp, var, list)`.

No primeiro caso, `makelist` é análogo a `SUM`, enquanto que o segundo caso é similar a `MAP`. Por exemplo:

```
(C8) makelist(concat(X,I),I,1,6);
(D8) [X1,X2,X3,X4,X5,X6]
(C9) makelist(X=Y,Y,[A,B,C]);
(D9) [X=A,X=B,X=C]
```

O seguinte comando Maxima gera uma sucessão de conjuntos:

```
(C10) makelist(setify(makelist(i,i,1,n)),n,1,6);
(D10) [{1}, {1, 2}, {1, 2, 3}, {1, 2, 3, 4}, {1, 2, 3, 4, 5}, {1, 2, 3, 4, 5, 6}]
```

Podemos também fazer da seguinte maneira:

```
(C11) block(s:set(),print(s),for i from 1 thru 6 do print(s:adjoin(i, s)));
{}
{1}
{1, 2}
{1, 2, 3}
{1, 2, 3, 4}
{1, 2, 3, 4, 5}
{1, 2, 3, 4, 5, 6}
(D11)                          DONE
```

Exemplo 28. *Pretendemos construir a função* `divisores(n)` *cujo valor é o conjunto dos divisores de um dado inteiro positivo n. (O Maxima já disponibiliza esta função no* package `nset`*. Veja no ficheiro* nset.html *o comando* `divisors`*. Vamos, mesmo assim, defini-la. É um bom treino!) Uma maneira fácil (embora extremamente ineficiente...) de resolver o problema consiste em aplicar a função* `x -> gcd(x,n)` *ao conjunto dos primeiros n inteiros positivos.*

```
(C12) divisores(n) := setify(makelist(gcd(i,n),i,1,n))$
(C13) divisores(99);
(D13)                  {1, 3, 9, 11, 33, 99}
```

Vamos agora proceder a alguns melhoramentos na eficiência computacional da função `divisores` *acima definida. Em primeiro lugar notamos que estamos a chamar a função* `gcd` *n vezes. Além dos divisores triviais 1 e n, todos os outros divisores de n estão entre 2 e* $n/2$*. Podemos assim poupar metade do tempo gasto nos vários cálculos do* `gcd`*.*

```
(C14) divisores(n):= union(setify(makelist(gcd(i,n),i,1,quotient(n,2))),set(1,n))$
```

4.2 Gráficos dos elementos de uma sucessão

O Maxima disponibiliza várias funções para fazer gráficos, nomeadamente, `plot2d`, `plot3d` e `openplot_curves`. Antes de fazer um gráfico em Maxima, deve-se ir ao menu `Options` > `Plot Windows` e seleccionar a opção `Separate`. Isto faz com que o gráfico apareça numa janela separada. O `openplot_curves` permite representar um conjunto discreto de pontos do plano

