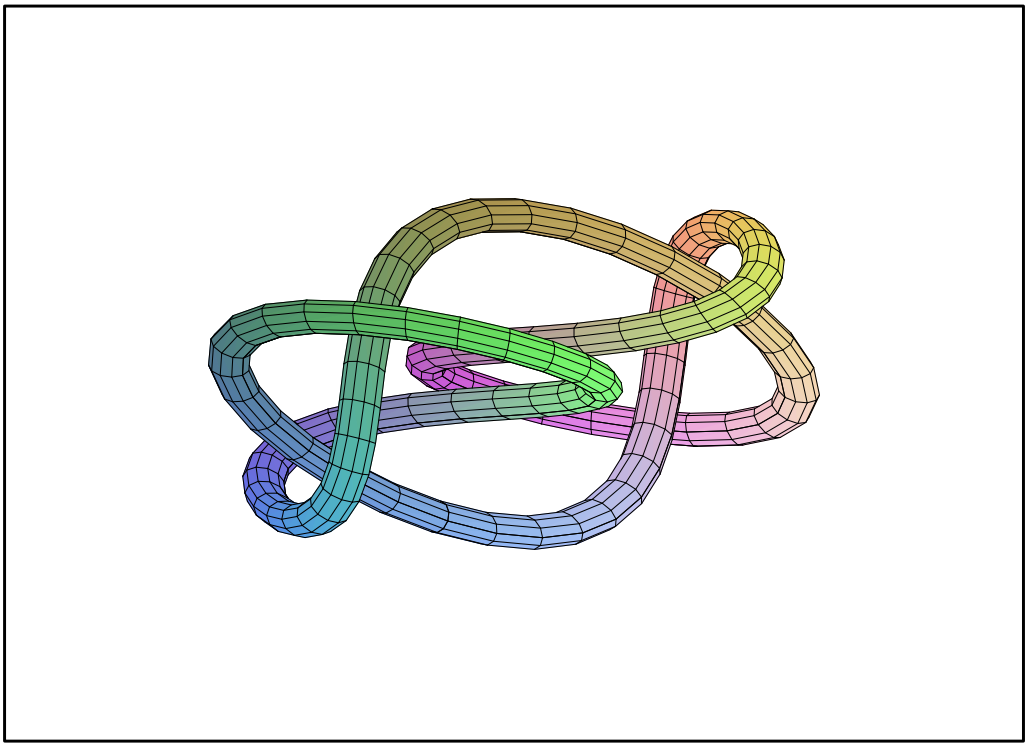
UNIVERSIDADE FEDERAL DA PARAÍBA
CENTRO DE CIÊNCIAS EXATAS E DA NATUREZA
DEPARTAMENTO DE MATEMÁTICA

Introdução à Computação Algébrica com o MAPLE



Lenimar Nunes de Andrade
lenimar@mat.ufpb.br
versão 1.1 – 24/julho/2003

Sumário

Prefácio

O Maple é um programa de Computação Algébrica de uso geral que vem sendo desenvolvido desde 1981 na Universidade de Waterloo, no Canadá.

Neste texto fazemos uma introdução geral a esse programa. Nosso principal objetivo é abordar a maior parte dos assuntos vistos nos cursos básicos das universidades: Cálculo Diferencial e Integral, Cálculo Vetorial, Álgebra Linear e Geometria Analítica. Assim, o texto pode ser um bom "companheiro" para quem estiver estudando essas disciplinas.

Os dois primeiros capítulos apresentam brevemente o ambiente de desenvolvimento do programa e os conceitos básicos necessários ao entendimento dos capítulos seguintes. O capítulo 3 trata dos assuntos básicos da Álgebra Linear. O capítulo 4 é sobre a construção de diversos tipos de gráficos planos e tridimensionais e animações gráficas. Os capítulos 5 e 6 abordam diversos assuntos do Cálculo e da Geometria Analítica. O capítulo 7 é específico de Equações Diferenciais, incluindo gráficos e sistemas. Por fim, o capítulo 8 é sobre o uso do Maple como uma linguagem de programação – um tema sobre o qual o leitor precisará ter um pouco mais de dedicação para poder dominá-lo completamente.

O conteúdo de cada um desses capítulos poderia ser expandido ao ponto de se transformar em um livro independente. Não haveria condições, nem era esse o nosso objetivo, esgotar algum desses conteúdos em uma modesta quantidade de páginas.

Ao final de cada capítulo colocamos alguns exercícios. Demos preferência a exercícios que pudessem chamar a atenção do leitor, sempre que isso foi possível. Sugerimos que o leitor tente resolver uma boa parte deles usando o programa, principalmente aqueles que parecerem mais desafiadores. Alguns exercícios são bem imediatos, enquanto que outros exigem um pouco mais de experiência com o uso do programa.

Há muito sobre Computação Algébrica e sobre o Maple na Internet, conforme podemos ver nos *sites* cujos endereços são www.SymbolicNet.org, do *Symbolic Mathematical Computation Information Center,* e www.mapleapps.com, do *Maple Application Center.*

O professor Paulo Antônio Fonseca Machado, da Universidade Federal de Minas Gerais, apresentou várias sugestões e recomendou algumas correções na versão preliminar que muito contribuíram para que o texto tivesse estes formato e conteúdo atuais. Nosso agradecimento ao professor Paulo pelos seus comentários.

Gostaria de agradecer também aos meus colegas Antônio Sales da Silva, Sérgio de Albuquerque Souza e Abdoral de Souza Oliveira, professores da Universidade Federal da Paraíba, que nos incentivaram e apresentaram valiosas sugestões.

João Pessoa, 13 de maio de 2003

Lenimar Nunes de Andrade

Capítulo 1

Introdução ao Maple

Neste capítulo apresentamos brevemente o ambiente de desenvolvimento integrado do Maple e algumas de suas funções e comandos básicos. Juntamente com o capítulo 2, que complementa o que está sendo apresentado aqui, constitui a base para o entendimento dos demais capítulos.

1.1 Introdução

O Maple é um programa de Computação Algébrica de uso geral que possui inúmeros recursos numéricos e gráficos, além de também funcionar como uma linguagem de programação. Ele vem sendo desenvolvido desde 1981 pela *Waterloo Maple Inc.* para vários sistemas operacionais. Com ele, é possível realizar cálculos que contenham símbolos como π, ∞ ou $\sqrt{2}$ sem a necessidade de fazer aproximações numéricas ou realizar simplificações e cálculos com expressões algébricas como $ax^2 + bx + c$ ou $x^3 + \log(x)$ sem ser preciso atribuir valores numéricos às variáveis ou constantes. Devido a essas propriedades, é possível encontrar soluções exatas para problemas práticos que envolvam resolução de equações, derivadas, integrais, cálculo matricial, etc, tudo isso integrado a recursos que permitem visualização de dados ou objetos planos ou tridimensionais.

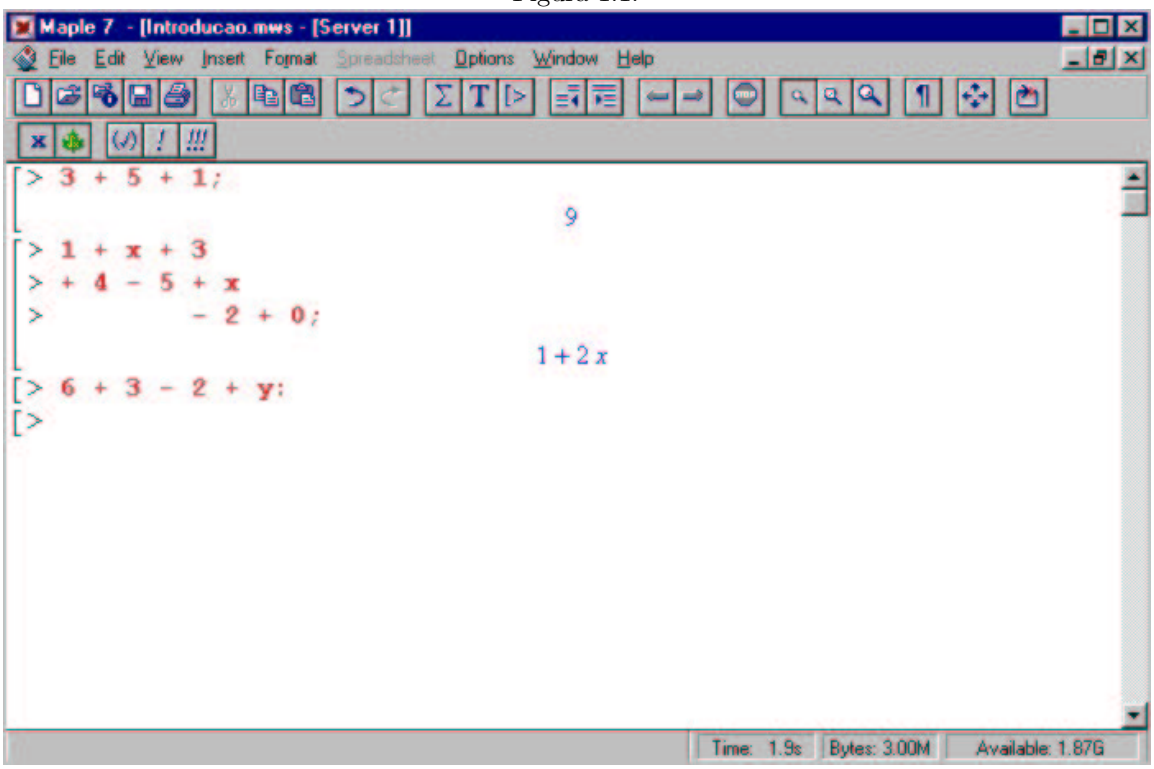
Na versão 7 para Windows, quando ele é chamado, aparece uma tela como a da Figura 1.1. Funcionando de modo interativo, o usuário vai digitando comandos ao lado de um aviso (*prompt*) cujo formato padrão é o símbolo $>$. Assim que um comando digitado sem erros é encerrado, o núcleo do programa o executa. Em outros sistemas operacionais (como o Linux) podemos obter telas muito parecidas com essa.

Os comandos são digitados em um ambiente próprio chamado *folha de trabalho* (*worksheet*) que pode ser gravada em disco através da opção File/Save ou File/Save As do menu principal ou ser carregada do disco com a opção File/Open.

É possível também trabalhar sem o ambiente de janelas, só com a linha de comandos. Nesse caso, é mostrada uma mensagem de direitos autorais como a mostrada a seguir e uma linha iniciando com o símbolo $>$ onde os comandos podem ser digitados.

```
    |\^/|      Maple 7 Diagnostic Program
._|\|   |/|_.  Copyright (c) 1981-2001 by Waterloo Maple Inc. All rights
 \  MINT   /   reserved. Maple and Maple V are registered trademarks of
  <____ ____>  Waterloo Maple Inc.
       |
>
```

Figura 1.1:



Cada comando digitado deve terminar com um ";" (ponto e vírgula) ou com ":" (dois pontos), seguido de Enter. Se o comando terminar com ponto e vírgula, o resultado da sua execução será mostrado logo em seguida. Se terminar com dois pontos, o resultado não será mostrado, podendo ficar guardado para uso posterior. A digitação de um comando pode se estender por mais de uma linha.

O Maple é sensível ao tipo das letras, ou seja, ele diferencia letras minúsculas das respectivas letras maiúsculas. Por exemplo, "x" é considerado diferente de "X".

Exemplo 1.1 *Ao lado do aviso do Maple digitamos o comando "*$3+5+1$*" e encerramos a linha com um ponto e vírgula (veja a Figura 1.1). Ele calcula a soma e mostra imediatamente o resultado:*

```
> 3 + 5 + 1;
```

$$9$$

Se não for digitado ponto e vírgula e nem dois pontos no final da linha, então o Maple convenciona que o comando continua na linha seguinte. Aqui calculamos o valor de "$1+x+3+4-5+x-2+0$*" digitado em três linhas:*

```
> 1 + x + 3
> + 4 - 5 + x
>         - 2 + 0;
```

$$1+2x$$

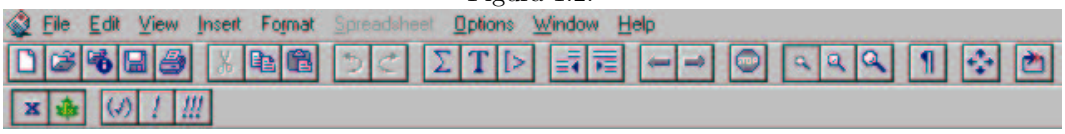
Se o comando for encerrado com dois pontos, então o resultado obtido não é mostrado de imediato:

```
> 6 + 3 - 2 + y:
```

1.2 O menu principal

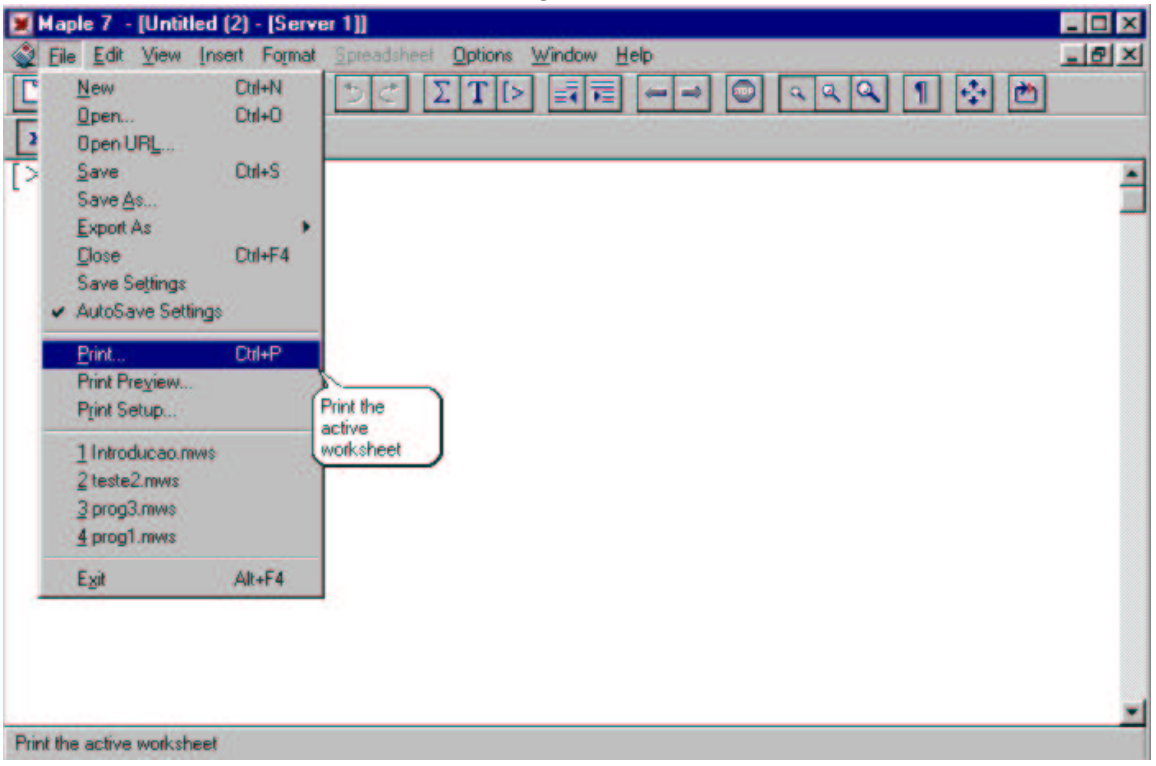
Na apresentação padrão, na parte superior da tela aparecem o menu principal, a barra de ferramentas e a barra de contexto (Figura 1.2). A barra de contexto é alterada de acordo com o que o programa estiver mostrando no momento.

Figura 1.2:



O menu principal corresponde à primeira linha da parte superior da tela (Figura 1.2) e é formado por vários submenus: File, Edit, View, Insert, Format, Spreadsheet, Options, Window e Help. Para ter acesso a um item do menu, deve-se pressionar com o mouse em cima do item ou pressionar no teclado simultaneamente as teclas Alt e a letra que estiver sendo mostrada sublinhada no nome do item desejado. Por exemplo, para ter acesso ao primeiro item, o File, pressione com o mouse em cima dele ou pressione no teclado as teclas Alt e F simultaneamente (Figura 1.3).

Figura 1.3:



1.3 A barra de ferramentas

As principais opções do menu principal também podem ser executadas pressionando-se os botões da barra de ferramentas.

 Os quatro primeiros botões da barra de ferramentas correspondem às operações de folha de trabalho em branco, abrir uma folha já existente, salvar e

imprimir a folha atual;

 os próximos botões correspondem às operações de recortar, copiar e colar;

 os próximos dois botões correspondem a desfazer e refazer a última correção realizada;

 os próximos três botões correspondem ao modo de execução do Maple: insere fórmulas matemáticas sem executá-las, insere texto e insere comando executável;

 os próximos dois botões correspondem a não *endentar* ou *endentar* o texto digitado.

 O par de setas leva para a anterior ou para a próxima conexão do texto.

 O botão STOP é o botão que pára o que estiver sendo executado.

 Os três botões com lupas correspondem à ampliação ou redução (mais zoom ou menos zoom) do que estiver sendo mostrado;

 os três últimos botões correspondem a mostrar caracteres não imprimíveis, redimensionamento de janela e botão de recomeçar (*restart*).

1.4 Usando as telas de ajuda

As telas de ajuda (*help*) do programa são a maneira mais completa de se obter informações sobre ele (Figura 1.4). São cerca de 3000 páginas de informações. Pode-se ter acesso a elas através da opção Help do menu principal ou digitando-se ao lado do aviso do programa um interrogação seguido de Enter.

Para obter informações específicas sobre algum comando, basta digitar um interrogação seguido do nome do comando ou pressionar a tecla sf F1 em cima do nome do comando. Por exemplo, para obter informação específica sobre a função trigonométrica cosseno, basta digitar:

```
> ?cos
```

Em geral, as telas de ajuda contém uma descrição detalhada em inglês do comando, com exemplos e referências a comandos relacionados.

1.5 Usando as palhetas

Uma outra alternativa de entrada de dados é através do uso de palhetas (*palettes*). O Maple possui quatro palhetas: uma com símbolos, uma com vetores, uma com matrizes e outra com expressões (Figura 1.5).

O acesso às palhetas se dá através da opção View/Palettes do menu principal.