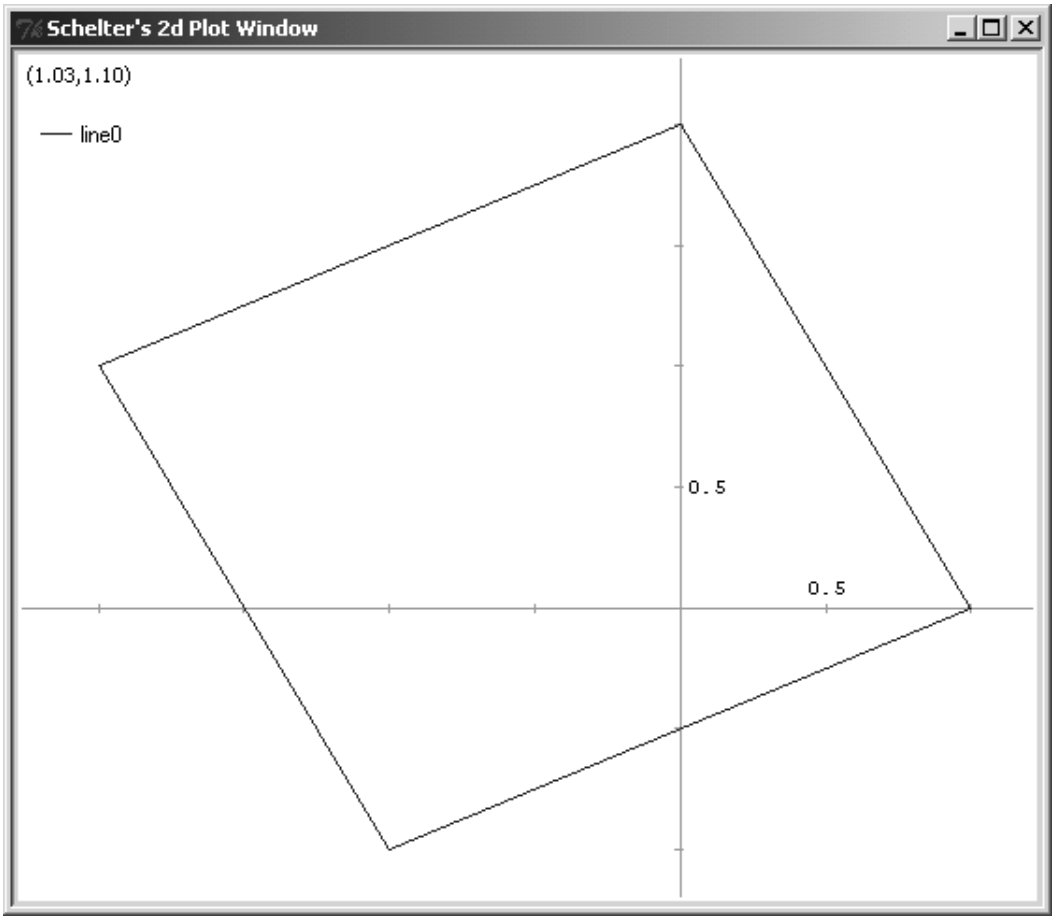
$$(x_1, y_1)\ , (x_2, y_2)\ , \ldots , (x_n, y_n)\ . \tag{4.1}$$

Um ponto (x, y) do plano Cartesiano é representado como uma *lista* de dois elementos. Como sabemos, a lista é um tipo de dados disponibilizado pelo Maxima, que consiste numa sucessão de objectos entre parêntesis rectos. O ponto (x, y) é então representado em Maxima na forma `[x,y]`. Os pontos (4.1) que pretendemos esboçar por meio do `openplot_curves`, devem também ser organizados numa lista:

$$[[x_1, y_1]\ , [x_2, y_2]\ , \ldots , [x_n, y_n]]\ .$$

Os seguintes comandos geram um quadrado com vértices nos pontos $(-1, -1)$, $(1, 0)$, $(0, 2)$, $(-2, 1)$.

```
(C1) v : [[-1,-1],[1,0],[0,2],[-2,1],[-1,-1]]$
(D1) openplot_curves([v]);
```



```
(C2) listp(v);
(D2)                          TRUE
```

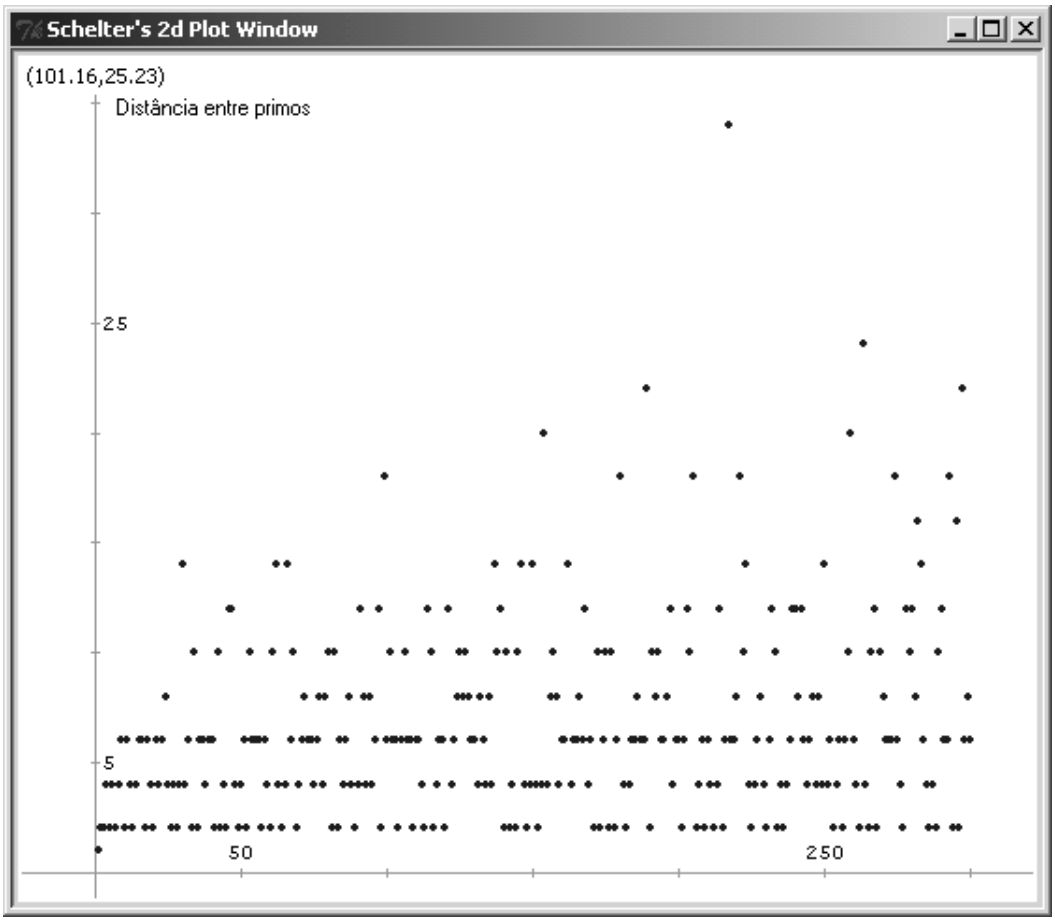
O Maxima liga automaticamente os pontos com um segmento. De modo a obtermos um quadrado introduzimos o vértice $(-1,-1)$ duas vezes.

Para fazermos um gráfico com os valores $u_1\,, u_2\,, \ldots\,, u_n$ de uma sucessão, consideramos os pontos de coordenadas Cartesianas

$$(1, u_1)\ , (2, u_2)\ , \ldots\ , (n, u_n)\ .$$

Exemplo 29. *Vamos ilustrar graficamente a sucessão $n \mapsto u_n = p_{n+1} - p_n$ (1, 2, ...), onde p_n representa o n-ésimo primo, visualizando os primeiros 300 elementos da sucessão.*

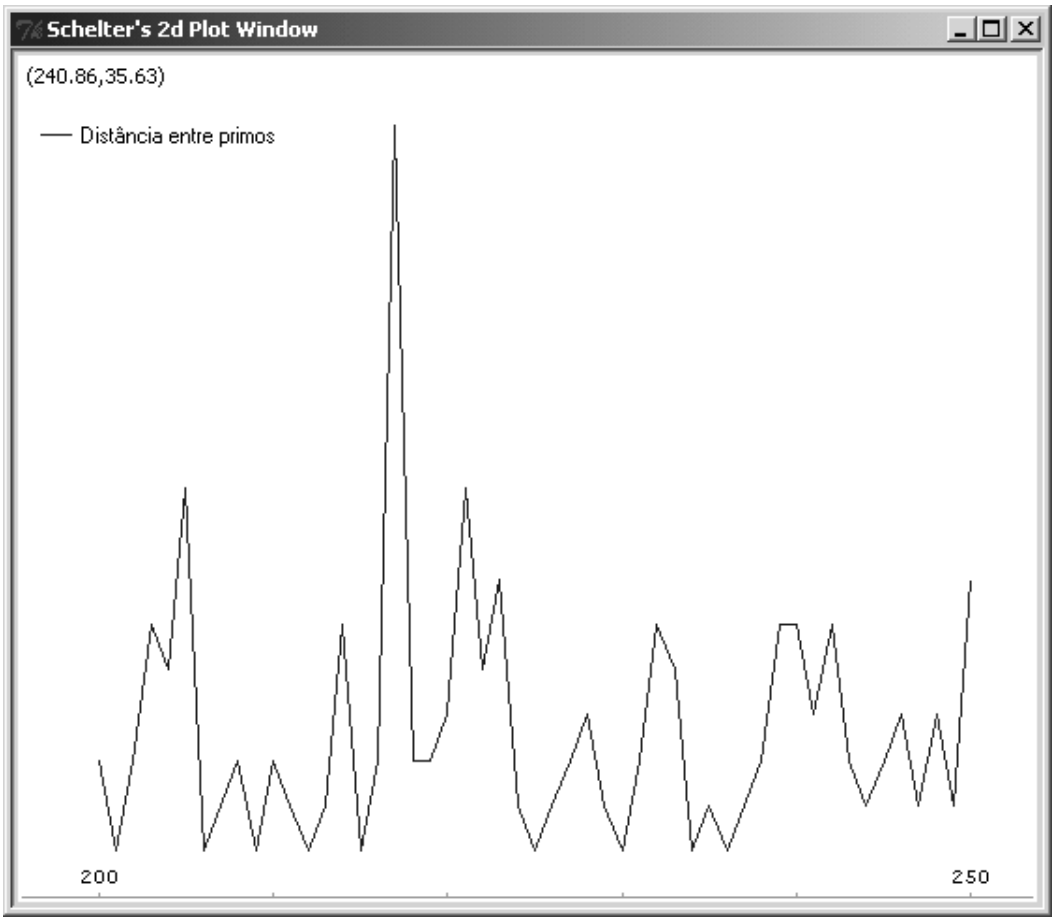
```
(C3) load(prime)$
(C4) u(n) :=  ithprime(n+1)-ithprime(n)$
(C5) dados: makelist([n,u(n)],n,1,300)$
(C6) openplot_curves([["{label } {nolines 1} {plotpoints 1}
     {yaxislabel {Distância entre primos}}"],dados]);
```