Usar uma palheta é como preencher um formulário em branco, basta ir escrevendo nos campos marcados com um %? que são mostrados, pressionando a tecla Tab para pular

Figura 1.4:

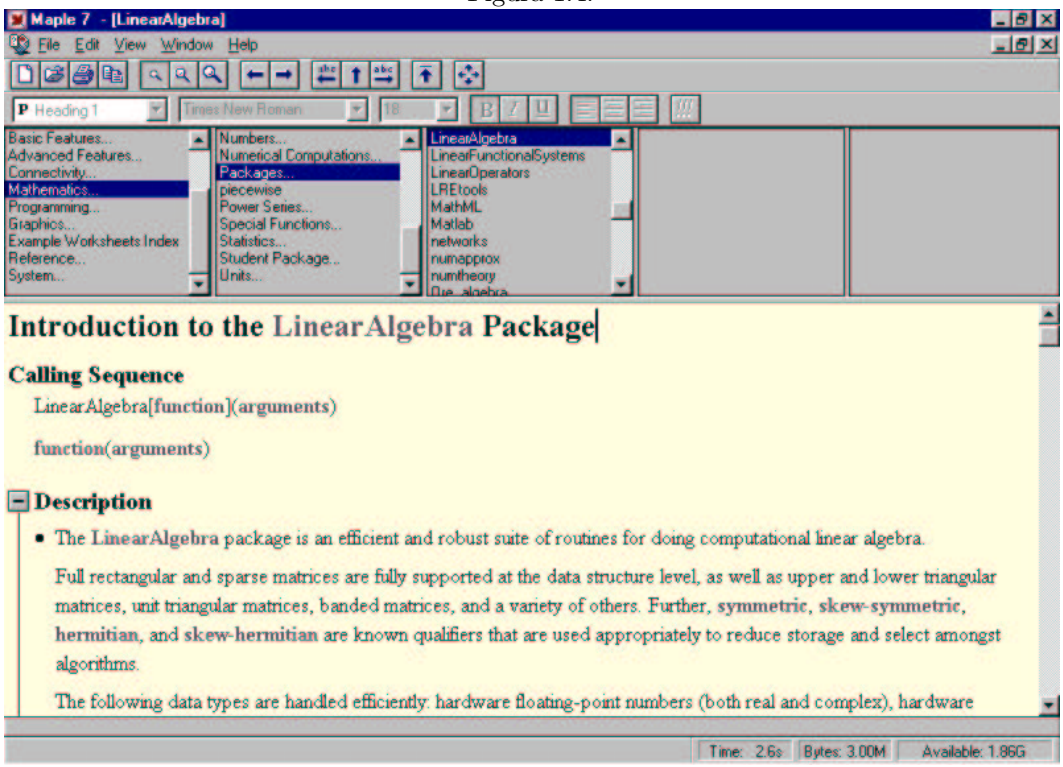
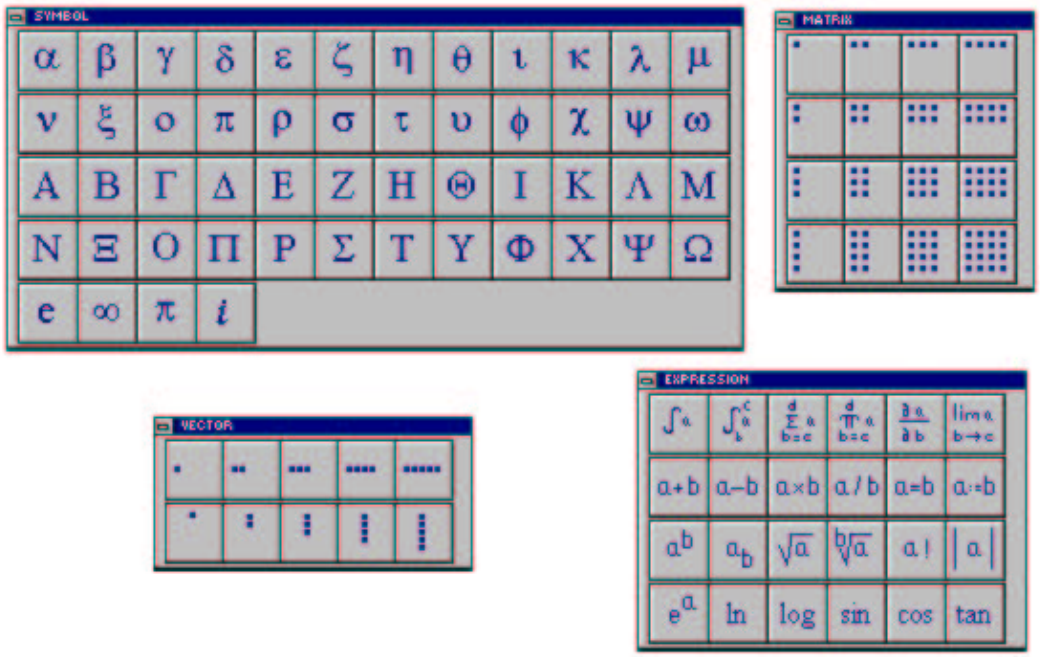


Figura 1.5:



para o campo seguinte ou Shift Tab para voltar ao campo anterior. Deve-se pressionar Enter no final.

1.6 Comentários

Na linha de comando do Maple, tudo o que for digitado à direita do símbolo "#" será considerado um comentário. Os comentários são ignorados na execução do programa, servindo só para orientação do usuário.

Exemplo 1.2 *Exemplo de comentário na digitação da folha de trabalho:*

```
> # Este é um exemplo de comentário, não é considerado
> # como um comando para ser executado.
```

1.7 Operações aritméticas e tipos numéricos básicos

As operações aritméticas adição, subtração, multiplicação, divisão e potenciação são representadas por $+, -, *, /$ e ^ , respectivamente.

A prioridade no cálculo das operações é a mesma usada na Matemática: primeiro o que estiver entre parênteses, depois as potenciações, depois as multiplicações e divisões e por último as adições e subtrações.

Exemplo 1.3 *Exemplificando o uso das operações aritméticas básicas.*

EXPRESSÃO	*VALOR*	*EXPRESSÃO*	*VALOR*
*1 + 2*3*	*7*	*4*3^2*	*36*
*(1 + 2)*3*	*9*	*(4*3)^2*	*144*
6 + 9/3 + 2	*11*	*(6 + 9)/(3 + 2)*	*3*
*2/(1 + 2^(3 - 5)*8)*	*2/3*	*3^2 - 2^3*	*1*

Cada objeto algébrico, numérico ou gráfico no Maple possui um tipo. O tipo é uma espécie de "família do objeto", agrupando objetos de propriedades semelhantes.

Entre os tipos numéricos básicos podemos destacar o tipo inteiro (`integer`) e o tipo real com ponto flutuante (`float`). Um número formado por uma seqüência de algarismos sem espaços em branco e sem ponto entre eles é considerado como sendo do tipo inteiro. Um número com ponto decimal flutuante é considerado como sendo do tipo real. Por exemplo, 144 é considerado inteiro, enquanto que 144.00 é considerado real.

É importante observar a diferença entre reais e inteiros porque o programa calcula imediatamente as expressões que contiverem números reais, mas mantém em forma inercial (sem calcular) algumas expressões que envolvam exclusivamente números inteiros. Por exemplo 2/3 (divisão de inteiros) é mantido em forma de fração, mas 2.0/3.0 (divisão de reais) é aproximado imediatamente para 0.6666666667, onde ele aparecer.

1.8 Constantes

Algumas constantes pré-definidas são `Pi` (o famoso $\pi = 3,14159\ldots$), `gamma` ($\gamma = 0,57721\ldots$) e I (unidade imaginária). Mas, cuidado para não usar `pi` no lugar de `Pi` e nem i no lugar de I porque não são a mesma coisa.

Também são consideradas constantes `true` e `false` (verdadeiro e falso) e `infinity` representando o infinito.

A base do logaritmo natural ($e = 2,71828\ldots$) deve ser usada como sendo exp(1).

1.9 Variáveis

Uma variável é um lugar na memória identificado por um nome e serve para "guardar valores". Esse nome pode ser formado por letras minúsculas ou maiúsculas, algarismos e o caracter sublinhado. O nome da variável não pode começar com um algarismo.

São exemplos de nomes de variáveis válidos: x, $y2$, $var11$, $Teste$, $v\_inicial$.

São exemplos de nomes inválidos: $x\#$, $2y$, $var.11$, $v–inicial$.

1.10 Atribuições

Um valor pode ser atribuído a uma variável com um comando *variável* := *valor*. Por exemplo, $x := 2$ atribui o valor 2 à variável x. Cuidado para não confundir com a igualdade $x = 2$ que compara x com 2 (ver seção 1.12).

A atribuição $x := f(x)$ redefine o valor da variável x como sendo o valor da função f calculado no valor anterior de x. Por exemplo, uma atribuição como $x := x + 1$ faz com que o valor da variável x seja substituído pelo valor que o x já tinha acrescentado de 1.

Ao digitar o nome de uma variável à direita do aviso do Maple (o sinal de maior), seguido de um ponto e vírgula, em muitos casos faz o programa mostrar o seu valor.

Exemplo 1.4 *Inicialmente definimos o valor de x como sendo 2.*

```
> x := 2;
```

$$x := 2$$

```
> x;    # mostra o valor de x
```

$$2$$

Agora, acrescentamos 1 ao valor de x:

```
> x := x + 1;
```

$$x := 3$$

Comparamos o x com 4 (o x continua valendo 3):

```
> x = 4;
```

$$3 = 4$$

Essa última igualdade deve ser entendida como uma expressão lógica a qual o Maple atribui um valor $false$ (falso).

Podemos fazer várias atribuições simultaneamente. Para isso, basta listar as variáveis separadas por vírgulas e atribuí-las uma lista de valores também separados por vírgulas. Por exemplo, para atribuir simultaneamente os valores $1, 2, 3$ às variáveis a, b, c, respectivamente, basta digitar o comando de atribuição: $a, b, c := 1, 2, 3$.

Para fazer com que o programa "esqueça" o valor de uma variável x, deve-se fazer uma atribuiçaõ: $x := 'x'$ ou usar um comando `unassign('x')`. É possível usar o `unassign` com várias variáveis: `unassign('var1', 'var2', 'var3', ...)`.

Para fazer com que o programa "esqueça" tudo o que foi atribuído anteriormente, basta usar um comando `restart`.

Exemplo 1.5 *Neste exemplo, "limpamos" inicialmente a lista de variáveis do Maple com um* `restart`*. Depois fazemos uma atribuição múltipla de valores a quatro variáveis $x, y, teste1$ e $teste2$.*

```
> restart;  # equivale a pressionar o restart da barra de ferramentas
> x, y, teste1, teste2 := -1, -2, a, b;
```

$$x,\ y,\ teste1,\ teste2\ :=\ -1,\ -2,\ a,\ b$$

Agora, conferimos o valor atribuído a cada variável, uma por uma:

```
> x;
```

$$-1$$

```
> y;
```

$$-2$$

```
> teste1;
```

$$a$$

```
> teste2;
```

$$b$$

*"Limpamos" as variáveis teste*1 *e teste*2*:*

```
> unassign('teste1', 'teste2');
> teste1;
```

$$teste1$$

```
> teste2;
```

$$teste2$$

E, finalmente, "limpamos" os valores de x *e* y*:*

```
> x, y := 'x', 'y';
```

$$x, y := x, y$$

```
> x;
```

$$x$$

```
> y;
```

$$y$$

1.11 Avaliação numérica

A avaliação numérica de uma expressão X é feita com o comando `evalf(`X`)`. O Maple usa como padrão uma quantidade de 10 algarismos significativos, mas é possível obter resultado com qualquer quantidade de algarismos significativos, bastando para isso fazer o seguinte:

- usar o `evalf` na forma `evalf(`X, n`)`. Isso mostra X com n algarismos significativos.
- usar um comando do tipo `Digits :=` n. A partir daí, todos os cálculos numéricos são mostrados com n algarismos significativos. `Digits` é uma variável pré-definida do programa que controla a quantidade de algarismos significativos utilizadas.

Exemplo 1.6 *Neste exemplo mostramos os valores de* π *e* $\sqrt{2}$ *com vários algarismos significativos.*

```
> Digits;
```

$$10$$

```
> Pi;
```

$$\pi$$

```
> evalf(Pi);
```

$$3.141592654$$

```
> sqrt(2);
```

$$\sqrt{2}$$

```
> evalf(sqrt(2));
```

$$1.414213562$$

Uma atribuição Digits := 30 *faz com que os resultados passem a ser mostrados com um novo padrão de 30 algarismos significativos.*

```
> Digits := 30;
```

$$Digits := 30$$

```
> evalf(Pi);
```

$$3.14159265358979323846264338328$$

```
> evalf(sqrt(2));
```

$$1.41421356237309504880168872421$$

Mas, com o `evalf` *pode-se escolher a quantidade de algarismos significativos (diferente do padrão, possivelmente) com o qual determinado resultado é mostrado:*

```
> evalf(sqrt(2), 60);
```

$$1.41421356237309504880168872420969807856967187537694807317668$$

E, finalmente, o valor de π *com duzentos algarismos significativos! Observe que o Maple coloca uma barra invertida no final da linha da resposta para indicar que ela continua na próxima.*

```
> evalf(Pi, 200);
```

$$3.14159265358979323846264338327950288419716939937510582097494459230781640628620899862803482534211706798214808651328230664709384460955058223\backslash$$
$$17253594081284811174502841027019385211055596446229489549303820$$

1.12 Expressões lógicas

Uma expressão lógica, avaliada em *true* ou $false$ (verdadeiro ou falso), contém operadores relacionais e operadores lógicos.

Os operadores relacionais são = (igual), < (menor), > (maior), <=(menor ou igual), >= (maior ou igual) e <> (diferente) e os operadores lógicos são `not` (não), `and` (e), `or` (ou), `xor` (ou exclusivo) e `implies` (implica).

A ordem decrescente de prioridade na avaliação de uma expressão lógica é:

1. <, <=, >, >=, =, <>
2. `not`
3. `and`
4. `or`
5. `xor`
6. `implies`

Além de *true* e $false$, uma operação lógica também pode retornar um valor $FAIL$ para indicar que houve algum tipo de falha no cálculo do valor da expressão.

Uma *expressão* lógica pode ser calculada com um comando `evalb(`*expressão*`)` ou `is(`*expressão*`)`, conforme mostramos a seguir.

Exemplo 1.7 *Avaliando algumas expressões lógicas com os comandos* `evalb` *ou* `is`.

```
> evalb(1 < 2);
```

$$true$$

```
> evalb(1 < 2 and 2 = 3);
```

$$false$$

```
> evalb(4 = 5 or 3 = 3);
```

$$true$$

Algumas expressões formadas por constantes e operadores lógicos podem ser avaliadas sem ser necessário usar `evalb` *ou outro comando:*

```
> 4 < 3 implies 4 = 9;
```

$$true$$

```
> 1 < 2 xor 3 <> 9;
```

$$false$$

```
> 3 < 4 implies 4 = 9;
```

$$false$$

O comando `is` *às vezes é mais eficiente do que o* `evalb`*, conforme mostramos a seguir. O* `is` *desenvolve e simplifica uma expressão algébrica, coisa que o* `evalb` *não faz.*

```
> evalb( a^2 - b^2 = (a + b)*(a - b));
```

$$false$$

```
> is(a^2 - b^2 = (a + b)*(a - b));
```

true

Exemplo 1.8 *Exemplificando o uso das operações lógicas básicas. Foi usado um comando* `evalb(`*expressão*`)` *para calcular o valor lógico de cada expressão.*

EXPRESSÃO	*VALOR*	*EXPRESSÃO*	*VALOR*
$1 < 3$	*true*	not $(1 < 3)$	*false*
$2 = 2$	*true*	$2 <> 2$	*false*
$3 < 5$ or $3 = 5$	*true*	$3 < 5$ and $3 = 5$	*false*
$4 <= 5$ and not $3 <> 3$	*true*	not $3 > 1$ and not $3 < 1$	*false*
$3 = 3$ implies $4 = 4$	*true*	$3 = 3$ xor $4 = 4$	*false*

1.13 Funções matemáticas

O Maple possui muitas funções matemáticas pré-definidas, não só as funções elementares básicas, como também muitas funções especiais como as funções beta, gama, logaritmo integral, seno integral, dilogaritmo, funções de Bessel, etc.

Listamos nesta seção somente uma pequena parte das funções básicas.

1.13.1 Funções básicas

FUNÇÃO	*DESCRIÇÃO*	*EXEMPLO*
abs(x)	Valor absoluto (módulo) de x	abs$(-3) = 3$
sqrt(x)	Raiz quadrada de x	sqrt$(16) = 4$
root[n](x)	Raiz de índice n de x	root[3]$(8) = 2$
surd(x, n)	Raiz de índice n de x	surd$(-27,\ 3) = -3$
exp(x)	Exponencial de x	exp$(4) = e^4$
ln(x)	Logaritmo natural de x	ln$(e) = 1$
log[b](x)	Logaritmo de x na base b	log[2]$(8) = 3$
log10(x)	Logaritmo decimal de x	log10$(1000) = 3$
binomial(n, r)	Coeficiente binomial n sobre r	binomial$(5, 2) = 10$
factorial(n)	Fatorial de n; o mesmo que $n!$	$5! = 120$
igcd($m, n, p, \dots$)	Máximo divisor comum	igcd$(6, 14) = 2$
ilcm($m, n, p, \dots$)	Mínimo múltiplo comum	ilcm$(3, 5) = 15$
max($x, y, z, \dots$)	Máximo de $\{x, y, z, \dots\}$	max$(-2, -3, 1) = 1$
min($x, y, z, \dots$)	Mínimo de $\{x, y, z, \dots\}$	min$(-2, -3, 1) = -3$
signum(x)	Sinal de x	signum$(-7) = -1$
ceil(x)	Menor inteiro maior ou igual a x	ceil$(2.7) = 3$
floor(x)	Maior inteiro menor ou igual a x	floor$(2.7) = 2$
round(x)	Inteiro mais próximo de x	round$(2.7) = 3$
trunc(x)	Parte inteira de x	trunc$(2.7) = 2$
frac(x)	Parte fracionária de x	frac$(2.7) = 0.7$

1.13.2 Funções trigonométricas e hiperbólicas

FUNÇÃO	*DESCRIÇÃO*	*FUNÇÃO*	*DESCRIÇÃO*
$\sin(x)$	Seno de x	$\sinh(x)$	Seno hiperbólico de x
$\cos(x)$	Cosseno de x	$\cosh(x)$	Cosseno hiperbólico de x
$\tan(x)$	Tangente de x	$\tanh(x)$	Tangente hiperbólica de x
$\sec(x)$	Secante de x	$\mathrm{sech}(x)$	Secante hiperbólica de x
$\csc(x)$	Cossecante de x	$\mathrm{csch}(x)$	Cossecante hiperbólica de x
$\cot(x)$	Cotangente de x	$\coth(x)$	Cotangente hiperbólica de x
$\arcsin(x)$	Arco-seno de x	$\mathrm{arcsinh}(x)$	Arco-seno hiperbólico de x
$\arccos(x)$	Arco-cosseno de x	$\mathrm{arccosh}(x)$	Arco-cosseno hiperbólico de x
$\arctan(x)$	Arco-tangente de x	$\mathrm{arctanh}(x)$	Arco-tangente hiperbólico de x
$\mathrm{arcsec}(x)$	Arco-secante de x	$\mathrm{arcsech}(x)$	Arco-secante hiperbólico de x
$\mathrm{arccsc}(x)$	Arco-cossecante de x	$\mathrm{arccsch}(x)$	Arco-cossecante hiperbólico de x
$\mathrm{arccot}(x)$	Arco-cotangente de x	$\mathrm{arccoth}(x)$	Arco-cotangente hiperbólico de x

1.13.3 Funções complexas

O Maple dispõe de um conjunto de funções básicas para manipulação de números complexos.

evalc(expressão) Calcula valor numérico complexo da expressão dada

abs(z) Valor absoluto de **z**

argument(z) Argumento de **z**

conjugate(z) Conjugado de **z**

polar(z) Forma polar de **z**

Re(z) Parte real de **z**

Im(z) Parte imaginária de **z**

Além disso, as operações aritméticas e funções básicas como raiz quadrada, exponencial, logaritmo, trigonométricas, hiperbólicas e suas inversas podem ter argumentos complexos.