A opção `{nolines 1}` mostra pontos desconectados. No exemplo acima usámos também a opção `yaxislabel` que nos permite associar um título ao eixo dos y. Uma análise da figura obtida permite-nos concluir que com o aumento do n aparecem maiores distâncias

entre os primos, embora de uma maneira muito irregular. A maior distância entre os dados considerados está localizada entre os primos p_{200} e o p_{250}. Para a localizarmos com maior precisão, vamos fazer um "*zoom* do gráfico", não usando agora a opção `noline` para uma melhor visualização.

```
> dados2: makelist([n,u(n)],n,200,250)$
> openplot_curves([["{label {Distância entre primos}}"],dados2]);
```



4.3 Sucessões Periódicas e Definições Recursivas

Uma sucessão $\{a_n\}_{n=1}^{\infty}$ diz-se *periódica* se consistir numa repetição infinita do mesmo padrão finito:

$$a_1\,,\ldots,a_k\,,\underbrace{a_{k+1}\,,a_{k+2}\,,\ldots,a_{k+T}}\,,\underbrace{a_{k+1}\,,a_{k+2}\,,\ldots,a_{k+T}}\,,\underbrace{a_{k+1}\,,a_{k+2}\,,\ldots,a_{k+T}}\,,\ldots$$

Vamos chamar *transição* ao número k e *período* ao número T. A periodicidade é expressa concisamente pela *notação recursiva*

$$a_{n+T}=a_n\,,\quad n\geq k+1\,. \tag{4.2}$$

A sucessão $\{a_n\}$ fica completamente definida por (4.2) especificando os valores de a_i para $i=1,\ldots,k+T$.

Exemplo 30. *Seja*

$$a : \mathbb{N} \to \{-1, 1\}$$
$$n \mapsto (-1)^n$$

A sucessão é periódica com transição nula ($k = 0$) e período dois ($T = 2$):

$$\underbrace{-1, 1}, \underbrace{-1, 1}, \underbrace{-1, 1}, \ldots$$

A sucessão é definida recursivamente como se segue:

$$a_1 = -1,\ a_2 = 1 \quad \textit{(condições iniciais)}$$
$$a_{n+2} = a_n \quad n \geq 1\,. \quad \textit{(fórmula recursiva)}$$

Em Maxima*, uma maneira definir esta sucessão é a seguinte:*

```
(C1) a[1]: -1$
(C2) a[2]: 1$
(C3) a[n]:= a[n-2]$
(C4) makelist(a[n],n,1,6);
(D4)                [- 1, 1, - 1, 1, - 1, 1]
```

Neste exemplo, definiu-se a sucessão recursivamente. Podemos defini-la de outras maneiras, por exemplo, recorrendo à estrutura `if`*:*

```
(C5) a(n):= if n > 0 then
               block(if oddp(n) then - 1 else 1)
            else "o argumento da função é um número natural (n = 1, 2,...)"
            $
(C6) a(1);
(D6)                        - 1
(C7) a(2);
(D7)                         1
(C8) a(0);
(D8)        o argumento da função é um número natural (n = 1, 2,...)
(C9) makelist(a(n),n,1,6);
(D9)                [- 1, 1, - 1, 1, - 1, 1]
```

Exemplo 31. *Consideremos a sucessão definida pela função característica do subconjunto* $\{0\}$ *em* $\mathbb{N}_0$*:*

```
(C10) chi(n):= if n = 0 then 1 else 0$
```

A sucessão é periódica com transição 1 e período 1.

```
(C11) makelist(chi(i), i,0,19);
(D11)      [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

Problema 3. *Para cada natural n seja $\mathcal{F}_n$ o conjunto dos números racionais no intervalo $[0,1]$ cujo denominador não excede n. Este conjunto, conhecido como* conjunto de Farey*, pode ser definido como se segue:*

$$\mathcal{F}_n = \left\{ \frac{p}{q} \in \mathbb{Q} : 1 \leq q \leq n,\, 0 \leq p \leq q \right\}.$$

Nesta definição algumas fracções aparecem várias vezes, mas as múltiplas ocorrências são eliminadas de acordo com a definição de conjunto. A função $n \mapsto \mathcal{F}_n$ define uma sucessão de conjuntos finitos cada um deles contido no seguinte:

$$\mathcal{F}_1 \subset \mathcal{F}_2 \subset \mathcal{F}_3 \subset \ldots$$

Os primeiros 3 termos da sucessão são:

$$\mathcal{F}_1 = \left\{ \frac{0}{1}, \frac{1}{1} \right\} \tag{4.3}$$

$$\mathcal{F}_2 = \left\{ \frac{0}{1}, \frac{1}{2}, \frac{1}{1} \right\} \tag{4.4}$$

$$\mathcal{F}_3 = \left\{ \frac{0}{1}, \frac{1}{3}, \frac{1}{2}, \frac{2}{3}, \frac{1}{1} \right\} \tag{4.5}$$

$$\tag{4.6}$$

Pretendemos construir em Maxima uma função `F(n)` cujo valor é o conjunto de Farey $\mathcal{F}_n$. Como sempre, é boa estratégia construir a função passo a passo, o que nos permite ir testando as funcionalidades pretendidas com alguns exemplos e detectar possíveis erros de estratégia. No final podemos juntar todos os passos numa única definição. Para começar construímos uma sucessão contendo todas as fracções com um dado denominador q. Vamos excluir os elementos 0 e 1 para diminuir o número de fracções repetidas na definição da função `F(n)`.

```
(C12) q: 10$
(C13) for p: 1 thru q-1 do
          print(p/q);
```

$$\frac{1}{10}$$
$$\frac{1}{5}$$
$$\frac{3}{10}$$
$$\frac{2}{5}$$
$$\frac{1}{2}$$
$$\frac{3}{5}$$
$$\frac{7}{10}$$

$$\frac{4}{5}$$

$$\frac{9}{10}$$

```
(D13)                          DONE
```

De seguida vamos incorporar o comando Maxima acima num outro ciclo `for` *que faz variar o valor q (q = 2, . . . , n). Reparar que o* `q` *abaixo é uma variável muda (variável local), não dependendo da afectação à variável* `q` *que fizemos anteriormente!*

```
(C14) n: 3$
(C15) block(s: set(),
            for q: 2 thru n do
                block(for p: 1 thru q-1 do
                          s: adjoin(p/q,s)),
            s);
```

(D15)
$$\left\{\frac{1}{3}, \frac{1}{2}, \frac{2}{3}\right\}$$

Por fim adicionamos os valores 0 e 1 em falta:

```
(C16) union(set(0,1), block(s: set(),
                            for q: 2 thru n do
                                block(for p: 1 thru q-1 do
                                          s: adjoin(p/q,s)),
                            s));
```

(D16)
$$\left\{0, \frac{1}{3}, \frac{1}{2}, \frac{2}{3}, 1\right\}$$

O conjunto visualizado não é mais do que o $\mathcal{F}_3$*. Estamos em condições de definir em Maxima a função* $\mathcal{F}(n)$*.*