Exemplo 1.9 *Exemplificamos algumas funções e operações com números complexos. A unidade imaginária é representada por I.*

```
> Re(3 + 4*I);
```

$$3$$

```
> Im(3 + 4*I);
```

$$4$$

```
> abs(3 + 4*I);
```

$$5$$

```
> argument(3 + 4*I);
```

$$\arctan\left(\frac{4}{3}\right)$$

```
> conjugate(3 + 4*I);
```

$$3 - 4I$$

```
> polar(3 + 4*I);
```

$$\text{polar}\left(5, \arctan\left(\frac{4}{3}\right)\right)$$

```
> evalc(polar(5, arctan(4/3)));
```

$$3 + 4I$$

```
> sqrt(-25);  # raiz quadrada de -25
```

$$5I$$

```
> root[3](-8.0);  # raiz cúbica de -8.0
```

$$1.000000000 + 1.732050807I$$

```
>  (1 + I)^20 + (1 - I)^20;  # soma de potências
```

$$-2048$$

```
> evalc(2^I);  # 2 elevado a i
```

$$\cos(\ln(2)) + I\sin(\ln(2))$$

```
> ln(-1);   # logaritmo natural de -1
```

$$I\pi$$

```
> evalc(arcsin(2));  # arco-seno de 2
```

$$12\pi - I\ln(2 + \sqrt{3})$$

```
> Re(cos(3 + 4*I));  # Parte real do cosseno de 3+4i
```

$$\cos(3)\cosh(4)$$

```
> evalc(sin(1 + 7*I));  # seno de 1+7i
```

$$\sin(1)\cosh(7) + I\cos(1)\sinh(7)$$

1.14 Aspas e apóstrofos

No Maple as *aspas duplas* `"` , o *apóstrofo* `'` e o *apóstrofo invertido* (crase) `` ` `` são utilizados com objetivos diferentes.

- as aspas são usadas para denotar seqüências de caracteres, conhecidas como *strings*. Exemplo: `X := "mensagem do tipo string"`.
- as crases podem ser usadas em nomes de variáveis. Exemplo: `teste` := 2 faz o mesmo efeito que `teste := 2`.
- os apóstrofos são utilizados para retardar a execução de comandos ou retardar substituição de valores. Exemplo: $y :=$ 'x' faz a atribuição de x a y, e não a atribuição do valor de x a y.

Exemplo 1.10

```
> restart;
> x := 9; y := 4;
```

$$x := 9$$

$$y := 4$$

```
> r := x + y;
```

$$r := 13$$

```
> s := 'x + y';
```

$$s := x + y$$

```
> t := 'x' + y;
```

$$t := x + 4$$

```
> u := x + 'y';
```

$$u := 9 + y$$

1.15 Seqüências, conjuntos e listas

Uma *seqüência de expressões* (`exprseq`) é uma relação de constantes ou expressões separadas por vírgulas. Por exemplo `1, 2, x, x+1, y, cos(x), ln(2)` é uma seqüência.

Uma lista (`list`) é uma seqüência entre colchetes e um conjunto (`set`) é uma seqüência entre chaves. Por exemplo, `[1, 2, x, x+1, y, cos(x), ln(2)]` é uma lista, enquanto que {`1, 2, x, x+1, y, cos(x), ln(2)`} é um conjunto.

Esses objetos diferenciam-se entre si pelas seguintes propriedades:

- Em um conjunto não importa a ordem dos elementos. Nas listas e nas seqüências a ordem dos elementos é importante.
- As listas e as seqüências podem ter elementos repetidos. Nos conjuntos, a repetição de elementos é automaticamente eliminada.
- Uma seqüência de seqüências é uma seqüência.

A quantidade de elementos de uma lista (ou conjunto) X é dada por `nops(`X`)`.

Exemplo 1.11 *Definimos uma seqüência X e, a partir dela, definimos uma lista Y e um conjunto Z. No final, calculamos a quantidade de elementos de Y e de Z.*

```
> X := 1, 1, 3, x, y, z, z, z;
```

$$X := 1, 1, 3, x, y, z, z, z$$

```
> Y := [X];
```

$$Y := [1, 1, 3, x, y, z, z, z]$$

```
> Z := {X};
```

$$Z := \{1, 3, x, y, z\}$$

```
> nops(Y);
```

$$8$$

```
> nops(Z);
```

$$5$$

Exemplo 1.12 *Neste exemplo, definimos inicialmente três seqüências X, Y e Z. A seqüência Z é formada por letras gregas que devem ser digitadas usando-se seus nomes ou pressionando-se botões da palheta de símbolos.*

```
> X := a, b, c:
> Y := m, n, p:
> Z := epsilon, delta, kappa:
```

A partir das seqüências definidas, construímos uma seqüência de seqüências S, uma lista de listas L e um conjunto de conjuntos C.

```
> S := X, Y, Z;  # seqüência de seqüências
```

$$S := a, b, c, m, n, p, \epsilon, \delta, \kappa$$

```
> L := [[X], [Y], [Z]]; # lista de listas
```

$$L := [[a, b, c], [m, n, p], [\epsilon, \delta, \kappa]]$$

```
> C := {{X}, {Y}, {Z}}; # conjunto de conjuntos
```

$$C := \{\{a, c, b\}, \{m, n, p\}, \{\epsilon, \delta, \kappa\}\}$$

Algumas operações podem ser realizadas com conjuntos como `union` (união), `intersect` (interseção), `minus` (diferença) e `subset` (subconjunto).

Exemplo 1.13 *Definimos dois conjuntos A e B e calculamos $X = A \cup B$, $Y = A \cap B$, $Z = A - B$ e testamos se $Y \subset A$.*

```
> restart;
> A := {1, 2, 3, 4}: B := {3, 4, 5, 6}:
> X := A union B;
```

$$X := \{1, 2, 3, 4, 5, 6\}$$

```
> Y := A intersect B;
```

$$Y := \{3, 4\}$$

```
> Z := A minus B;
```

$$Z := \{1, 2\}$$

```
> Y subset A;
```

$$true$$

O i-ésimo elemento de uma lista (ou conjunto) X é dado por `op(`i, X`)` ou por X`[`i`]`. No lugar de um único valor i, podemos ter um intervalo de valores inteiros de i a j indicado por $i..j$.

Exemplo 1.14 *Inicialmente definimos uma lista X formada por expressões algébricas.*

```
> X := [sin(alpha), cos(beta), -1, 0, 1, a, b, c, 1+x+x^2, 1/y];
```

$$X := \left[\sin(\alpha),\ \cos(\beta),\ -1,\ 0,\ 1,\ a,\ b,\ c,\ 1 + x + x^2,\ \frac{1}{y}\right]$$

```
> X[1];
```

$$\sin(\alpha)$$

```
> op(1, X);
```

$$\sin(\alpha)$$

```
> X[3];
```

$$-1$$

```
> op(3, X);
```

$$-1$$

Definimos Y como sendo uma lista formada pelo oitavo, nono e décimo elementos de X:

```
> Y := X[8..10];
```

$$Y := \left[c,\ 1 + x + x^2,\ \frac{1}{y}\right]$$

```
> X[1..4];
```

$$[\sin(\alpha), \cos(\beta), -1, 0]$$

Finalmente, definimos Z como sendo uma seqüência formada pelos quatro primeiros elementos de X:

```
> Z := op(1..4, X);
```

$$Z := \sin(\alpha),\ \cos(\beta),\ -1,\ 0$$

O operador \$ pode ser usado para indicar repetição de elementos. Por exemplo, $x\$n$ corresponde a n termos consecutivos iguais a x.

Exemplo 1.15 *Definimos uma lista L e uma seqüência S com repetição de vários elementos.*

```
> L := [x$3, y$4, z, w$4];
```

$$L := [x, x, x, y, y, y, y, z, w, w, w, w]$$

```
> S := 1$4, -3$2, Pi$5;
```

$$S := 1, 1, 1, 1, -3, -3, \pi, \pi, \pi, \pi, \pi$$

O conjunto vazio é representado por { } e uma lista vazia por [ ].

1.16 Concatenação

O operador || é usado para concatenar nomes. Desse modo, $x||y$ é um novo nome de variável chamado xy (que não tem nada a ver com o produto de x por y).

Podemos também usar $nome1 \,||\, nome2 \,||\, nome3 \,||\, \ldots$ na forma

$$\texttt{cat}(nome1,\ nome2,\ nome3,\ \ldots).$$

Exemplo 1.16 *Usamos o operador de concatenação para definir as variáveis* `mens` *e* `teste`.

```
> x := 100; y := -45;
```

$$x := 100$$

$$y := -45$$

```
> mens := "O valor de x é " || x || " e o valor de y é " || y;
```

$$mens := \text{“}O\ valor\ de\ x\ é\ 100\ e\ o\ valor\ de\ y\ é - 45\text{”}$$

```
> mens;
```

$$\text{“}O\ valor\ de\ x\ é\ 100\ e\ o\ valor\ de\ y\ é - 45\text{”}$$

```
> teste := ‘O valor de x + y é ‘ || (x + y);
```

$$teste := O\ valor\ de\ x + y\ é\ 55$$

```
> teste;
```

$$O\ valor\ de\ x + y\ é\ 55$$

Exemplo 1.17 *A concatenação também pode ser realizada com um intervalo de valores da forma m..n. Neste caso, cada valor do intervalo será usado para formar um novo nome.*

```
> restart;
> x || (1..10);        # é o mesmo que cat(x, 1..10)
```

$$x1, x2, x3, x4, x5, x6, x7, x8, x9, x10$$

```
> "teste" || (1..5);  # é o mesmo que cat("teste", 1..5)
```

$$\text{“}teste1\text{”}, \text{“}teste2\text{”}, \text{“}teste3\text{”}, \text{“}teste4\text{”}, \text{“}teste5\text{”}$$

```
> x := -3:  x1 := 4:  x2 := b:
> y := x||1 + x||2;
```

$$y := 4 + b$$

1.17 Expressões previamente computadas

As três últimas expressões que foram efetivamente computadas podem ser referenciadas com os símbolos % (última), %% (penúltima) e %%% (antepenúltima). Essas expressões podem ser usadas na digitação de novos comandos.

Exemplo 1.18 *Calculamos o valor de* $\operatorname{sen}^2(\pi/20) + \cos^2(\pi/20)$. *Depois, somamos* 3 *ao resultado mostrado e calculamos sua raiz quadrada.*

```
> sin(Pi/20);
```

$$\sin\left(\frac{1}{20}\pi\right)$$

```
> cos(Pi/20);
```

$$\cos\left(\frac{1}{20}\pi\right)$$

```
> evalf((%)^2 + (%%)^2);
```

$$1.000000000$$

```
> z := % + 3;
```

$$z := 4.000000000$$

```
> sqrt(%);
```

$$2.000000000$$

Tenha muito cuidado ao usar esses símbolos. Eles se referem aos comandos que acabaram de ser executados e os comandos executados nem sempre seguem a mesma ordem na qual são mostrados. Ao carregar uma folha de trabalho elaborada em outro momento, podem ser executados somente alguns comandos dessa folha. Assim, o % pode não se referir ao último comando da folha, se ele não tiver sido executado.

1.18 Pacotes

Quando o Maple é iniciado, ele carrega o seu núcleo (*kernel*). O núcleo corresponde a aproximadamente 10% do programa e foi elaborado em linguagem C. É ele quem faz as operações aritméticas básicas, interpreta os comandos digitados e mostra resultados.

A parte do programa que não faz parte do núcleo, cerca de 90% do total, foi escrito na própria linguagem do Maple e consiste de duas partes: a biblioteca principal e um conjunto de vários pacotes (*packages*) separados. Assim como os comandos que fazem parte do núcleo, os comandos da biblioteca principal são carregados automaticamente na hora da inicialização e estão prontos para serem usados.

O Maple possui vários pacotes. Eles são agrupamentos de comandos com fins específicos. Tem pacote para construção de gráficos (como o `plots`), para Álgebra Linear (como o `LinearAlgebra` e o `linalg`), para a definição de objetos geométricos (como o `geometry`), etc. Uma relação de pacotes com breve descrição de cada um encontra-se no apêndice A.

Para que um comando de um pacote possa ser utilizado, precisamos fornecer o nome do pacote que o contém. Isso pode ser feito fornecendo-se o nome do pacote seguido do nome do comando entre colchetes: *pacote*[*comando*]. Uma expressão dessa forma é

chamada o *nome completo* do comando. Por exemplo, o nome completo do comando `Determinant` que faz parte do pacote `LinearAlgebra` é `LinearAlgebra[Determinant]` e o nome completo de `IsEquilateral` do pacote `geometry` é `geometry[IsEquilateral]`.

Uma maneira mais prática de se ter acesso aos comandos de um pacote é digitar, antes de usar o comando, um `with(pacote)`. Com isso, todos os comandos do pacote ficam disponíveis sem ser necessário escrevermos seus nomes completos.

Exemplo 1.19 *Considere as três seguintes atribuições feitas com alguns comandos do pacote* `LinearAlgebra`*:*

```
> d := LinearAlgebra[Determinant](M):
> N := LinearAlgebra[InverseMatrix](M):
> y := LinearAlgebra[CharacteristPolynomial](M, x):
```

Veja como é mais simples utilizá-los assim:

```
> with(LinearAlgebra):
> d := Determinant(M):
> N := InverseMatrix(M):
> y := CharacteristicPolynomial(M, x):
```

Se o `with(`*pacote*`)` for usado com um ponto e vírgula no final, então ele mostra uma relação com todos os nomes de comandos do pacote.

Para obter uma tela de ajuda com a descrição do pacote, basta digitar `?`*pacote*, por exemplo

```
> ?LinearAlgebra
```

e para obter informações sobre um comando de um pacote, basta digitar `?`*pacote*`[`*comando*`]` ou `?`*pacote*`,`*comando* ou, em muitos casos, simplesmente `?`*comando*.

1.19 Exercícios

1) Calcule de duas maneiras quantos algarismos possui a potência 9^{9^9} ao ser escrita por extenso no sistema decimal:

a) calcule a potência e conte os algarismos;

b) calcule a parte inteira do seu logaritmo decimal e adicione uma unidade.

2) Neste exercício são calculados duas vezes determinados valores. O objetivo é verificar que são obtidas as mesmas respostas, mesmo usando-se diferentes métodos.

a) Escreva por extenso o desenvolvimento de 100! e verifique diretamente quantos zeros aparecem no final desse desenvolvimento;

b) Fatore o resultado anterior escrevendo-o como produto de números primos e verifique quais são as potências de 2 e de 5 que aparecem nessa fatoração;

c) Sabendo que se p^α for uma potência de número primo p que aparece na fatoração de $n!$, então o maior valor possível para o expoente α é dado pela soma das partes inteiras de todos os possíveis valores de n/p^k com k inteiro positivo satisfazendo $p^k \leq n$, calcule dessa forma os maiores expoentes r e s das potências de 2 e 5 que aparecem na fatoração do item anterior;

d) Verifique que o menor entre r e s do item anterior coincide com a quantidade de zeros obtida no item (a).

3) Verifique que $\dfrac{355}{113}$, $\sqrt[4]{\dfrac{2143}{22}}$, $\sqrt[5]{\dfrac{77729}{254}}$ e $\ln\left(\dfrac{10691}{462}\right)$ são aproximações para π com pelo menos 6 casas decimais exatas.

4) Verifique que $\dfrac{\pi^4+\pi^5}{e^6}$ é próximo de 1.

5) Verifique que $|x-n|<10^{-12}$, onde $x=e^{\pi\sqrt{163}}$ e $n=262537412640768744$.

6) Verifique que $z=(5+i)^{16}(239-i)^4$ é um número real.

7) Verifique que $(\pi+20)^i$ é próximo de -1.

8) Dados três valores positivos a, b e c, verifique que

$$\left(\frac{a}{b}\right)^{\log c}\cdot\left(\frac{b}{c}\right)^{\log a}\cdot\left(\frac{c}{a}\right)^{\log b}=1.$$

9) A função `isprime(`n`)` retorna o valor *true* se n for um inteiro positivo primo e *false* em caso contrário. Por exemplo, `isprime`(7) $=$*true* e `isprime`(8) $=$*false*. Usando essa função, descubra alguns números da forma $2\times 10^k+3$ que sejam primos e que $k\geq 2$.

Capítulo 2

Expressões algébricas, equações e funções

Todo programa de Computação Algébrica pretende ser eficiente na parte de simplificação de expressões algébricas. Mas, essa é uma parte difícil de se obter a perfeição, até mesmo porque às vezes o conceito de simplicidade é algo subjetivo.

Neste capítulo introduzimos os comandos que simplificam, fatoram ou expandem uma expressão algébrica, mostramos como resolver equações e sistemas de equações e como definir funções.

2.1 Substituição

Em uma *expressão* algébrica nas variáveis $x, y, \ldots$ podemos substituir x por $expr1$, y por $expr2$, etc. se usarmos um comando do tipo

$$\texttt{subs}(x = expr1,\ y = expr2,\ \ldots,\ expressão)\,.$$

Exemplo 2.1 *Na expressão algébrica $E = x^2 + y^2 + z^2$, inicialmente substituímos x por -2.*

```
> restart;
> E := x^2 + y^2 + z^2;
```

$$E := x^2 + y^2 + z^2$$

```
> subs(x = -2, E);
```

$$4 + y^2 + z^2$$

```
> E;
```

$$x^2 + y^2 + z^2$$

Observe que a expressão E não foi alterada com a substituição efetuada, porque não foi feita uma nova atribuição de valor a E.

Agora, observe o que acontece quando substituímos x por a, y por b e atribuímos o resultado a E:

```
> E := subs(x = a, y = b, E);
```

$$E := a^2 + b^2 + z^2$$

```
> E;
```

$$a^2 + b^2 + z^2$$

Assim, a expressão E foi alterada com a troca de x por a e y por b.

Finalmente, substituímos a, b e z por valores numéricos.

```
> E := subs(a = -1, b = 3, z = -2, E);
```

$$E := 14$$

Exemplo 2.2 *Vamos fazer algumas substituições na expressão trigonométrica definida por* $y = 3 + \operatorname{tg}(a)^2 - 4\cos(2a)$.

```
> y := 3 + tan(a)^2 - 4*cos(2*a);
```

$$y := 3 + \tan(a)^2 - 4\cos(2a)$$

```
> subs(a = x + 1, y);
```

$$3 + \tan(x+1)^2 - 4\cos(2x+2)$$

```
> subs(tan(a) = sin(a)/cos(a), y);
```

$$3 + \frac{\sin(a)^2}{\cos(a)^2} - 4\cos(2a)$$

```
> subs(tan(a)^2 = sec(a)^2 - 1, y);
```

$$2 + \sec(a)^2 - 4\cos(2a)$$

```
> subs(cos(2*a) = 2*cos(a)^2 - 1, tan(a) = sin(a)/cos(a), y);
```

$$7 + \frac{\sin(a)^2}{\cos(a)^2} - 8\cos(a)^2$$

```
> subs(cos(2*a) = cos(a)^2 - sin(a)^2, y);
```

$$3 + \tan(a)^2 - 4\cos(a)^2 + 4\sin(a)^2$$

2.2 Simplificação

A simplificação é fundamental na apresentação de muitos resultados. Para isso, o Maple possui os comandos `simplify(`*expressão*`, `*opções*`)` e `combine(`*expressão*`, `*opções*`)` onde *opções* é opcional e pode ser usado para fazer suposição sobre as variáveis envolvidas (por exemplo, supor valores positivos).

Nem sempre é fácil definir o que seja uma "expressão mais simples". Um critério poderia ser: cancelar os fatores comuns ao numerador e denominador de uma fração. Assim, fica fácil decidir quem é o mais simples entre $\dfrac{a-b}{a^2-b^2}$ e $\dfrac{1}{a+b}$. Mas, note que já não é tão fácil escolher o mais simples entre $\dfrac{a-b}{a^{12}-b^{12}}$ e

$$\frac{1}{a^{11} + ba^{10} + b^2a^9 + b^3a^8 + b^4a^7 + b^5a^6 + b^6a^5 + b^7a^4 + b^8a^3 + b^9a^2 + b^{10}a + b^{11}}.$$

Exemplo 2.3 *Simplificar cada expressão* `Expr1`*,* `Expr2`*,* `Expr3`*,* `Expr4` *e* `Expr5`*.*

```
> Expr1 := (x^6  + 3*x^5  - 3*x^4  - 42*x^3  - 153*x^2  + 3*x + 11) /
>          (x^6  - 4*x^5  - 15*x^4  + 56*x^3  + 15*x^2  - 4*x - 1);
```

$$Expr1 := \frac{x^6+3x^5-3x^4-42x^3-153x^2+3x+11}{x^6-4x^5-15x^4+56x^3+15x^2-4x-1}$$

```
> simplify(Expr1);
```

$$\frac{x^2+3x+11}{x^2-4x-1}$$

```
> Expr2 := (1/x^2+1/y^2)/(1/x^2-1/y^2) + (1/x^2-1/y^2)/(1/x^2+1/y^2);
```

$$Expr2 := \frac{\frac{1}{x^2}+\frac{1}{y^2}}{\frac{1}{x^2}-\frac{1}{y^2}} + \frac{\frac{1}{x^2}-\frac{1}{y^2}}{\frac{1}{x^2}+\frac{1}{y^2}}$$

```
> simplify(Expr2);
```

$$4\frac{x^2y^2}{-y^4+x^4}$$

```
> Expr3 := ((p*x^2+(k-s)*x+r)^2 - (p*x^2+(k+s)*x+r)^2)/((p*x^2
>           + (k+t)*x+r)^2-(p*x^2+(k-t)*x+r)^2);
```

$$Expr3 := \frac{(px^2+(k-s)x+r)^2-(px^2+(k+s)x+r)^2}{(px^2+(k+t)x+r)^2-(px^2+(k-t)x+r)^2}$$

```
> simplify(Expr3);
```

$$-\frac{s}{t}$$

```
> Expr4 := a^3/((a-b)*(a-c)) + b^3/((b-c)*(b-a)) + c^3/((c-a)*(c-b));
```

$$Expr4 := \frac{a^3}{(a-b)(a-c)} + \frac{b^3}{(b-c)(b-a)} + \frac{c^3}{(c-a)(c-b)}$$

```
> Expr4 := simplify(Expr4);
```

$$Expr4 := b+a+c$$

```
> Expr5 := (1 - (1-(y+1)/(x+y+1)) / (1-x/(x+y+1))) / ((y+1)^2 -
>                       x / (1 + x/(y-x+1))*(x*(y+1)/(y-x+1) - x));
```

$$Expr5 := \frac{1-\frac{1-\frac{1+y}{x+y+1}}{1-\frac{x}{x+y+1}}}{(1+y)^2-\frac{x\left(\frac{x(1+y)}{y-x+1}-x\right)}{1+\frac{x}{y-x+1}}}$$

```
> Expr5 := simplify(Expr5);
```

$$Expr5 := \frac{1}{x^2+x+xy+1+2y+y^2}$$

Uma das opções do `simplify` pode ser um conjunto de restrições em forma de igualdades envolvendo polinômios de uma ou várias variáveis.

Exemplo 2.4 *Seja $z = a^3c^3d^4$. Queremos fazer em "z" a substituição de "ac" por "k".*

```
> z := a^3*c^3*d^4;
```

$$z := a^3c^3d^4$$

```
> subs(a*c = k, z);
```

$$a^3c^3d^4$$

O comando `subs` *não funciona porque a expressão "ac" não aparece explicitamente. Mas, a substituição pode ser feita com uma simplificação de "z" com a restrição "$ac = k$".*

```
> simplify(z, {a*c=k});
```

$$d^4k^3$$

Exemplo 2.5 *Simplificar $x^4 + y^4 + z^4$ com as restrições $xy = 2$ e $xz = 3$.*

```
> simplify(x^4 + y^4 + z^4, {x*y = 2, x*z = 3} );
```

$$x^4 + \frac{97}{81}z^4$$

Exemplo 2.6 *Calcular o valor de $x^3 + y^3 + z^3$ sabendo que $xyz = 27$, $x + y + z = 9$ e $xy + xz + yz = 27$.*

```
> restart;
> restricoes := {x*y*z = 27, x + y + z = 9, x*y + x*z + y*z = 27}:
> simplify(x^3 + y^3 + z^3, restricoes);
```

$$81$$

Só foi possível usar o `simplify` *neste caso porque as substituições são polinomiais (nas variáveis x, y e z).*

Exemplo 2.7 *O Maple não cancela automaticamente raiz quadrada com quadrados, conforme podemos observar a seguir. Se nesses casos o comando* `simplify` *for usado sem opções adicionais, ele pode não fazer nada com a expressão fornecida.*

```
> restart;
> y := sqrt(a^2 * ( b + 1)^2);
```

$$y := \sqrt{a^2(b+1)^2}$$

```
> simplify(y);
```

$$\sqrt{a^2(b+1)^2}$$

Agora, observe o que acontece quando supomos que todos os valores envolvidos são não-negativos.

```
> simplify(y, assume=nonnegative);
```

$$a(b+1)$$

O Maple passa a não se preocupar se os valores envolvidos são positivos ou negativos se for usado com a opção `symbolic`.

```
> simplify(y, symbolic);
```

$$a(b+1)$$

Exemplo 2.8 *Este exemplo é semelhante ao anterior: usamos o* `simplify` *com diversas opções para simplificar uma expressão que apareça raiz quadrada de um quadrado.*

```
> x := sqrt((alpha - 1)^2);
```

$$x := \sqrt{(\alpha - 1)^2}$$

Suponhamos inicialmente que os valores envolvidos sejam todos reais.

```
> simplify(x, assume=real);
```

$$|\alpha - 1|$$

Suponhamos agora valores no intervalo $[1, \infty[$*, ou seja,* $\alpha \geq 1$*. O intervalo* $[a, b]$ *de números reais pode ser definido como sendo* `RealRange(a, b)`*.*

```
> simplify(x, assume=RealRange(1, infinity));
```

$$\alpha - 1$$

Suponhamos agora valores no intervalo $]-\infty, 1]$*.*

```
> simplify(x, assume=RealRange(-infinity, 1));
```

$$1 - \alpha$$

Exemplo 2.9 *Usamos o comando* `combine` *para escrever um produto de funções trigonométricas como uma soma dessas funções.*

```
> y := cos(a)*cos(b):
> y := combine(y);
```

$$y := \frac{1}{2}\cos(a - b) + \frac{1}{2}\cos(a + b)$$

O `combine` *identifica ocorrências de* $\operatorname{sen}(nx)$ *ou* $\cos(nx)$ *em expressões.*

```
> z := 2*sin(x)*cos(x) + cos(x)^2 - sin(x)^2;
```

$$z := 2\sin(x)\cos(x) + \cos(x)^2 - \sin(x)^2$$

```
> combine(z);
```

$$\sin(2x) + \cos(2x)$$

Exemplo 2.10 *O* `combine` *detecta a ocorrência de senos ou cossenos de somas ou diferenças.*

```
> y := exp(sin(a)*sin(b))*exp(cos(a)*cos(b));
```

$$y := e^{(sin(a)sin(b))}e^{(cos(a)cos(b))}$$

```
> combine(y);
```

$$e^{(cos(a-b))}$$

Exemplo 2.11 *O* `combine` *leva em consideração os domínios das funções envolvidas. Observe que ele não simplifica de imediato a expressão L fornecida.*

```
> L := ln(x) + ln(y) - ln(z):
> combine(L);
```

$$\ln(x) + \ln(y) - \ln(z)$$

Se for usada a opção `symbolic`*, o* `combine` *passa a não se preocupar mais com os domínios das funções, efetuando a simplificação da expressão L sem maiores exigências.*

```
> combine(L, symbolic);
```

$$\ln\left(\frac{xy}{z}\right)$$

O comando `assume(`*propriedade*`)` pode ser usado para estabelecer relações ou propriedades de determinadas variáveis. Por exemplo, com um `assume(a > 0)` o Maple passa a tratar o a como sendo um valor positivo.

Depois de uma suposição a respeito de uma variável, o Maple passa a mostrá-la acompanhada de um til à direita do nome da variável.

Exemplo 2.12 *Inicialmente, tentamos simplificar uma soma de logaritmos sem fazer suposição sobre as variáveis envolvidas. Neste caso, o* `combine` *não faz nada com a expressão fornecida.*

```
> combine(5*log(u) + 3*log(v));
```

$$5\ln(u) + 3\ln(v)$$

Agora, suponhamos que u e v sejam positivos e apliquemos novamente o mesmo comando.

```
> assume(u > 0, v > 0);
> combine(5*log(u) + 3*log(v));
```

$$\ln(u \sim^5 v \sim^3)$$

Observe que os nomes das variáveis ficaram com um til à direita de cada um, mas o programa fez a simplificação esperada: combinou os logaritmos em um só.

Exemplo 2.13 *Vamos tentar combinar um produto de raízes quadradas em uma só raiz. O* `combine` *só funciona neste caso se forem dadas informações adicionais sobre os valores envolvidos. Para isso, usamos um* `assume`*.*

```
> combine(sqrt(m - 1)*sqrt(n - 2));
```

$$\sqrt{m-1}\sqrt{n-2}$$

```
> assume(m >= 1, n >= 2);
> combine(sqrt(m - 1)*sqrt(n - 2));
```

$$\sqrt{(m \sim -1)(n \sim -2)}$$

Os exemplos usados nesta seção mostram a eficiência do Maple na simplificação de muitos tipos de expressões. No entanto ele tem suas limitações. Por exemplo, o programa só simplifica $\dfrac{4^x - 1}{2^x - 1}$ depois de escrever as potências na base e, uma complicação desnecessária. Além disso, ele não consegue escrever $2x + 2y$ na forma $2(x + y)$, porque, para o programa, $2x+2y$ é o mais simples e ele não vai "complicar" algo que seja simples.

2.3 Expansão

Uma simplificação pode ocorrer não só no sentido de diminuir o tamanho de uma expressão. Em alguns casos é necessário desenvolver potências e efetuar produtos, e, com isso, o tamanho da expressão pode aumentar significativamente. O comando de uso geral para expansão é o `expand(`*expressão*`)`, cujo uso exemplificamos nos exemplos a seguir.

Exemplo 2.14 *Vamos expandir o produto* $(x-3)(x-4)$ *e desenvolver o binômio* $(a+b)^6$.

```
> expand((x-3)*(x-4));
```

$$x^2 - 7x + 12$$

```
> expand((a + b)^6);
```

$$a^6 + 6a^5b + 15a^4b^2 + 20a^3b^3 + 15a^2b^4 + 6ab^5 + b^6$$

Exemplo 2.15 *Se for usado em uma fração, o* `expand` *atua apenas no numerador, possivelmente escrevendo a fração como soma de várias frações.*

```
> expand((x^2 + x + 1)/(x^2 - 5*x + 6));
```

$$\frac{x^2}{x^2-5x+6} + \frac{x}{x^2-5x+6} + \frac{1}{x^2-5x+6}$$

Exemplo 2.16 *Com uma função trigonométrica* F, *o* `expand` *desenvolve expressões do tipo* $F(x+y)$, $F(x-y)$ *ou* $F(nx)$, *usando as conhecidas fórmulas do* $\cos(x+y)$, $\cos(2x)$, *etc.*

```
> expand(cos(x+y));
```

$$\cos(x)\cos(y) - \sin(x)\sin(y)$$

```
> expand(cos(2*x));
```

$$2\cos(x)^2 - 1$$

```
> expand(cos(3*x));
```

$$4\cos(x)^3 - 3\cos(x)$$

```
> expand(tan(5*x));
```

$$\frac{5\tan(x) - 10\tan(x)^3 + \tan(x)^5}{1 - 10\tan(x)^2 + 5\tan(x)^4}$$

Exemplo 2.17 *Às vezes é necessário usar vários comandos para simplificar uma única expressão. Neste exemplo usamos o* `simplify`, *o* `expand` *e o* `subs` *para obter que* $\dfrac{\cos(6x)+\cos(4x)}{\text{sen}\,(6x) - \text{sen}\,(4x)} = \text{cotg}(x)$.

```
> y := (cos(6*x) + cos(4*x))/(sin(6*x) - sin(4*x));
```

$$y := \frac{\cos(4x) + \cos(6x)}{-\sin(4x) + \sin(6x)}$$

```
> y := simplify(y);
```

$$y := \frac{\cos(2x) + 1}{\sin(2x)}$$

```
> y := expand(y);
```

$$y := \frac{\cos(x)}{\sin(x)}$$

```
> y := subs(cos(x)/sin(x) = cot(x), y);
```

$$y := \cot(x)$$

Se for usado um comando `expand(`*expressão1*`,` *expressão2*`)` então a *expressão1* é expandida totalmente, mantendo-se inalteradas as ocorrências da *expressão2*.

Exemplo 2.18 *Fornecemos uma expressão $p(x)$ e a desenvolvemos de várias maneiras: expansão completa, expansão mantendo-se inalteradas as potências de $x+2$ e expansão mantendo-se inalteradas as ocorrências de senos e de cossenos.*

```
> p(x) := (x + 2)^4 - 3*cos(x + 1)^2*(x + 2)^2 + 4*sin(x + 1)^2;
```

$$p(x) := (x+2)^4 - 3\cos(x+1)^2(x+2)^2 + 4\sin(x+1)^2$$

```
> expand(p(x)); # expansão completa
```

$$x^4+8x^3+24x^2+32x+16-3\cos(x)^2\cos(1)^2x^2-12\cos(x)^2\cos(1)^2x-12\cos(x)^2\cos(1)^2+6\cos(x)\cos(1)\sin(x)\sin(1)x^2+24\cos(x)\cos(1)\sin(x)\sin(1)x+32\cos(x)\cos(1)\sin(x)\sin(1)-3\sin(x)^2\sin(1)^2x^2-12\sin(x)^2\sin(1)^2x-12\sin(x)^2\sin(1)^2+4\sin(x)^2\cos(1)^2+4\cos(x)^2\sin(1)^2$$

```
> expand(p(x), x+2);
```

$$(x+2)^4-3(x+2)^2\cos(x)^2\cos(1)^2+6(x+2)^2\cos(x)\cos(1)\sin(x)\sin(1)-3(x+2)^2\sin(x)^2\sin(1)^2+4\sin(x)^2\cos(1)^2+8\cos(x)\cos(1)\sin(x)\sin(1)+4\cos(x)^2\sin(1)^2$$

```
> expand(p(x), cos, sin);
```

$$x^4+8x^3+24x^2+32x+16-3\cos(x+1)^2x^2-12\cos(x+1)^2x-12\cos(x+1)^2+4\sin(x+1)^2$$

Além do `expand`, outro comando que pode ser usado para desenvolver frações é o `normal(`*expressão*`,` *opções*`)`. Ele atua especialmente nos denominadores. Se for usado com a opção `expanded` então ele expande também as expressões que estiverem nos numeradores.

Exemplo 2.19 *Usamos o* `expand` *e o* `normal` *para expandir os produtos de polinômios que aparecem no numerador e no denominador de uma função racional dada.*

```
> y := ((x + 1)*(x - 2)^2)/((x^2 - 4*x + 7)*(x -6)^3);
```

$$y := \frac{(x+1)(x-2)^2}{(x^2-4x+7)(x-6)^3}$$

```
> expand(y);
```

$$\frac{x^3}{(x^2-4x+7)(x-6)^3} - \frac{3x^2}{(x^2-4x+7)(x-6)^3} + \frac{4}{(x^2-4x+7)(x-6)^3}$$

```
> normal(y, expanded);
```

$$\frac{x^3-3x^2+4}{x^5-22x^4+187x^3-774x^2+1620x-1512}$$

2.4 Fatoração

O comando `factor(`*expressão*`, `*opções*`)` pode ser usado para fatorar a *expressão* dada. Se não for fornecida nenhuma informação adicional através do parâmetro *opções*, o Maple entende que a fatoração desejada é para obter resultados com coeficientes inteiros.

Exemplo 2.20 *Fatorar* $x^4 - 16$ *e* $x^5 + x + 1$.

```
> x^4 - 16 = factor(x^4 - 16);
```

$$x^4 - 16 = (x-2)(x+2)(x^2+4)$$

```
> x^5 + x + 1 = factor(x^5 + x + 1);
```

$$x^5 + x + 1 = (x^2+x+1)(x^3-x^2+1)$$

Exemplo 2.21 *O* `factor` *atua no numerador e no denominador de uma função racional.*

```
> factor((x^2+x-2)/(x^2-5*x+6));
```

$$\frac{(x+2)(x-1)}{(x-2)(x-3)}$$

```
> q := (x^4+4*x^3+10*x^2+12*x+5)/(x^4-14*x^3+67*x^2-120*x+50);
```

$$q := \frac{x^4+4x^3+10x^2+12x+5}{x^4-14x^3+67x^2-120x+50}$$

```
> factor(q);
```

$$\frac{(x^2+2x+5)(x+1)^2}{(x^2-4x+2)(x-5)^2}$$

Para fatorar um polinômio em uma extensão $\mathbb{Q}[\alpha_1, \cdots, \alpha_n]$ dos racionais, basta fornecer $\alpha, \cdots, \alpha_n$ como parâmetros adicionais do comando `factor`. Isso faz com que a fatoração obtida possua fatores não só com coeficientes inteiros, mas também com coeficientes que sejam combinações de potências de $\alpha_1, \cdots, \alpha_n$.

Exemplo 2.22 *Inicialmente tentamos fatorar* $x^2 - 2$ *como produto de polinômios com coeficientes inteiros. Pela resposta dada, vemos que isso não é possível.*

```
> x^2 - 2 = factor(x^2 - 2);
```

$$x^2 - 2 = x^2 - 2$$

Agora, fatoramos $x^2 - 2$ *permitindo que a resposta contenha fatores com coeficientes envolvendo combinações de* $\sqrt{2}$.

```
> x^2 - 2 = factor(x^2 - 2, sqrt(2));
```

$$x^2 - 2 = (x+\sqrt{2})(x-\sqrt{2})$$

Exemplo 2.23 *Inicialmente, tentamos fatorar* $x^3 - 5$ *como produto de polinômios com coeficientes inteiros.*

```
> factor(x^3 - 5);
```

$$x^3 - 5$$

Se quisermos uma resposta na qual os coeficientes possam ser combinações de potências de $\sqrt[3]{5}$ *(ou seja, polinômios em* $\mathbb{Q}[\sqrt[3]{5}][x]$ *) então a fatoração é possível.*

```
> factor(x^3 - 5, root[3](5));
```

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

Obtivemos dessa forma que

$$x^3 - 5 \;=\; (x^2 + \sqrt[3]{5}x + \sqrt[3]{5^2})(x - \sqrt[3]{5})$$

A fatoração de polinômios também pode ser feita em corpos finitos. Para obter uma fatoração em $\mathbb{Z}_p$, p primo, basta usar um comando `Factor(`*expressão*`)` (com inicial maiúscula) seguido de um `mod p`.