```
(C17) F(n):= union(set(0,1), block(s: set(),
                                   for q: 2 thru n do
                                       block(for p: 1 thru q-1 do
                                                 s: adjoin(p/q,s)),
                                   s))$
(C18) F(5);
```

(D18)
$$\left\{0, \frac{1}{5}, \frac{1}{4}, \frac{1}{3}, \frac{2}{5}, \frac{1}{2}, \frac{3}{5}, \frac{2}{3}, \frac{3}{4}, \frac{4}{5}, 1\right\}$$

4.4 Definições Recursivas em Maxima

Até agora temos dado ênfase a sucessões definidas explicitamente por intermédio de uma função:

$$x_n = \text{uma função explícita de } n, \quad n \geq 1\,.$$

Esta construção nem sempre é a mais intuitiva. Um caso muito interessante, e importante, é o das sucessões definidas *recursivamente*. Na sua forma mais simples, uma sucessão definida recursivamente tem a forma

$$x_n = \text{uma função explícita de } x_{n-1}, \quad n \geq 1\,,$$

isto é, cada elemento da sucessão é definido de maneira explícita à custa do elemento anterior. Uma sucessão deste tipo é apelidada de *sucessão recursiva de primeira ordem*. As sucessões recursivas de primeira ordem são definidas a partir do momento que especificamos o elemento x_0, chamado de *condição inicial*. Concretamente, seja $x_0 = \beta$ e $x_n = f(x_{n-1})$, $n \geq 1$. Os primeiros termos da sucessão são:

$$\begin{aligned} x_0 &= \beta \\ x_1 &= f(x_0) = f(\beta) \\ x_2 &= f(x_1) = f\left(f(\beta)\right) \\ x_3 &= f(x_2) = f\left(f\left(f(\beta)\right)\right) \\ &\vdots \end{aligned}$$

Vemos que numa sucessão recursiva a regra que associa a n o elemento x_n é: "aplica f a x_0 n vezes".

Exemplo 32. *Seja*

$$\begin{aligned} f : \mathbb{Z} &\to \mathbb{Z} \\ n &\mapsto -2n + 1 \end{aligned}$$

Com condição inicial $x_0 = -1$, a sucessão recursiva correspondente é dada por:

$$x_0 = -1\,, x_1 = f(-1) = 3\,, x_2 = f(3) = -5\,, x_3 = f(-5) = 11\,, \ldots$$

Em Maxima podemos fazer:

```
(C1) f(n):= -2*n + 1$
(C2) x: -1$
(D2)                          - 1
(C3) x: f(x);
(D3)                           3
(C4) x: f(x);
(D4)                          - 5
(C5) x: f(x);
(D5)                          11
```

Outra maneira é usar variáveis ditto*:*

```
(C6) -1;
(D6)                          - 1
(C7) f(%);
(D7)                           3
(C8) f(%);
(D8)                          - 5
(C9) f(%);
(D9)                          11
```

Mudando a condição inicial obtemos, obviamente, valores diferentes:

```
(C10) %pi;
(D10)                        %PI
(C11) f(%);
(D11)                     1 - 2 %PI
(C12) f(%);
(D12)                 1 - 2 (1 - 2 %PI)
(C13) f(%);
(D13)             1 - 2 (1 - 2 (1 - 2 %PI))
```

Em Maxima*, também é possível definir uma sucessão recursiva de maneira análoga à usada em Matemática:*

```
(C14) f[0]: -1$
(C15) f[n]:= -2*f[n-1] + 1$
(C16) f[0];
(D16)                        - 1
(C17) f[1];
(D17)                         3
(C18) f[2];
(D18)                        - 5
(C19) f[3];
(D19)                        11
```

É preciso ter em atenção que, se se pretender alterar a definição da sucessão, é necessário usar o comando `kill` *antes da sua alteração (isto faz com que o* Maxima *"esqueça"os resultados em* cache*):*

```
(C20) kill(f)$
(C21) f[0]: %pi$
(C22) f[n]:= -2*f[n-1] + 1$
(C23) f[0];
(D23)                        %PI
(C24) f[1];
(D24)                     1 - 2 %PI
(C25) f[2];
(D25)                 1 - 2 (1 - 2 %PI)
(C26) f[3];
(D26)             1 - 2 (1 - 2 (1 - 2 %PI))
```

Podemos apresentar os resultados de maneira mais simplificada:

```
(C27) f[0];
(D27)                        %PI
(C28) f[1];
(D28)                     1 - 2 %PI
(C29) expand(f[2]);
(D29)                     4 %PI - 1
(C30) expand(f[3]);
(D30)                     3 - 8 %PI
```

Problema 4. *Seja Ω_A o conjunto de todas as palavras definidas sobre o alfabeto A. Para $A = \{a, b\}$ definimos a função $f : \Omega_A \to \Omega_A$ cujo valor em p é a palavra cujas letras são obtidas a partir das de p substituindo cada a por b e cada b por ab. Por exemplo, se $p = aaba$ então $f(p) = bbabb$. À custa da função f definimos recursivamente a seguinte sucessão:*

$$p_0 = p\,, \quad p_{n+1} = f(p_n)\,, \quad n \geq 0\,. \tag{4.7}$$

Por exemplo, para a condição inicial $p = a$ temos:

$$a\,, b\,, ab\,, bab\,, abbab\,, bababbab\,, \ldots \tag{4.8}$$

Pretendemos definir a sucessão (4.7) *em* Maxima*, representando as palavras como listas cujos elementos são as letras constituintes da palavra. Por exemplo a palavra $p = abbb$ será representada em* Maxima *como*

```
(C31) p: [a,b,b,b]$
```

A função f é definida em Maxima *à custa de uma* substituição simultânea*:*

```
(C32) f(p) := block( L:sublis([a=[b],b=[a,b]],p),
                     s:[],
                     for i from 1 thru length(L) do
                       s:append(s,L[i]),
                     s
                   )$
(C33) f([a,b,b]);
(D33)                   [b, a, b, a, b]
```

Não avance sem perceber porque é que uma substituição sequencial não faz o que se pretende! A construção da sucessão recursiva (4.8) *é obtida da maneira usual:*

```
(C34) [a];
(D34)                        [a]
(C35) f(%);
(D35)                        [b]
(C36) f(%);
(D36)                       [a, b]
(C37) f(%);
(D37)                      [b, a, b]
(C38) f(%);
(D38)                   [a, b, b, a, b]
(C39) f(%);
(D39)              [b, a, b, a, b, b, a, b]
```

No caso geral, uma sucessão definida recursivamente em $\mathbb{N}_0$ tem a seguinte estrutura:

$$x_n = f\left(x_{n-1}, x_{n-2}, \ldots, x_{n-k}, n\right)\,, \quad n \geq k\,. \tag{4.9}$$

A sucessão fica completamente especificada fixando os k valores iniciais x_0, …, x_{k-1}. O inteiro k define a *ordem* da recursividade. A função f diz-se *autónoma* quando não depende de n; *não-autónoma* no caso contrário.

Exemplo 33. *A sucessão factorial*

$$0! = 1$$
$$(n+1)! = (n+1)\,n!\,, \quad n \geq 0$$

é uma sucessão recursiva não-autónoma de primeira ordem. Na notação (4.9) *temos* $f(x_{n-1}, n) = n\,x_{n-1}$*. Esta sucessão pode ser definida da seguinte maneira:*

```
(C40) f[0]: 1$
(C41) f[n]:= if n >= 0 then n*f[n-1]$
(C42) f[0];
(D42)                          1
(C43) f[1];
(D43)                          1
(C44) f[10];
(D44)                       3628800
```

O Maxima disponibiliza a função `factorial`*:*

```
(C45) factorial(10);
(D45)                       3628800
```

Exemplo 34. *A sucessão dos números de Fibonacci é definida recursivamente como se segue:*

$$F_0 = 0\,, F_1 = 1$$
$$F_{n+1} = F_n + F_{n-1}\,, \quad n \geq 1\,.$$

Na notação (4.9) *temos* $f(x_{n-1}, x_{n-2}) = x_{n-1} + x_{n-2}$*: trata-se de uma sucessão recursiva autónoma de segunda ordem (*$k = 2$ *e* f *não depende directamente de* n*). Vamos calcular* F_{10} *com o Maxima:*

```
(C46) F[0]: 0$
(C47) F[1]: 1$
(C48) F[n]:= F[n-1] + F[n-2]$
(C49) F[10];
(D49)                          55
```

O Maxima disponibiliza a função `fib`*:*

```
(C50) fib(10);
(D50)                          55
```

4.5 Composição de Funções

Como observámos na página 46, numa sucessão recursiva

$$x_0 = \beta\,, \quad x_n = f\,(x_{n-1})\;, n \geq 1$$

o elemento x_n é obtido aplicando f a x_0 n vezes:

$$x_n = \underbrace{(f \circ f \circ \cdots \circ f)}_{n \text{ vezes}}(x_0)\,.$$

A composição de duas funções de uma variável f e g é feita em Maxima da maneira habitual:

```
(C1) f(x):= b*x*(1-x)$
(C2) g(x):= x^2$
(C3) g(f(x));
(D3)
```

$$b^2\left(1-x\right)^2x^2$$