Exemplo 2.24 *Inicialmente tentamos fatorar o seguinte $p(x)$ em $\mathbb{Z}[x]$. Vemos que a fatoração não é possível, ou seja, que ele é irredutível.*

```
> p(x) := x^7 + 3*x^4 - 9*x^3 + 12*x^2 - 33;
```

$$p(x) := x^7 + 3x^4 - 9x^3 + 12x^2 - 33$$

```
> factor(p(x));
```

$$x^7 + 3x^4 - 9x^3 + 12x^2 - 33$$

Agora, fatoramos $p(x)$ em $\mathbb{Z}_2[x]$.

```
> Factor(p(x)) mod 2;
```

$$(x+1)^5(x^2 + x + 1)$$

Fatoramos também $p(x)$ em $\mathbb{Z}_5[x]$.

```
> y := Factor(p(x)) mod 5;
```

$$y := (x+1)(x^2+2)(x^4 + 4x^3 + 4x^2 + 4x + 1)$$

Vamos conferir a última fatoração obtida. Inicialmente, vemos como fica $p(x)$ com coeficientes em $\mathbb{Z}_5$.

```
> p(x) mod 5;
```

$$x^7 + 3x^4 + x^3 + 2x^2 + 2$$

E agora expandimos y em $\mathbb{Z}_5$. Para isso basta acrescentar um `mod 5` *ao comando* `expand`.

```
> expand(y) mod 5;
```

$$x^7 + 3x^4 + x^3 + 2x^2 + 2$$

2.5 Equações e inequações

Uma equação é identificada pelo sinal de igualdade entre duas expressões.

O comando para resolução exata de uma equação é o `solve(`*equação*`)`. A resposta fornecida pelo `solve` é uma seqüência de raízes encontradas.

Para facilitar referências futuras, muitas vezes é conveniente atribuir uma equação a uma variável. Essa atribuição é um comando da forma

variável `:=` *expressão1* `=` *expressão2.*

O primeiro membro (lado esquerdo, *left hand side*) de uma *equação* pode ser referenciado com um comando `lhs(`*equação*`)` e o segundo membro (lado direito, *right hand side*) com um `rhs(`*equação*`)`. Por exemplo, sendo $E := x + 1 = 5 - x$, temos $\texttt{lhs}(E) = x + 1$ e $\texttt{rhs}(E) = 5 - x$.

Se não for fornecida uma *expressão2* como segundo membro da equação o Maple assume que ele é 0.

Exemplo 2.25 *Vamos resolver a equação do primeiro grau* $4x - 1 = 7x + 9$ *de duas maneiras: diretamente através do* `solve` *e passo a passo através de operações com as expressões de cada membro da equação.*

```
> eq := 4*x - 1 = 7*x + 9;
```

$$eq := 4x - 1 = 7x + 9$$

```
> solve(eq);
```

$$-\frac{10}{3}$$

Agora, vamos resolver a mesma equação passo a passo. Inicialmente, "passamos tudo para o primeiro membro" e igualamos a zero.

```
> eq := lhs(eq) - rhs(eq) = 0;
```

$$eq := -3x - 10 = 0$$

Somamos 10 *aos dois membros. Equivale a passar* -10 *do primeiro membro para o segundo membro com sinal trocado.*

```
> eq := lhs(eq) + 10 = rhs(eq) + 10;
```

$$eq := -3x = 10$$

Dividimos tudo por -3*. Equivale a passar o* -3 *para dividir o segundo membro. Com isso obtemos a raiz da equação.*

```
> eq := lhs(eq)/(-3) = rhs(eq)/(-3);
```

$$eq := x = -\frac{10}{3}$$

Exemplo 2.26 *O* `solve` *encontra soluções inteiras, irracionais ou complexas de muitos tipos de equações. Aqui, resolvemos três equações do segundo grau.*

```
> solve( x^2 + 4*x - 45 = 0 );  # é o mesmo que solve(x^2 + 4*x - 45)
```

$$5,\ -9$$

```
> solve( x^2 + 4*x + 2 = 0 );   # é o mesmo que solve(x^2 + 4*x + 2)
```

$$-2 + \sqrt{2},\ -2 - \sqrt{2}$$

```
> solve( x^2 + 4*x + 5 = 0 );   # é o mesmo que solve(x^2 + 4*x + 5)
```

$$-2 + I,\ -2 - I$$

Exemplo 2.27 *Se a equação possuir mais de uma variável, podemos escolher uma delas para obter seu valor. Neste exemplo fornecemos uma equação EQ com três variáveis* x, y *e* z *e isolamos o valor de cada variável. Para isso, basta especificar como segundo parâmetro do comando* `solve` *qual é a variável escolhida.*

```
> EQ := a*x + b*y + z = 3;
```

$$EQ := ax + by + z = 3$$

```
> x = solve(EQ, x);
```

$$x = -\frac{by + z - 3}{a}$$

```
> y = solve(EQ, y);
```

$$y = -\frac{ax + z - 3}{b}$$

```
> z = solve(EQ, z);
```

$$z = -ax - by + 3$$

Exemplo 2.28 *Resolver as equações exponenciais* $2^x = 16$ *e* $2^{2x} - 5 \cdot 2^x + 6 = 0$.

```
> solve( 2^x = 16);
```

$$\frac{\ln(16)}{\ln(2)}$$

```
> simplify(%);
```

$$4$$

```
> eq4 := 2^(2*x) - 5*2^x + 6 = 0;
```

$$eq4 := 2^{(2x)} - 5\ 2^x + 6 = 0$$

```
> solve(eq4);
```

$$\frac{\ln(3)}{\ln(2)},\ 1$$

Exemplo 2.29 *Resolver a equação logarítmica* $\log_3(5x+4) - \log_3(x) - \log_3(x-2) = 1$.

```
> eq5  := log[3](5*x + 4) - log[3](x) - log[3](x -2) = 1;
```

$$eq5 := \frac{\ln(5x+4)}{\ln(3)} - \frac{\ln(x)}{\ln(3)} - \frac{\ln(x-2)}{\ln(3)} = 1$$

```
> x = solve(eq5);
```

$$x = 4$$

Exemplo 2.30 *Resolver as equações irracionais* $\sqrt{x+6} + \sqrt{x+1} = \sqrt{7x+4}$ *e* $\sqrt{2x + \sqrt{1+6x^2}} = x + 1$.

```
> eq6 := sqrt(x + 6) + sqrt(x + 1) = sqrt(7*x + 4);
```

$$eq6 := \sqrt{x+6} + \sqrt{x+1} = \sqrt{7x+4}$$

```
> solve(eq6);
```

$$3$$

```
> eq7 := sqrt(2*x + sqrt(1 + 6*x^2)) = x + 1;
```

$$eq7 := \sqrt{2x + \sqrt{1+6x^2}} = x + 1$$

```
> solve(eq7);
```

$$0, \quad 2$$

A variável booleana _EnvAllSolutions controla a quantidade de raízes mostradas. Se ela for `false` (que é o padrão) são mostradas apenas algumas soluções. Se ela for `true` são mostradas todas as soluções. Neste caso, o Maple usa constantes genéricas cujos nomes sempre iniciam com um carácter sublinhado como _B1, _B2, _Z1, _Z2, ... As que tem a letra B são as que podem assumir valores 0 ou 1, as que tem a letra Z são as que podem assumir valores inteiros. Normalmente, essas constantes vêm acompanhadas de um til à direita para indicar que elas são restritas a determinados valores.

Exemplo 2.31 *Resolver a equação trigonométrica* $2\,\mathrm{sen}\,(3x) + \sqrt{2} = 0$.

```
> eq8 := 2*sin(3*x) + sqrt(2) = 0;
```

$$eq8 := 2\sin(3x) + \sqrt{2} = 0$$

```
> solve(eq8);
```

$$-\frac{1}{12}\pi$$

```
> _EnvAllSolutions := true:
> solve(eq8, x);
```

$$-\frac{1}{12}\pi + \frac{1}{2}\pi\_B1 \sim + \frac{2}{3}\pi\_Z1 \sim$$

Em um livro sobre trigonometria, uma resposta dessas seria apresentada na forma $-\frac{1}{12}\pi + \frac{1}{2}\pi + \frac{2}{3}k\pi$ *e* $-\frac{1}{12}\pi + \frac{2}{3}k\pi$, *com* $k \in \mathbb{Z}$.

Para obter informações sobre _B1 *e* _Z1 *podemos usar o comando* `about`*:*

```
> about(_B1);
```

Originally _B1, renamed _B1~: is assumed to be: OrProp(0,1)

```
> about(_Z1);
```

Originally _Z1, renamed _Z1~: is assumed to be: integer

Exemplo 2.32 *Resolver a equação trigonométrica* $\mathrm{sen}\,(2x) + \mathrm{sen}\,(4x) = 0$. *Inicialmente, vamos determinar só algumas soluções.*

```
> _EnvAllSolutions := false:
> eqtrig := sin(2*x) + sin(4*x) = 0:
> solve(eqtrig);
```

$$\frac{1}{2}\pi,\ 0,\ \frac{1}{3}\pi,\ -\frac{1}{3}\pi$$

Agora, vamos determinar todas as soluções.

```
> _EnvAllSolutions := true:
> solve(eqtrig);
```

$$\frac{1}{2}\pi + \pi\_Z1 \sim,\ \pi\_Z2 \sim,\ \frac{1}{3}\pi + \pi\_Z3 \sim,\ -\frac{1}{3}\pi + \pi\_Z3 \sim$$

Uma inequação pode ser resolvida de maneira semelhante a uma equação. Normalmente, a resposta é dada em forma de intervalo de $\mathbb{R}$. O intervalo fechado $[a, b]$ é representado por RealRange(a, b), enquanto que o intervalo aberto $]a, b[$ é representado por RealRange(Open(a), Open(b)).

Exemplo 2.33 *Resolver as inequações* $x^2 - 5x + 6 > 0$ *e* $|x + 5| \leq 4$.

```
> solve(x^2 - 5*x + 6 > 0);
```

$$RealRange(-\infty, Open(2)),\ RealRange(Open(3), \infty)$$

A solução assim encontrada é a união dos intervalos $]-\infty, 2[$ *e* $]3, \infty[$.

```
> ineq := abs(x + 5) <= 4;
```

$$ineq := |x + 5| \leq 4$$

```
> solve(ineq, x);
```

$$RealRange(-9, -1)$$

Ou seja, a solução é o intervalo $[-9, -1]$.

Exemplo 2.34 *Resolver a inequação* $\dfrac{x^2 - 10x + 9}{x^2 - 12x + 35} > 0$.

```
> ineq := (x^2 - 10*x + 9)/(x^2 - 12*x + 35) > 0;
```

$$ineq := 0 < \frac{x^2 - 10x + 9}{x^2 - 12x + 35}$$

```
> solve(ineq);
```

$$RealRange(-\infty, Open(1)),\ RealRange(Open(5), Open(7)),\ RealRange(Open(9), \infty)$$

Vemos assim que a solução da inequação dada é $]-\infty, 1[\cup]5, 7[\cup]9, \infty[$.

Quando o Maple não consegue resolver uma equação, ele costuma deixá-la indicada na forma `RootOf(`*equação*`,` *índice*`)`, como no seguinte exemplo.

Exemplo 2.35 *Tentamos resolver as equações* $x^6 + x - 1 = 0$ *e* $\cos(x) = x^2$ *com o comando* `solve`. *Como isso não é possível, o programa usa o* `RootOf` *nas respostas dadas.*

```
> solve(x^6 + x - 1 = 0);
```

$$RootOf(\_Z^6 + \_Z - 1, index = 1),\quad RootOf(\_Z^6 + \_Z - 1, index = 2),$$

$$RootOf(\_Z^6 + \_Z - 1, index = 3),\quad RootOf(\_Z^6 + \_Z - 1, index = 4),$$

$$RootOf(\_Z^6 + \_Z - 1, index = 5),\quad RootOf(\_Z^6 + \_Z - 1, index = 6)$$

```
> solve(cos(x) = x^2);
```

$$RootOf(-\cos(\_Z) + \_Z^2,\ label = \_L6)$$

2.6 Resolução numérica de equações

Podemos obter solução aproximada para uma equação com um comando `fsolve(`*equação*`,` *opções*`)`. Em *opções* pode aparecer o intervalo no qual a raiz da equação está sendo procurada, o valor da aproximação inicial da raiz ou o método de resolução a ser utilizado.

Exemplo 2.36 *Determinamos algumas raízes da equação polinomial* $x^5 - 3x^2 + 1 = 0$. *Inicialmente são encontradas todas as raízes reais.*

```
> eqpoli := x^5 - 3*x^2 + 1 = 0;
```

$$eqpoli := x^5 - 3x^2 + 1 = 0$$

```
> fsolve(eqpoli);
```

$$-.5610700072,\ .5992410280,\ 1.348046941$$

Agora, determinamos somente as raízes que se encontram no intervalo $[-2,\ 0]$.

```
> fsolve(eqpoli, x=-2..0);
```

$$-.5610700072$$

E agora, determinamos aquelas que se encontram em $[0,\ 2]$.

```
> fsolve(eqpoli, x=0..2);
```

$$.5992410280,\ 1.348046941$$

O padrão do Maple é mostrar 10 algarismos significativos. Vamos alterar essa quantidade para 30 algarismos significativos e observar como ficam as soluções positivas da equação dada.

```
> Digits := 30;
```

$$Digits := 30$$

```
> fsolve(eqpoli, x=0..2);
```

$$.599241027965685779228523068608,\ 1.348046941291338476851728104 44$$

Exemplo 2.37 *Resolver a equação* $\cos(x) - x^2 = 0$ *atribuindo inicialmente o valor* 1 *para aproximação inicial da raiz.*

```
> restart;
> eq := cos(x) - x^2 = 0;
```

$$eq := \cos(x) - x^2 = 0$$

```
> fsolve(eq, x=1);
```

$$.8241323123$$

Vamos resolver a mesma equação atribuindo o valor -2 *para aproximação inicial da raiz.*

```
> fsolve(eq, x=-2);
```

$$-.8241323123$$

Exemplo 2.38 *Determinados valores podem ser evitados na resolução da equação. A equação* $2^x = x^2$ *admite as soluções* 2 *e* 4*. Vamos determinar outra solução.*

```
> fsolve(2^x = x^2, x, avoid = {x=2, x=4});  # resolve evitando 2 e 4
```

$$-.7666646960$$

Assim, $x = -0,7666646960$ *é a outra solução real da equação dada.*

Exemplo 2.39 *O comando* `fsolve` *calcula também soluções complexas se for acrescentado um parâmetro* `complex`*. No caso de equações polinomiais ele encontra todas as raízes. Vamos resolver a equação* $x^7 + 8x^4 - x^3 + 101x - 9 = 0$.

```
> fsolve(x^7 + 8*x^4 - x^3 + 101*x - 9 = 0, x);
```

$$.08911092197$$

```
> fsolve(x^7 + 8*x^4 - x^3 + 101*x - 9 = 0, x, complex);
```

$-2.012925726 - .8034270247I,\ -2.012925726 + .8034270247I,\ .08911092197,$
$.2788769287 - 2.149930768I,\ .2788769287 + 2.149930768I,\ 1.689493337 - 1.311603271I,$
$1.689493337 + 1.311603271I$

2.7 Soluções inteiras para equações e congruências

Para determinar soluções inteiras de uma equação, o Maple possui o comando `isolve(`*equação*`)`. As equações que se buscam soluções inteiras são conhecidas pelo nome de *equações diofantinas*.

Exemplo 2.40 *Determinar todas as soluções inteiras de* $x^2 + y^2 = 1$.

```
> isolve(x^2 + y^2 = 1);
```

$$\{y = -1, x = 0\},\ \{x = -1, y = 0\},\ \{y = 1, x = 0\},\ \{x = 1, y = 0\}$$

Exemplo 2.41 *Determinar a solução geral das equações diofantinas* $2x + 5y = 9$ *e* $2x + 3y + 4z = 9$.

```
> isolve(2*x + 5*y = 9);
```

$$\{y = 1 + 2\_Z1 \sim,\ x = 2 - 5\_Z1 \sim\}$$

```
> isolve(2*x + 3*y + 4*z = 7);
```

$$\{x = 2 - 3\_Z1 - 2\_Z2,\ y = 1 + 2\_Z1,\ z = \_Z2\}$$

O Maple denota inteiros genéricos por $\_Z1,\ \_Z2,\ \ldots$

Uma congruência na variável x do tipo $f(x) \equiv b \pmod{p}$ pode ser resolvida com um comando `msolve(`*f(x) = b, p*`)`.

Exemplo 2.42 *Determinar* x *que satisfaça* $x^2 + x + 1 \equiv 0 \pmod{7}$.

```
> msolve(x^2 + x + 1 = 0, 7);
```

$$\{x = 2\},\ \{x = 4\}$$

Exemplo 2.43 *Resolver a congruência linear* $315x \equiv 12 \pmod{501}$.

```
> msolve(315*x = 12, 501);
```

$$\{x = 140\},\ \{x = 307\},\ \{x = 474\}$$

Qualquer outra solução é congruente módulo 501 *às soluções encontradas.*

Exemplo 2.44 *Resolver* $3^m \equiv 5 \pmod 7$

```
> msolve(3^m = 5, 7);
```

$$\{m = 5 + 6\_Z1 \sim\}$$

2.8 Sistemas de equações

Um sistema de equações pode ser resolvido de forma semelhante às equações, bastando escrever as equações do sistema em forma de conjunto. Se for necessário, podemos escrever também as variáveis em forma de conjunto e fornecê-lo ao comando de resolução como segundo parâmetro.

Exemplo 2.45 *Resolver o sistema linear*

$$\begin{cases} 3x + 5y &= 1 \\ 2x + 4y &= -9 \end{cases}$$

```
> solve ( { 3*x + 5*y = 1, 2*x + 4*y = -9} );
```

$$\{x = \frac{49}{2},\ y = -\frac{29}{2}\}$$

Exemplo 2.46 *Resolver o sistema*

$$\begin{cases} xy &= 16 \\ \log_2 x &= \log_2 y + 2 \end{cases}$$

```
> sis := {x*y = 16, log[2](x) = log[2](y) + 2}
```

$$sis := \left\{ \frac{\ln(x)}{\ln(2)} = \frac{\ln(y)}{\ln(2)} + 2,\ xy = 16 \right\}$$

```
> solve(sis);
```

$$\{x = 8,\ y = 2\}$$

Exemplo 2.47 *Resolver o sistema*

$$\begin{cases} x^2 + y^2 + z^2 &= 1 \\ xy + yz + zx &= 0 \\ x - y + 2z &= -1 \end{cases}$$

```
> restart;
> sist := {x^2 + y^2 + z^2 = 1, x*y + y*z + z*x = 0, x - y + 2*z = -1};
```

$$sist := \{x - y + 2z = -1,\ xy + yz + zx = 0,\ x^2 + y^2 + z^2 = 1\}$$

```
> var := {x, y, z};
```

$$var := \{x, y, z\}$$

```
> S := solve(sist, var);
```

$$S := \{y = 0, z = 0, x = -1\},\ \{z = 0, y = 1, x = 0\},\ \{x = \frac{2}{7}, z = -\frac{6}{7}, y = -\frac{3}{7}\},\ \{x = \frac{3}{7}, z = -\frac{2}{7}, y = \frac{6}{7}\}$$

Observe que a resposta é apresentada em forma de seqüência S de conjuntos, cada conjunto representando uma solução. Vamos conferir a resposta dada, substituindo cada solução S[i] no sistema dado.

```
> subs(S[1], sist);  # substitui a primeira solução
```

$$\{0 = 0, 1 = 1, -1 = -1\}$$

```
> subs(S[2], sist);  # substitui a segunda solução
```

$$\{0 = 0, 1 = 1, -1 = -1\}$$

```
> subs(S[3], sist);  # substitui a terceira solução
```

$$\{0 = 0, 1 = 1, -1 = -1\}$$

```
> subs(S[4], sist);  # substitui a quarta solução
```

$$\{0 = 0, 1 = 1, -1 = -1\}$$

Como obtivemos apenas sentenças verdadeiras como $0 = 0$, etc. temos que todas as soluções apresentadas estão corretas.

Para obter numericamente soluções de sistemas podemos usar o `fsolve`.

Exemplo 2.48 *Resolver o sistema não-linear*

$$\begin{cases} -13x^2 + xy + 3x + y^2 + 2z - 1 &= 0 \\ x^3 - 7xy^2 - xy - 3z^3 - 4xyz^2 - 101 &= 0 \\ x^4 - 2x^2 + 15x - zy^3 + 10z^2 + 2y - z - 4 &= 0 \end{cases}$$

e testar a solução obtida.

```
> restart;
> eq1 :=  -13*x^2 + x*y + 3*x + y^2 + 2*z - 1 = 0:
> eq2 :=   x^3 - 7*x*y^2 - x*y - 3*z^3 - 4*x*y*z^2 - 101 = 0:
> eq3 :=   x^4 - 2*x^2 + 15*x - z*y^3 + 10*z^2 + 2*y - z - 4 = 0:
> sist1 := {eq1, eq2, eq3};
```

$$sist1 := \{-13x^2 + xy + 3x + y^2 + 2z - 1 = 0,\ x^4 - 2x^2 + 15x - zy^3 + 10z^2 + 2y - z - 4 = 0, x^3 - 7xy^2 - xy - 3z^3 - 4xyz^2 - 101 = 0\}$$

```
> S := fsolve(sist1);
```

$$S := \{z = -3.594600364,\ x = -.3954409383,\ y = -3.185690959\}$$

```
> evalf(subs(S, sist1));
```

$$\{0. = 0.,\ .5\ 10^{-8} = 0.,\ -.54\ 10^{-7} = 0.\}$$

Pelo resultado obtido na substituição, temos que a solução obtida é realmente uma solução aproximada.

Exemplo 2.49 *Resolver o sistema não-linear*

$$\begin{cases} \cos x + \operatorname{sen} y \, \ln x &= 1 \\ x^3 - e^{-\operatorname{sen} y} &= xy \end{cases}$$

Obter uma resposta com 15 algarismos significativos e usar como aproximação inicial das raízes os valores $x = 0$ *e* $y = 0$.

```
> restart;
> Digits := 15:
> eq1 := cos(x) + sin(y)*ln(x) = 1;
```

$$eq1 := \cos(x) + \sin(y)\ln(x) = 1$$

```
> eq2 := x^3 - exp(-sin(y)) = x*y;
```

$$eq2 := x^3 - e^{(-\sin(y))} = xy$$

```
> sis := {eq1, eq2}:
> ini := {x = 0, y = 0}:
> sol := fsolve(sis, ini);
```

$$sol := \{x = .710694680392728,\ y = -2.35368429627066\}$$

Substituindo a solução obtida no sistema dado:

```
> subs(sol, sis);
```

$$\{\cos(.710694680392729) + \sin(-2.35368429627067)\ln(.710694680392729) = 1,$$
$$.358962593390370 - e^{(-\sin(-2.35368429627067))} = -1.67275090868347\}$$

```
> evalf(%);
```

$$\{-1.67275090868346 = -1.67275090868347,\ .999999999999999 = 1.\}$$

Observe que obtivemos igualdades aproximadas, logo a solução obtida é uma solução aproximada.

Resolver o mesmo sistema usando como aproximação inicial os valores $x = 1$ *e* $y = -10$.

```
> fsolve(sis, {x = 1, y = -10});
```

$$\{x = .0796290718233735,\ y = -12.5676228869274\}$$

Observe que é encontrada dessa forma uma outra solução para o sistema.

2.9 Funções

Existem duas maneiras de definir uma função $f(x)$ no Maple:

- com o "*operador seta*":

  `f := x ->` *expressão na variável* `x`

- com um comando `unapply`:

  `f := unapply(`*expressão*`, x)`

Exemplo 2.50 *Definir a função* $f(x) = x^2$ *e calcular* $f(-3)$.

```
> f := x -> x^2;
```

$$f := x \to x^2$$

```
> f(-3);
```

$$9$$

Se f *for usada como valor de uma atribuição a uma variável* g*, então a variável fica sendo uma função igual à função* f*:*

```
> g := f;
```

$$g := f$$

```
> g(-3);
```

$$9$$

Exemplo 2.51 *Aqui mostramos como uma função* **não** *deve ser definida. Vamos tentar definir a função* $h(x) = x^2 + 1$ *da seguinte forma:*

```
> h(x) := x^2 + 1;
```

$$h(x) := x^2 + 1$$

Observe que isso não funciona. Por exemplo, o Maple não obtém a resposta 10 *para* $h(-3)$*:*

```
> h(-3);
```

$$h(-3)$$

Também não obtém a resposta 5 *para* $h(2)$*:*

```
> h(2);
```

$$h(2)$$

Assim, não foi definida uma função h *como estava sendo desejado. O que se fez foi definir uma expressão algébrica de nome* $h(x)$ *e atribuir o valor* `x^2 + 1` *a essa expressão algébrica.*

Também não conseguimos definir essa mesma função na seguinte forma:

```
> y := x^2 + 1;
```

$$y := x^2 + 1$$

```
> y(2);
```

$$x(2)^2 + 1$$

Note que se y *fosse função de* x *o valor de* $y(2)$ *deveria ser* 5*. Neste caso,* y *é apenas o nome da expressão algébrica* `x^2 + 1`*.*

Exemplo 2.52 *Definimos uma função* $f(t)$ *que assume como valor a lista* $[t^2,\ t^3 - t,\ 2t + 3]$.

```
f := t -> [t^2, t^3 - t, 2*t + 3];
```

$$f := t \to [t^2, t^3 - t, 2t + 3]$$

```
> f(1);
```

$$[1, 0, 5]$$

```
> f(-2);
```

$$[4, -6, -1]$$

Exemplo 2.53 *Usamos o* `unapply` *para definir uma função* $f(x) = x^2 + 1$ *a partir de uma expressão algébrica.*

```
> f := unapply(x^2 + 1, x);
```

$$f := x \to x^2 + 1$$

```
> f(-3);
```

$$10$$

Agora vamos usar o `unapply` *para definir outra função* $h(t)$ *obtida a partir de uma expressão algébrica na variável* t*.*

```
> expressao := 4*t^2 + 5*t - 1;
```

$$expressao := 4t^2 + 5t - 1$$

```
> h := unapply(expressao, t);
```

$$h := t \to 4t^2 + 5t - 1$$

```
> h(-1);
```

$$-2$$

```
> h(2);
```

$$25$$

Funções de várias variáveis também são definidas com o operador seta ou com o comando `unapply`.

Exemplo 2.54 *Neste exemplo definimos as funções* $F(x, y) = (x^2 - y)(4x^2 - y)$ *e* $G(u, v, w) = u^2 + v^2 + w^2$ *e testamos as funções recém-definidas calculando valores em dois pontos.*

```
> F := (x, y) -> (x^2 - y)*(4*x^2 - y);
```

$$F := (x, y) \to (x^2 - y)(4x^2 - y)$$

```
> G := (u, v, w) -> u^2 + v^2 + w^2;
```

$$G := (u, v, w) \to u^2 + v^2 + w^2$$

```
> F(2, 2);
```

$$28$$

```
> G(1, 2, -1);
```

$$6$$

Exemplo 2.55 *Usamos o* `unapply` *para definir a função* $g(x, y) = \cos(xy)$*.*

```
g := unapply(cos(x*y), x, y);
```

$$g := (x, y) \to \cos(xy)$$

```
> g(Pi/8, -2);
```

$$\frac{1}{2}\sqrt{2}$$

E agora usamos o `unapply` *para definir a função* $fH(\alpha, \beta) = \cos(\alpha) + \cos(\beta)$ *a partir de uma expressão algébrica* H*.*

```
> H := cos(alpha) + cos(beta);
```

$$H := \cos(\alpha) + \cos(\beta)$$

```
> fH := unapply(H, alpha, beta);
```

$$fH := (\alpha, \beta) \to \cos(\alpha) + \cos(\beta)$$

```
> fH(Pi/6, -Pi/4);
```

$$\frac{1}{2}\sqrt{3} + \frac{1}{2}\sqrt{2}$$

2.9.1 Funções definidas por várias sentenças

Funções definidas por várias sentenças podem ser definidas com um comando `piecewise` que fornece uma expressão algébrica

`piecewise(`*cond1, f1, cond2, f2,* $\cdots$*, condN, fN, f_outros*`)`

onde *f1, f2, fN, f_outros* são expressões algébricas e *cond1, cond2, condN* são expressões lógicas. Se *cond1* for satisfeita será usado o valor da expressão *f1*, se *cond2* for satisfeita será usado o valor de *f2* e assim por diante. Se nenhuma condição *cond1, cond2,* $\cdots$ *condN* for verificada e uma expressão opcional *f_outros* tiver sido definida, será usado o valor dessa expressão.

Exemplo 2.56 *A função* $f(x) = \begin{cases} x^2 + x + 1 & \text{se} \quad x < 3 \\ 2x - 7 & \text{se} \quad x \geq 3 \end{cases}$ *pode ser definida por*

```
> f := x -> piecewise(x < 3, x^2 + x + 1, x >= 3, 2*x - 7);
```

$$f := x \to \text{piecewise}(x < 3,\ x^2 + x + 1,\ 3 \leq x,\ 2x - 7)$$

```
> f(2);   # testando a definição de f
```

$$7$$

```
> f(10);  # testando a definição de f
```

$$13$$

Exemplo 2.57 *A função*

$$f(x) = \begin{cases} -7, & \text{se} \quad 1 \leq x \leq 3 \\ \cos(x), & \text{se} \quad x = 0 \text{ ou } x > 5 \\ x^3, & \text{nos demais casos} \end{cases}$$

pode ser definida por

```
>  f := x -> piecewise(x >= 1 and x <= 3, -7, x=0 or x>5, cos(x), x^3);
```

$$f := x \to \text{piecewise}(1 \le x \textbf{ and } x \le 3, -7, x = 0 \textbf{ or } 5 < x, \cos(x), x^3)$$

```
> f(1);    # testando a definição de f
```

$$-7$$

```
> f(4);    # testando a definição de f
```

$$64$$

```
> f(9*Pi); # testando a definição de f
```

$$-1$$

2.9.2 Composição de funções

A composição de funções é feita com o operador @. Por exemplo, $f@g$ é a composta $f \circ g$. A composta de f consigo mesma, n vezes, pode ser abreviada por $f@@n$.

A função inversa de f é a função $f@@(-1)$, mas o Maple só consegue calculá-la em poucos casos ($\ln(x)$, $\exp(x)$ ou função trigonométrica ou hiperbólica básica).

Exemplo 2.58 *Dadas as funções $f(x) = 4x + 1$ e $g(x) = x^2 - x$, calcular $f \circ g$, $g \circ f$, $f \circ f \circ f \circ f$ e $f \circ f \circ g \circ g \circ g$.*

```
> f := x -> 4*x + 1;
```

$$f := x \to 4x + 1$$

```
> g := x -> x^2 - x;
```

$$g := x \to x^2 - x$$

```
> (f@g)(x);
```

$$4x^2 - 4x + 1$$

```
> (g@f)(x);
```

$$(4x + 1)^2 - 4x - 1$$

```
> (f@f@f@f)(x);
```

$$256x + 85$$

```
> (f@@4)(x);
```

$$256x + 85$$

```
> (f@f@g@g@g)(x);   # é o mesmo que ((f@@2)@(g@@3))(x)
```

$$16((x^2 - x)^2 - x^2 + x)^2 - 16(x^2 - x)^2 + 16x^2 - 16x + 5$$

Exemplo 2.59 *Calcular a composta $\underbrace{f \circ f \circ \cdots \circ f}_{10 \ \ vezes}$ de $f(x) = \sqrt{1 + x}$ consigo mesma.*

```
> f := x -> sqrt(1 + x);
```

$$f := x \to \sqrt{1 + x}$$

```
> (f@@10)(x);
```

$$\sqrt{1 + \sqrt{1 + \sqrt{1 + \sqrt{1 + \sqrt{1 + \sqrt{1 + \sqrt{1 + \sqrt{1 + \sqrt{1 + x}}}}}}}}}$$

2.9.3 O comando `map`

O comando `map(`f`, `A`)` permite que uma função f seja aplicada a cada elemento de um conjunto A (ou lista).

Exemplo 2.60 *Aplicar $f(x) = x^2$ aos elementos de $A = \{a, b, c, d\}$.*

```
> f := x -> x^2;
```

$$f := x \to x^2$$

```
> A := {a, b, c, d};
```

$$A := \{a, b, c, d\}$$

```
> map(f, A);    # calcula f(A)
```

$$\{a^2, b^2, c^2, d^2\}$$

```
> map(g, A);
```

$$\{g(a), g(b), g(c), g(d)\}$$

A função g não foi definida, mas o `map` *a aplica de forma simbólica.*

Exemplo 2.61 *Aplicar a função cosseno e a função $x \to x/\pi$ a uma lista L dada.*

```
> L := [0, Pi/8, Pi/7, Pi/6, Pi/5, Pi/4, Pi/3, Pi/2];
```

$$L := \left[0, \frac{1}{8}\pi, \frac{1}{7}\pi, \frac{1}{6}\pi, \frac{1}{5}\pi, \frac{1}{4}\pi, \frac{1}{3}\pi, \frac{1}{2}\pi\right]$$

```
> map(cos, L);
```

$$\left[1, \cos\left(\frac{1}{8}\pi\right), \cos\left(\frac{1}{7}\pi\right), \frac{1}{2}\sqrt{3}, \cos\left(\frac{1}{5}\pi\right), \frac{1}{2}\sqrt{2}, \frac{1}{2}, 0\right]$$

```
> map( x->x/Pi, L);
```

$$\left[0, \frac{1}{8}, \frac{1}{7}, \frac{1}{6}, \frac{1}{5}, \frac{1}{4}, \frac{1}{3}, \frac{1}{2}\right]$$

A função $x \to x/\pi$ foi utilizada sem um nome; neste caso dizemos que ela é uma "função anônima".

Se a função F a ser aplicada pelo `map` possuir outros parâmetros ou opções, então esses parâmetros também podem ser passados pelo `map`.

Exemplo 2.62 *Calculamos F em um conjunto L com opções $op1$ e $op2$.*

```
> L := {1, 2, 3, 4, 5}:
> map(F, L, op1, op2);
```

$$\{F(1, op1, op2),\ F(2, op1, op2),\ F(3, op1, op2),\ F(4, op1, op2),\ F(5, op1, op2)\}$$

2.10 Polinômios

O Maple possui alguns comandos para usar com polinômios definidos por expressões algébricas:

degree(p, x) Grau do polinômio $p(x)$

coeffs(p, x) Seqüência de coeficientes não-nulos de $p(x)$

coeff(p, x, n) n-ésimo coeficiente de $p(x)$

sort(p) Ordena os termos segundo a ordem decrescente das potências

collect(p, x) Soma os termos semelhantes de $p(x)$

quo(f, g, x) Quociente da divisão de $f(x)$ por $g(x)$

rem(f, g, x) Resto da divisão de $f(x)$ por $g(x)$

gcd(f, g, x) Máximo divisor comum entre $f(x)$ e $g(x)$

lcm(f, g, x) Mínimo múltiplo comum entre $f(x)$ e $g(x)$

gcdex(f, g, x, 'a', 'b') Máximo divisor comum "estendido": além de $MDC(f(x), g(x)) = d(x)$, calcula $a(x)$ e $b(x)$ tais que $a(x)f(x) + b(x)g(x) = d(x)$

Exemplo 2.63 *Determine o grau e os coeficientes de* $p(x) = x^5 + 4x^3 - 10x^2 + 7$.

```
> p(x) := x^5 + 4*x^3 - 10*x^2 + 7:
> degree(p(x), x);   # Grau de p(x)
```

$$5$$

```
> coeff(p(x), x, 2); # Segundo coeficiente não-nulo
```

$$-10$$

```
> coeffs(p(x), x);   # Coeficientes
```

$$7, 1, 4, -10$$

```
> op(p(x));          # Seqüência de termos de p(x)
```

$$x^5, 4x^3, -10x^2, 7$$

Exemplo 2.64 *Somar os termos semelhantes de* $ax^5 + bx^3 + x^2 - x + bx^5 + 3x^3$.

```
> f := a*x^5 + b*x^3 + x^2 - x + 4 + b*x^5 - 4 + 3*x^3;
```

$$f := ax^5 + bx^3 + x^2 - x + bx^5 + 3x^3$$

```
> f := collect(f, x);
```

$$f := (a + b)x^5 + (b + 3)x^3 + x^2 - x$$

Exemplo 2.65 *Calcular o quociente e o resto da divisão do polinômio* f *pelo polinômio* g *e o MDC entre eles.*

```
> f := x^4 + 2*x^3 - 5*x + 10:
> g := x^2 + x + 3:
> q := quo(f, g, x);  # quociente
```

$$q := x^2 + x - 4$$

```
> r := rem(f, g, x);  # resto
```

$$r := 22 - 4x$$

Testando o quociente e o resto obtidos:

```
> g*q + r;              # deve ser igual a f
```

$$(x^2 + x + 3)(x^2 + x - 4) + 22 - 4x$$

```
> simplify(%);
```

$$x^4 + 2x^3 - 5x + 10$$

```
> gcd(f, g);            # MDC
```

$$1$$

Exemplo 2.66 *Dados dois polinômios f e g calcular $d := MDC(f(x), g(x))$ e polinômios a e b tais que $a(x)f(x) + b(x)g(x) = d(x)$.*

```
> f := x^2 - 5*x + 6:
> g := x^3 - 6*x^2 + 12*x - 8:
> d := gcdex(f, g, x, 'a', 'b');
```

$$d := x - 2$$

```
> a;
```

$$1 - x$$

```
> b;
```

$$1$$

```
> a*f + b*g;
```

$$(1 - x)(x^2 - 5x + 6) + x^3 - 6x^2 + 12x - 8$$

```
> simplify(%);
```

$$x - 2$$

Exemplo 2.67 *Escrever um polinômio p de grau 2 na forma de quadrado de uma soma mais uma constante (geralmente, esse procedimento chama-se "completar o quadrado"). Igualamos p a um polinômio da forma $a(x+b)^2+c$ e resolvemos o sistema formado pelas equações obtidas ao se igualar todos os coeficientes a 0.*

```
> restart;
> p := 5*x^2 + 3*x + 7:
> q := a*(x + b)^2 + c;
> eq := p - q;
```

$$eq := 5x^2 + 3x + 7 - a(x + b)^2 - c$$

```
> eq := collect(eq, x);
```

$$eq := (5 - a)x^2 + (-2ab + 3)x + 7 - ab^2 - c$$

```
> sis := {coeffs(eq, x)};
```

$$sis := 5 - a, -2ab + 3, 7 - ab^2 - c$$

```
> S := solve(sis);
```

$$S := \{b = \frac{3}{10},\ a = 5,\ c = \frac{131}{20}\}$$

```
> q := subs(S, q);
```

$$q := 5(x + \frac{3}{10})^2 + \frac{131}{20}$$

Conferindo se o q assim obtido é igual a p:

```
> expand(q);
```

$$5x^2 + 3x + 7$$

Exemplo 2.68 *Dado um polinômio qualquer, escrevê-lo ordenado pela ordem decrescente das potências.*

```
> p := x^4 + 3*x + 2*x^2 + 4 - 7*x^5;
```

$$p := x^4 + 3x + 2x^2 + 4 - 7x^5$$

```
> p := sort(p);
```

$$p := -7x^5 + x^4 + 2x^2 + 3x + 4$$

2.11 Outros comandos de simplificação

2.11.1 Racionalização de denominadores

O comando `rationalize(`*expressão*`)` racionaliza uma expressão com radicais no denominador.