```
(C4) f(g(x));
(D4)
```

$$bx^2\left(1-x\right)^2$$

Composições múltiplas da mesma função são obtidas em Maxima fazendo, por exemplo:

```
(C5) f(f(f(x)));
(D5)
```

$$b^3\left(1-x\right)x\left(1-b\left(1-x\right)x\right)\left(1-b^2\left(1-x\right)x\left(1-b\left(1-x\right)x\right)\right)$$

O valor de x_3 da sucessão do Exemplo 32 com $x_0 = -1$ é obtido fazendo

```
(C6) f(n):= -2*n + 1$
(C7) f(f(f(-1)));
(D7)                          11
```

De modo semelhante, o valor de p_5 para a sucessão (4.7) do Problema 4 com condição inicial $p_0 = [a]$ obtém-se fazendo:

```
(C8) f(p) := block( L:sublis([a=[b],b=[a,b]],p),
                    s:[],for i from 1 thru length(L) do
                           s:append(s,L[i]),s)$
(C9) f(f(f(f(f([a])))));
(D9)             [b, a, b, a, b, b, a, b]
```

Capítulo 5

Conclusão

O Maxima é um programa de matemática muito poderoso que pode ser usado na resolução dos mais variados problemas. Apesar do seu aspecto gráfico ficar um pouco aquém dos seus concorrentes comerciais, o uso do Maxima com o programa Texmacs permite obter um interface moderno WYSIWYG (What You See Is What You Get). Este programa pode ser obtido gratuitamente na página do Maxima [1].

Por ser open-source, tem a vantagem de poder ser copiado, modificado e distribuído livremente. Infelizmente, também significa que algumas funcionalidades podem não estar implementadas ou a funcionar correctamente. No entanto, como se encontra em constante desenvolvimento, tais falhas deverão ser corrigidas num futuro próximo.

Como o Maxima vem integrado com uma linguagem de programação de muito alto nível, as suas capacidades podem ser facilmente expandidas. Por exemplo, o Maxima não inclui, actualmente, nenhuma função para determinar o i-ésimo número primo. Porém, é possível construir, de forma simples, uma função com essa finalidade. Por exemplo:

```
ithprime(n):= block([I,count],
                    I:2, count:0,
                    do (if primep(I) then count: count + 1,
                    if count < n then I:I+1 else return(I)))$
```

Pode-se incluir esta função, ou qualquer outra, no próprio Maxima, por forma a disponibilizá-la sempre que necessário, sem ter de a reescrever:

Passo 1: Abrir o Maxima .

Passo 2: Escrever a função.

Passo 3: Gravar a função através do menu `File > Save Expressions to File`, e seleccionar a localização do ficheiro. Por exemplo, a pasta
`"C:\Programas\Maxima-5.9.1\share\maxima\5.9.1\share\contrib"`
com o nome de ficheiro "Prime.lisp".

Desta forma, para aceder a esta função no Maxima, deve-se primeiro carregá-la com o comando `LOAD(prime)`, e depois usá-la normalmente.

Em suma, o Maxima é potencialmente tão completo como as alternativas comerciais (como sejam o Maple ou o Mathematica), tendo a vantagem de ser gratuito.

Bibliografia

[1] *Web site do Maxima*, `maxima.sourceforge.net/`.

[2] M. Clarkson, *DOE-Maxima Reference Manual*, Version 5.9, 2002 (updated from [3] by Mike Clarkson) `starship.python.net/crew/mike/TixMaxima/macref.pdf`.

[3] W. Schelter, *Maxima Manual*, `maxima.sourceforge.net/docs/original/maximareforig.pdf`.

[4] P. N. de Souza, R. J. Fateman, J. Moses, C. Yapp, *The Maxima Book*, 19th September 2004, `maxima.sourceforge.net/docs/maximabook/maximabook-19-Sept-2004.pdf`.

[5] B. Willis, *Barton Willis Homepage*, `www.unk.edu/acad/math/people/willisb/`.

[6] *Symbolic Math - A Workflow*, `www.hippasus.com/resources/symmath/index.html`.

[7] *Apontamentos de Física dos Sistemas Dinâmicos*, `fisica.fe.up.pt/eic2107/apontamentos/`.

[8] F. Vivaldi, *Experimental Mathematics with Maple*, Chapman & Hall/CRC Mathematics, 2001.