Exemplo 2.69 *Racionalizar o denominador de* $\frac{2-\sqrt{5}}{3+\sqrt{5}}$.

```
> x := (2 - sqrt(5))/(3 + sqrt(5));
```

$$x := \frac{2-\sqrt{5}}{3+\sqrt{5}}$$

```
> x: = rationalize(x);
```

$$x := \frac{1}{4}(-2+\sqrt{5})(-3+\sqrt{5})$$

```
> x := expand(x);
```

$$x := \frac{11}{4} - \frac{5}{4}\sqrt{5}$$

2.11.2 Convertendo expressões

O comando `convert(`*expressão*`, `*tipo*`)` converte a expressão fornecida como primeiro parâmetro para uma expressão equivalente do tipo indicado como segundo parâmetro. Em alguns casos pode simplificar a expressão dada.

Exemplo 2.70 *Converter as funções hiperbólicas* senh, arcsenh, tgh *e* arctgh *em expressões que contenham logaritmos naturais ou exponenciais.*

```
> convert(sinh(x), exp);
```

$$\frac{1}{2}e^x - \frac{1}{2}\frac{1}{e^x}$$

```
> convert(arcsinh(x), ln);
```

$$\ln(x + \sqrt{x^2+1})$$

```
> convert(arctanh(x), ln);
```

$$\frac{1}{2}\ln(x+1) - \frac{1}{2}\ln(1-x)$$

```
> convert(tanh(x), exp);
```

$$\frac{(e^x)^2 - 1}{(e^x)^2 + 1}$$

Exemplo 2.71 *Uma lista pode ser convertida em um conjunto, ou vice-versa.*

```
> L := [1, 2, 3, 4];
```

$$L := [1, 2, 3, 4]$$

```
> convert(L, set);
```

$$\{1, 2, 3, 4\}$$

2.11.3 Os comandos `numer` e `denom`

Os comandos `numer(`*expressão*`)` e `denom(`*expressão*`)` calculam o numerador e o denominador de uma expressão dada.

Exemplo 2.72 *Dada uma função racional y na variável x, calcular seu numerador e seu denominador.*

```
> y := (3*x^2 + 6*x + 2)/(x*(x + 1)*(x + 2));
```

$$y := \frac{3x^2 + 6x + 2}{x(x+1)(x+2)}$$

```
> denom(y);
```

$$x(x+1)(x+2)$$

```
> numer(y);
```

$$3x^2 + 6x + 2$$

Substituir x por w somente no denominador da expressão y.

```
> y := numer(y)/subs(x = w, denom(y));
```

$$y := \frac{3x^2 + 6x + 2}{w(w+1)(w+2)}$$

Exemplo 2.73 *A partir de uma lista dada, construir duas outras listas: uma com os numeradores e outra com os denominadores da lista dada. Para isso, basta usar o comando* `map` *combinado com o* `numer` *e o* `denom` *aplicado à lista.*

```
> lista := [3/4, a/b, -1/x, 1/2, 4, m, n, -3/5];
```

$$lista := [\frac{3}{4}, \frac{a}{b}, -\frac{1}{x}, \frac{1}{2}, 4, m, n, -\frac{3}{5}]$$

```
> Numeradores := map(numer, lista);
```

$$Numeradores := [3, a, -1, 1, 4, m, n, -3]$$

```
> Denominadores := map(denom, lista);
```

$$Denominadores := [4, b, x, 2, 1, 1, 1, 5]$$

2.12 Exercícios

1) Se for possível, escreva $p(x) = x^{10} + x^n + 1$ como sendo um produto de dois polinômios não constantes de coeficientes inteiros, com n inteiro variando de 1 a 9.

2) Considere a equação $4^x + 6^x = 9^x$.
a) Obtenha uma solução aproximada com o comando `fsolve`;
b) Tente resolvê-la com um comando `solve`. Neste caso, só será possível resolvê-la depois de uma divisão por 6^x seguida de uma simplificação;
c) Entre as respostas obtidas no item (b), usando um comando `select` determine as soluções reais da equação.

3) Verifique que se $a + b + c = 0$, então:
a) $a^5(b^2 + c^2) + b^5(a^2 + c^2) + c^5(a^2 + b^2) = \dfrac{(a^3 + b^3 + c^3)(a^4 + b^4 + c^4)}{2}$;
b) $\dfrac{a^7 + b^7 + c^7}{7} = \dfrac{a^5 + b^5 + c^5}{5}\,\dfrac{a^2 + b^2 + c^2}{2}$.

4) Simplifique as seguintes expressões:
a) $3^{\sqrt{\log_3 2}} - 2^{\sqrt{\log_2 3}}$;
b) $a^2\dfrac{(d-b)(d-c)}{(a-b)(a-c)} + b^2\dfrac{(d-c)(d-a)}{(b-c)(b-a)} + c^2\dfrac{(d-a)(d-b)}{(c-a)(c-b)}$;
c) $\dfrac{\sqrt{\sqrt[4]{8} + \sqrt{\sqrt{2} - 1}} - \sqrt{\sqrt[4]{8} - \sqrt{\sqrt{2} - 1}}}{\sqrt{\sqrt[4]{8} - \sqrt{\sqrt{2} + 1}}}$ (Sugestão: calcule o quadrado e expanda)

5) Simplifique e fatore:

$$\frac{(a^2 - b^2)^3 + (b^2 - c^2)^3 + (c^2 - a^2)^3}{(a-b)^3 + (b-c)^3 + (c-a)^3}.$$

6) Mostre que

$$\sqrt[3]{2 + \frac{10}{9}\sqrt{3}} + \sqrt[3]{2 - \frac{10}{9}\sqrt{3}}$$

é um número inteiro.

7) Simplifique:
a) $\cos(\frac{2\pi}{7}) + \cos(\frac{4\pi}{7}) + \cos(\frac{6\pi}{7})$;
b) $\operatorname{sen}^4(\frac{\pi}{16}) + \operatorname{sen}^4(\frac{3\pi}{16}) + \operatorname{sen}^4(\frac{5\pi}{16}) + \operatorname{sen}^4(\frac{7\pi}{16})$.

8) Resolva as equações:
a) $\sqrt[4]{x-1} + 2\sqrt[3]{3x+2} = 4 + \sqrt{3-x}$;
b) $\sqrt[5]{(x-2)(x-32)} - \sqrt[4]{(x-1)(x-33)} = 1$;
c) $\sqrt[4]{78 + \sqrt[3]{24+\sqrt{x}}} - \sqrt[4]{84 - \sqrt[3]{30-\sqrt{x}}} = 0$.

9) Resolva as equações:
a) $x^5 + x^4 - 6x^3 - 14x^2 - 11x - 3 = 0$;
b) $x^6 - 18x^5 + 147x^4 - 684x^3 + 1911x^2 - 3042x + 2197 = 0$.

10) Resolva a equação

$$\operatorname{sen}^{10}(x) + \cos^{10}(x) = \frac{29}{16}\cos^4(2x)$$

de duas maneiras:
a) diretamente, com o comando `solve`;
b) substituindo $\cos^{10}(x)$ por $\left(\frac{1+\cos(2x)}{2}\right)^5$, $\operatorname{sen}^{10}(x)$ por $\left(\frac{1-\cos(2x)}{2}\right)^5$, simplificando e substituindo $\cos(2x)$ por y. Depois, resolvendo a equação polinomial na variável y e cada equação do tipo $\cos(2x) = y$.

11) Determine todas as raízes complexas da equação

$$x^8 - 13x^7 + x^6 - 10x^4 + 2x^2 - 5x + 25 = 0.$$

Obtenha resultados com 15 algarismos significativos.

12) A partir de $x = 1, y = 1, z = 1$ obtenha uma solução do sistema

$$\begin{cases} x + \ln(y^2 + z^3) &= 3 \\ \cos(x) + y - e^z &= 1 \\ xy + \operatorname{sen}(x+z) &= -2 \end{cases}$$

com 20 algarismos significativos. Teste a solução encontrada substituindo-a em cada equação do sistema.

13) Calcule e simplifique $\sqrt[3]{-8}$ e $-\sqrt[3]{8}$. Tente explicar a diferença entre os resultados obtidos ao usar os comandos `root` e `surd`.

14) Sabendo que

$$x = \sqrt[3]{-\frac{q}{2} + \sqrt{\frac{q^2}{4} + \frac{p^3}{27}}} + \sqrt[3]{-\frac{q}{2} - \sqrt{\frac{q^2}{4} + \frac{p^3}{27}}}.$$

é solução da equação do terceiro grau $x^3 + px + q = 0$, resolva a equação $x^3 + 2x - 5 = 0$ de duas maneiras: usando a fórmula acima e usando o comando `fsolve`. Analise os resultados obtidos.

Capítulo 3

Vetores, matrizes e Álgebra Linear

3.1 Os pacotes `LinearAlgebra` e `linalg`

O Maple possui dois grandes pacotes de comandos para uso em Álgebra Linear: um mais antigo chamado `linalg` e outro mais recente chamado `LinearAlgebra`. Cada um tem mais de 100 funções, são independentes e executam quase as mesmas tarefas. Para que os exemplos deste capítulo possam ser executados em todas as versões do programa, exploraremos simultanemente esses dois pacotes.

Podemos usar um `with` para ver os comandos de `LinearAlgebra`:

```
> with(LinearAlgebra);
```

[*Add, Adjoint, BackwardSubstitute, BandMatrix, Basis, BezoutMatrix, BidiagonalForm, BilinearForm, CharacteristicMatrix, CharacteristicPolynomial, Column, ColumnDimension, ColumnOperation, ColumnSpace, CompanionMatrix, ConditionNumber, ConstantMatrix, ConstantVector, CreatePermutation, CrossProduct, DeleteColumn, DeleteRow, Determinant, DiagonalMatrix, Dimension, Dimensions, DotProduct, Eigenvalues, Eigenvectors, Equal, ForwardSubstitute, FrobeniusForm, GaussianElimination, GenerateEquations, GenerateMatrix, GetResultDataType, GetResultShape, GivensRotationMatrix, GramSchmidt, HankelMatrix, HermiteForm, HermitianTranspose, HessenbergForm, HilbertMatrix, HouseholderMatrix, IdentityMatrix, IntersectionBasis, IsDefinite, IsOrthogonal, IsSimilar, IsUnitary, JordanBlockMatrix, JordanForm, LA_Main, LUDecomposition, LeastSquares, LinearSolve, Map, Map2, MatrixAdd, MatrixInverse, MatrixMatrixMultiply, MatrixNorm, MatrixScalarMultiply, MatrixVectorMultiply, MinimalPolynomial, Minor, Multiply, NoUserValue, Norm, Normalize, NullSpace, OuterProductMatrix, Permanent, Pivot, QRDecomposition, RandomMatrix, RandomVector, Rank, ReducedRowEchelonForm, Row, RowDimension, RowOperation, RowSpace, ScalarMatrix, ScalarMultiply, ScalarVector, SchurForm, SingularValues, SmithForm, SubMatrix, SubVector, SumBasis, SylvesterMatrix, ToeplitzMatrix, Trace, Transpose, TridiagonalForm, UnitVector, VandermondeMatrix, VectorAdd, VectorAngle, VectorMatrixMultiply, VectorNorm, VectorScalarMultiply, ZeroMatrix, ZeroVector, Zip*]

Usamos o `with` novamente para ver a relação de todos os comandos de `linalg`:

```
> with(linalg);
```

[*BlockDiagonal, GramSchmidt, JordanBlock, LUdecomp, QRdecomp, Wronskian, addcol, addrow, adj, adjoint, angle, augment, backsub, band, basis, bezout, blockmatrix, charmat, charpoly, cholesky, col, coldim, colspace, colspan, companion, concat, cond, copyinto, crossprod, curl, definite, delcols, delrows, det, diag, diverge, dotprod, eigenvals, eigenvalues, eigenvectors, eigenvects, entermatrix, equal, exponential, extend, ffgausselim, fibonacci, forwardsub, frobenius, gausselim, gaussjord, geneqns, genmatrix, grad, hadamard, hermite, hessian, hilbert, htranspose, ihermite, indexfunc, innerprod, intbasis, inverse, ismith, issimilar, iszero, jacobian, jordan, kernel, laplacian, leastsqrs, linsolve, matadd, matrix, minor, minpoly, mulcol, mulrow, multiply, norm, normalize, nullspace, orthog, permanent, pivot, potential, randmatrix, randvector, rank, ratform, row, rowdim, rowspace, rowspan, rref, scalarmul, singularvals, smith, stackmatrix, submatrix, subvector, sumbasis, swapcol, swaprow, sylvester, toeplitz, trace, transpose, vandermonde, vecpotent, vectdim, vector, wronskian*]

3.2 Vetores

No pacote `linalg` um vetor $(v_1, v_2, \cdots, v_n)$ pode ser definido com um comando `vector([`$v_1, v_2, \cdots, v_n$`])`. No pacote `LinearAlgebra`, esse mesmo vetor pode ser definido com um comando `Vector([`$v_1, v_2, \cdots, v_n$`])` ou na forma $< v_1, v_2, \cdots, v_n >$.

A enésima coordenada de um vetor `v` pode ser referenciada como sendo `v[n]`.

Exemplo 3.1 *Definir um vetor* $(4, 5, -7)$ *e calcular a soma das suas coordenadas.*

Usando o pacote `linalg`*:*

```
> with(linalg):
> v := vector([4, 5, -7]);
```

$$v := [4, 5, -7]$$

```
> s1 := v[1] + v[2] + v[3];
```

$$s1 := 2$$

Usando agora o pacote `LinearAlgebra`*:*

```
> with(LinearAlgebra):
> w := Vector([4, 5, -7]);
```

$$w := \begin{bmatrix} 4 \\ 5 \\ -7 \end{bmatrix}$$

```
> s2 := w[1] + w[2] + w[3];
```

$$s2 := 2$$

A função `vectdim(`v`)` do pacote `linalg` pode ser usada para se obter a dimensão do vetor v. No pacote `LinearAlgebra` isso pode ser feito com `Dimension(`v`)`.

Exemplo 3.2 *Dados vetores* $\vec{v}$ *e* $\vec{w}$*, calcular suas dimensões.*

Usando `linalg`*:*

```
> with(linalg):
> v := vector([0, -2, 5, 1/3]);
```

$$v := [0, -2, 5, \frac{1}{3}]$$

```
> vectdim(v);
```

$$4$$

Agora, usando `LinearAlgebra`*:*

```
> with(LinearAlgebra):
> w := Vector([4, -5, 11, 1, 0]);
```

$$w := \begin{bmatrix} 4 \\ -5 \\ 11 \\ 1 \\ 0 \end{bmatrix}$$

```
> Dimension(w);
```

$$5$$

3.3 Operações com vetores

No pacote `linalg` as operações básicas com vetores são:

evalm(k*v) Produto do escalar **k** pelo vetor **v**;

crossprod(v, w) Produto vetorial de **v** por **w**;

dotprod(v, w) Produto interno de **v** por **w**. Também pode ser usado na forma `evalm(v &* w)`;

evalm(v + w) Soma dos vetores **v** e **w**;

angle(v, w) Ângulo entre os vetores **v** e **w** (em radianos);

norm(v, 2) Norma do vetor **v**. Esse comando admite um segundo parâmetro que deve ser igual a 2 para que a norma obtida seja a norma euclidiana usual.

Exemplo 3.3 *Sendo* $\vec{u} = (1, 2, 3)$, $\vec{v} = (0, 1, 5)$ *e* $\vec{w} = (5, 0, 2)$, *calcular* $\vec{u} + \vec{v}$, $2\vec{v}$, $\vec{v} - \vec{w}$, $\vec{v} \cdot \vec{w}$, $\vec{t} = \vec{v} \times \vec{w}$ *e o ângulo entre* $\vec{u}$ *e* $\vec{w}$.

```
> with(linalg):
> u := vector([1, 2, 3]):
> v := vector([0, 1, 5]):
> w := vector([5, 0, 2]):
>
> evalm(u + v);  # adicao de vetores
```

$$[1, 3, 8]$$

```
> evalm(2*v);    # produto por escalar
```

$$[0, 2, 10]$$

```
> evalm(v - w);  # subtracao de vetores
```

$$[-5, 1, 3]$$

```
> evalm(v &* w); # produto interno
```

$$10$$

```
> t := crossprod(v, w); # produto vetorial
```

$$t := [2, 25, -5]$$

```
> angle(u, w);
```

$$\arccos\left(\frac{11\sqrt{14}\sqrt{29}}{406}\right)$$

O `evalm` *deve ser usado na hora de mostrar o resultado final de uma expressão vetorial. Se ele não for usado, os cálculos ficam apenas indicados.*

```
> evalm(u - v + 3*w - 2*t);
```

$$[12, -49, 14]$$

```
> u - v + 3*w - 2*t;
```

$$u - v + 3w - 2t$$

As operações básicas com vetores no pacote `LinearAlgebra` estão listadas a seguir:

VectorScalarMultiply(v, k) Produto do escalar **k** pelo vetor **v**. Pode ser usado na forma **k*****v**;

CrossProduct(v, w) Produto vetorial de **v** por **w**;

DotProduct(v, w) Produto interno de **v** por **w**. Pode ser usado na forma $\mathbf{v} \cdot \mathbf{w}$;

$\mathbf{v} + \mathbf{w}$ Soma dos vetores **v** e **w**;

VectorAngle(v, w) Ângulo entre **v** e **w** (em radianos);

VectorNorm(v, 2) Norma euclidiana do vetor **v**.

Exemplo 3.4 *Usando as funções do* `LinearAlgebra`*, definimos dois vetores u e v e fazemos diversas operações com eles. Definimos também uma função $Norma(v)$ como sendo equivalente a $VectorNorm(v, 2)$.*

```
> u := <1, 1, -1>:      # definição do vetor u
> v := <5, 0, -3>:      # definição do vetor v
>
> VectorScalarMultiply(v, 5);    # produto de v pelo escalar 5
```

$$\begin{bmatrix} 25 \\ 0 \\ -15 \end{bmatrix}$$

```
> Norma := v -> VectorNorm(v, 2);
```

$$Norma := v \rightarrow LinearAlgebra : -VectorNorm(v, 2)$$

```
> Norma(u);
```

$$\sqrt{3}$$

```
> alpha := VectorAngle(u, v);     # ângulo entre u e v
```

$$\alpha := \arccos\left(\frac{4}{51}\sqrt{3}\sqrt{34}\right)$$

```
> convert(alpha, degrees);        # converte alpha em graus
```

$$180\frac{\arccos\left(\frac{4}{51}\sqrt{3}\sqrt{34}\right)}{\pi}\ degrees$$

```
> evalf(%);    # número decimal que corresponde ao ângulo em graus
```

$$37.61611202\ degrees$$

```
> w := CrossProduct(u, v);   # produto vetorial u x v
```

$$w := \begin{bmatrix} -3 \\ -2 \\ -5 \end{bmatrix}$$

```
> VectorAngle(w, u);   # verificando se w e u são ortogonais
```

$$\frac{1}{2}\pi$$

```
> VectorAngle(w, v);   # verificando se w e v são ortogonais
```

$$\frac{1}{2}\pi$$

```
> convert(%, degrees);   # convertendo o ângulo anterior para graus
```

$$90\ degrees$$

Exemplo 3.5 *Apesar da grande semelhança, um vetor não deve ser confundido com uma lista. Definimos um vetor e uma lista e usamos o comando* `whattype` *para detectar o tipo de cada um.*

```
> a := [1, 2, 3]: whattype(a);
```

$$list$$

```
> b := <1, 2, 3>: whattype(b);
```

$$Vector_{column}$$

Neste caso, obteremos uma mensagem de erro se tentarmos usar um comando como `CrossProduct(a, b)`*.*

Exemplo 3.6 *Neste exemplo definimos uma função* `Misto`(u, v, w) *que calcula o produto misto de três vetores u, v e w. Depois, calculamos o produto misto de a, b, c.*

```
> with(linalg):
> Misto := (u, v, w) -> dotprod(u, crossprod(v, w));
```

$$Misto := (u, v, w) \rightarrow dotprod(u, crossprod(v, w))$$

```
> a:=vector([1, 2, 3]):  b:=vector([1, 1, 1]):  c:=vector([0, 0, 1]):
> `[a, b, c]` = Misto(a, b, c);
```

$$[a, b, c] = -1$$

Exemplo 3.7 *Podemos provar identidades vetoriais que às vezes têm uma verificação trabalhosa. Por exemplo, vamos provar que*

$$\vec{u} \times (\vec{v} \times \vec{w}) + \vec{v} \times (\vec{w} \times \vec{u}) + \vec{w} \times (\vec{u} \times \vec{v}) = \vec{0}$$

é válido para quaisquer vetores do $\mathbb{R}^3$.

```
> restart: with(linalg):
> u:=vector([a, b, c]); v:=vector([d, e, f]); w:=vector([g, h, i]);
```

$$u := [a, b, c]$$
$$v := [d, e, f]$$
$$w := [g, h, i]$$

```
> M := crossprod(u, crossprod(v, w));
```

$$M := [b(dh - eg) - c(fg - di), c(ei - fh) - a(dh - eg), a(fg - di) - b(ei - fh)]$$

```
> N := crossprod(v, crossprod(w, u));
```

$$N := [e(gb - ha) - f(ia - gc), f(hc - ib) - d(gb - ha), d(ia - gc) - e(hc - ib)]$$

```
> P := crossprod(w, crossprod(u, v));
```

$$P := [h(ae - bd) - i(cd - af), i(bf - ce) - g(ae - bd), g(cd - af) - h(bf - ce)]$$

```
> simplify(evalm(M + N + P));
```

$$[0, 0, 0]$$

3.4 Matrizes

No pacote `linalg`, uma matriz

$$\begin{bmatrix} a_{11} & a_{12} & \cdots a_{1n} \\ a_{21} & a_{22} & \cdots a_{2n} \\ \cdots & \cdots & \cdots\cdots \\ a_{m1} & a_{m2} & \cdots a_{mn} \end{bmatrix}$$

pode ser definida com um comando

$$\texttt{matrix}([[a_{11}, a_{12}, \cdots, a_{1n}], [a_{21}, a_{22}, \cdots a_{2n}], \cdots, [a_{m1}, a_{m2}, \cdots, a_{mn}]])\,.$$

Depois de definida, podemos fazer referência separadamente aos elementos da matriz. O elemento A_{ij} na i-ésima linha e j-ésima coluna da matriz A pode ser referenciado como `A[i, j]`.

Exemplo 3.8 *Vamos definir uma matriz* X*, modificar os elementos* X_{13} *e* X_{22} *e listar a matriz modificada.*

```
> with(linalg):
> X := matrix([[1,2,3], [4,5,6]]);
```

$$X := \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

```
> X[2, 3]; # lista o elemento X índice 23
```

$$6$$

```
> X[1, 3] := 0; X[2, 2] = -101;
```

$$X_{1,\,3} := 0$$

$$X_{2,\,2} := -101$$

```
> evalm(X);
```

$$\begin{bmatrix} 1 & 2 & 0 \\ 4 & -101 & 6 \end{bmatrix}$$

Uma matriz (a_{ij}) também pode ser definida fornecendo-se ao comando `matrix` a sua ordem e uma função f de duas variáveis. Neste caso o elemento a_{ij} será definido como sendo igual a $f(i, j)$.

Exemplo 3.9 *Neste exemplo definimos uma matriz* 2×2 *cujo elemento da linha* i *e coluna* j *é definido como sendo* $f(i, j) = i + j - 10$.

```
> f := (i,j) -> i+j-10:
> matrix(2, 2, f);
```

$$\begin{bmatrix} -8 & -7 \\ -7 & -6 \end{bmatrix}$$

Essa matriz também poderia ter sido definida na forma `matrix(2, 2, (i,j)->i+j-10)`.

No pacote `LinearAlgebra` a definição de matrizes é feita com um comando `Matrix` que é semelhante ao `matrix` do pacote `linalg`.

Exemplo 3.10 *Definimos uma matriz* 5×5 *a partir de uma lei de formação* $A_{ij} = g(i, j)$ *onde* g *é uma função dada.*

```
> with(LinearAlgebra):
> g := (x,y) -> x - y + 1 + max(x, y)*(max(x, y) - 1):
> A := Matrix(5, 5, g);
```

$$A := \begin{bmatrix} 1 & 2 & 5 & 10 & 17 \\ 4 & 3 & 6 & 11 & 18 \\ 9 & 8 & 7 & 12 & 19 \\ 16 & 15 & 14 & 13 & 20 \\ 25 & 24 & 23 & 22 & 21 \end{bmatrix}$$

Uma matriz também pode ser definida por colunas. Para isso, basta fornecer cada coluna entre sinais de menor e maior, separá-las por barras verticais e envolver todas as colunas por sinais de menor e maior.

Exemplo 3.11 *Neste exemplo definimos uma matriz M por colunas.*

```
> M := <<1,2,3> | <4,5,6> | <7,8,9> | <10,11,12>>;
```

$$M := \begin{bmatrix} 1 & 4 & 7 & 10 \\ 2 & 5 & 8 & 11 \\ 3 & 6 & 9 & 12 \end{bmatrix}$$

Exemplo 3.12 *Definir uma matriz A de ordem 9×9 linha por linha com o comando* `matrix`*. Depois, definir uma outra matriz $B := A^2$ (igual à matriz identidade de ordem 9).*

```
> restart: with(linalg):
> A := matrix([[c,b,d,e,a,e,d,b,c], [b,e,e,b,0,-b,-e,-e,-b],
>             [d,e,c,-b,-a,-b,c,e,d], [e,b,-b,-e,0,e,b,-b,-e],
>             [a,0,-a,0,a,0,-a,0,a],  [e,-b,-b,e,0,-e,b,b,-e],
>             [d,-e,c,b,-a,b,c,-e,d], [b,-e,e,-b,0,b,-e,e,-b],
>             [c,-b,d,-e,a,-e,d,-b,c]]);
```

$$\begin{bmatrix} c & b & d & e & a & e & d & b & c \\ b & e & e & b & 0 & -b & -e & -e & -b \\ d & e & c & -b & -a & -b & c & e & d \\ e & b & -b & -e & 0 & e & b & -b & -e \\ a & 0 & -a & 0 & a & 0 & -a & 0 & a \\ e & -b & -b & e & 0 & -e & b & b & -e \\ d & -e & c & b & -a & b & c & -e & d \\ b & -e & e & -b & 0 & b & -e & e & -b \\ c & -b & d & -e & a & -e & d & -b & c \end{bmatrix}$$

Escolhendo agora os seguintes valores para a, b, c, d e e: $a = \frac{1}{\sqrt{5}}$, $b = \frac{1}{2}\sqrt{\frac{5-\sqrt{5}}{10}}$, $c = \frac{-1+\sqrt{5}}{4\sqrt{5}}$, $d = \frac{1+\sqrt{5}}{4\sqrt{5}}$, $e = \frac{1}{2}\sqrt{\frac{5+\sqrt{5}}{10}}$.

```
> a := 1/sqrt(5): b := 1/2 *sqrt((5 - sqrt(5))/10):
> c := (-1 + sqrt(5))/(4*sqrt(5)): d := (1 + sqrt(5))/(4*sqrt(5)):
> e := 1/2 * sqrt((5 + sqrt(5))/10):
```

Calcula B e não mostra na tela, pois o resultado é extenso:

```
> B := A^2:
```

Mostrando apenas o primeiro elemento $B[1,1]$:

```
> B[1,1];
```

$$\frac{1}{40}(-1+\sqrt{5})^2 + \frac{7}{10} + \frac{1}{40}(1+\sqrt{5})^2$$

Simplificando-o ...

```
> simplify(B[1,1]);
```

$$1$$

Vamos simplificar todos os elementos da matriz B. Para isso, usamos dois comandos `for` *para fazer i e j variarem de 1 a 9.*

```
> for i from 1 to 9 do                  # para i variando de 1 a 9
>   for j from 1 to 9 do                # para j variando de 1 a 9
>     B[i,j] := simplify(B[i,j]):       # simplifica B[i, j]
>   end do;
> end do;
```

Agora, mostrando quem é a matriz B após a simplificação.

```
> evalm(B);
```

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

3.5 Matrizes especiais

O Maple possui várias funções para construção de tipos particulares de matrizes: matriz identidade, matriz de Vandermonde, matriz de Hilbert e outras.

Exemplo 3.13 *Exemplificamos alguns tipos especiais de matrizes.*

```
> with(LinearAlgebra):
> IdentityMatrix(4); # Matriz Identidade 4 x 4. No pacote linalg
>                    # um comando equivalente e' diag(1, 1, 1, 1)
```

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

```
> VandermondeMatrix(<a, b, c, d>); # ou vandermonde([a, b, c, d]) no linalg
```

$$\begin{bmatrix} 1 & a & a^2 & a^3 \\ 1 & b & b^2 & b^3 \\ 1 & c & c^2 & c^3 \\ 1 & d & d^2 & d^3 \end{bmatrix}$$

```
> HilbertMatrix(4); # ou hilbert(4) no pacote linalg
```

$$\begin{bmatrix} 1 & 1/2 & 1/3 & 1/4 \\ 1/2 & 1/3 & 1/4 & 1/5 \\ 1/3 & 1/4 & 1/5 & 1/6 \\ 1/4 & 1/5 & 1/6 & 1/7 \end{bmatrix}$$

```
> DiagonalMatrix(<1, 2, 3, 4>); # ou diag(1, 2, 3, 4) no linalg
```

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 4 \end{bmatrix}$$

```
> RandomMatrix(4, 4); # Matriz 4 x 4 gerada aleatoriamente. No linalg
>                     # um comando equivalente e' randmatrix(4, 4)
```

$$\begin{bmatrix} -41 & -34 & -56 & 62 \\ 20 & -62 & -8 & -79 \\ -7 & -90 & -50 & -71 \\ 16 & -21 & 30 & 28 \end{bmatrix}$$

3.6 Matrizes inversas

A matriz inversa de M é calculada com o comando `inverse(`M`)` do pacote `linalg` ou com o `MatrixInverse(`M`)` do `LinearAlgebra`.

O cálculo da matriz inversa também pode ser feita com um comando na forma de potência M`^(-1)` ou `evalm(`M`^(-1))`.

No pacote `LinearAlgebra` é possível usar o comando na forma `MatrixInverse(`M`,` *opções*`)`, onde *opções* especifica o método de cálculo utilizado, por exemplo, `method=integer` ou `method=Cholesky`, entre outros.

Exemplo 3.14 *Calcular a inversa de uma matriz A.*

```
> with(linalg):
> A := matrix([[3,1], [-5,2]]);
```

$$A := \begin{bmatrix} 3 & 1 \\ -5 & 2 \end{bmatrix}$$

```
> B := inverse(A);
```

$$B := \begin{bmatrix} \frac{2}{11} & -\frac{1}{11} \\ \frac{5}{11} & \frac{3}{11} \end{bmatrix}$$

```
> evalm(A &* B);   # verificando se o produto é a matriz identidade
```

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Exemplo 3.15 *Uma boa opção aos nomes longos das funções de* `LinearAlgebra` *é a criação de apelidos com o comando* `alias`*, como fazemos neste exemplo em que criamos o apelido* `Inv` *para o comando* `MatrixInverse`*.*

```
> with(LinearAlgebra):
> alias(Inv = MatrixInverse):
> M := Matrix([[24,32,9,12], [40,56,15,21], [15,20,6,8], [25,35,10,14]]);
```

$$M := \begin{bmatrix} 24 & 32 & 9 & 12 \\ 40 & 56 & 15 & 21 \\ 15 & 20 & 6 & 8 \\ 25 & 35 & 10 & 14 \end{bmatrix}$$

```
> Inv(M);
```

$$\begin{bmatrix} 14 & -8 & -21 & 12 \\ -10 & 6 & 15 & -9 \\ -35 & 20 & 56 & -32 \\ 25 & -15 & -40 & 24 \end{bmatrix}$$

3.7 Determinante, traço e matriz transposta

No pacote `linalg`, o determinante, o traço (soma dos elementos da diagonal principal) e a transposta de uma matriz M são calculados com os comandos `det(`M`)`, `trace(`M`)` e `transpose(`M`)`, respectivamente. No pacote `LinearAlgebra`, esses comandos são `Determinant(`M`)`, `Trace(`M`)` e `Transpose(`M`)`, respectivamente.

Exemplo 3.16 *No pacote* `LinearAlgebra` *o comando* `Random(m, n, generator=a..b)` *gera aleatoriamente uma matriz* $m \times n$ *com elementos no intervalo* $[a, b]$. *Neste exemplo, geramos aleatoriamente uma matriz* A *de ordem* 2×2, *calculamos sua transposta* B *e os determinantes e os traços dessas matrizes (que devem ter os mesmos valores).*

```
> with(LinearAlgebra):
> A := RandomMatrix(2, 2, generator=-9..9);
```

$$A := \begin{bmatrix} -4 & 2 \\ 4 & 8 \end{bmatrix}$$

```
> B := Transpose(A);
```

$$B := \begin{bmatrix} -4 & 4 \\ 2 & 8 \end{bmatrix}$$

```
> x := Trace(A); y := Trace(B);
```

$$x := 4$$
$$y := 4$$

```
> z := Determinant(A); w := Determinant(B);
```

$$z := -40$$
$$w := -40$$

Exemplo 3.17 *Usando agora o* `linalg`, *definimos uma matriz* 4×4, *calculamos e fatoramos seu determinante.*

```
> with(linalg);
> M := matrix([[a,b,c,d],[-b,a,-d,c],[-c,d,a,-b],[-d,-c,b,a]]);
```

$$M := \begin{bmatrix} a & b & c & d \\ -b & a & -d & c \\ -c & d & a & -b \\ -d & -c & b & a \end{bmatrix}$$

```
> det(M);
```

$$a^4 + 2a^2b^2 + 2d^2a^2 + 2a^2c^2 + b^4 + 2d^2b^2 + 2b^2c^2 + c^4 + 2c^2d^2 + d^4$$

```
> factor(%); # fatora o determinante de M
```

$$(a^2 + b^2 + c^2 + d^2)^2$$

Exemplo 3.18 *Vamos definir uma matriz de Vandermonde de ordem 3 e calcular seu determinante na forma fatorada. Criamos um apelido* `DET` *para abreviar o comando* `Determinant`.

```
> with(LinearAlgebra):
> alias(DET = Determinant):
> P := VandermondeMatrix(<x, y, z>);
```

$$P := \begin{bmatrix} 1 & x & x^2 \\ 1 & y & y^2 \\ 1 & z & z^2 \end{bmatrix}$$

```
> DET(N);
```

$$yz^2 - zy^2 + xy^2 - xz^2 + x^2z - x^2y$$

```
> factor(%);
```

$$-(-z + y)(x - z)(x - y)$$

3.8 Sistemas lineares

Sistemas lineares aparecem em muitos problemas de Álgebra Linear. Eles podem ser resolvidos de vários modos:

- com um comando `linsolve(`A`,` B`)` do pacote `linalg`, onde A é a matriz dos coeficientes e B é a matriz dos termos constantes;
- com um comando `LinearSolve(`A`, opções)` do pacote `LinearAlgebra`, onde A é a matriz completa dos coeficientes das equações do sistema;
- com um comando `solve(`*equações*`)` (veja seção 2.8).

Exemplo 3.19 *Neste exemplo, resolvemos o sistema*

$$\begin{cases} x+y+z &= 6 \\ x-y-z &= 0 \\ 2x+3y+6z &= 18 \end{cases}$$

cuja matriz completa é

$$A = \begin{bmatrix} 1 & 1 & 1 & 6 \\ 1 & -1 & -1 & 0 \\ 2 & 3 & 6 & 18 \end{bmatrix}.$$

```
> with(LinearAlgebra):
> A := Matrix([[ 1, 1, 1, 6], [1, -1, -1, 0], [2, 3, 6, 18]]);
```

$$A := \begin{bmatrix} 1 & 1 & 1 & 6 \\ 1 & -1 & -1 & 0 \\ 2 & 3 & 6 & 18 \end{bmatrix}$$

```
> LinearSolve(A);   # resolve o sistema definido pela matriz A
```

$$\begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix}$$

Temos assim a solução: $x = 3$, $y = 2$, $z = 1$.

Usando agora o `linalg`:

```
> restart: with(linalg):
> A := matrix([[ 1, 1, 1], [1, -1, -1], [2, 3, 6]]);
```

$$A := \begin{bmatrix} 1 & 1 & 1 \\ 1 & -1 & -1 \\ 2 & 3 & 6 \end{bmatrix}$$

```
> B := matrix([[6], [0], [18]]);
```

$$B := \begin{bmatrix} 6 \\ 0 \\ 18 \end{bmatrix}$$

```
> linsolve(A, B);   # resolve o sistema A.X = B
```

$$\begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix}$$

Exemplo 3.20 *Neste exemplo resolvemos um sistema linear do tipo $A.X = B$ e, no final, conferimos a resposta obtida.*

```
> with(linalg):
> A := <<1,0,2,1>|<1,0,0,1>|<1,0,1,0>|<-1,-1,1,0>|<5,7,3,0>|<0,-2,0,1>>;
```

$$A := \begin{bmatrix} 1 & 1 & 1 & -1 & 5 & 0 \\ 0 & 0 & 0 & -1 & 7 & -2 \\ 2 & 0 & 1 & 1 & 3 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

```
> B := <6, -6,1,-5>;
```

$$B := \begin{bmatrix} 6 \\ -6 \\ 1 \\ -5 \end{bmatrix}$$

```
> X:=linsolve(A, B);
```

$$X := \left[-11 + \frac{3}{2}\_t2 - 6\_t1,\ 6 - \frac{5}{2}\_t2 + 6\_t1,\ 17 - \_t2 + 2\_t1,\ 6 + 7\_t1 - 2\_t2,\ \_t1,\ \_t2\right]$$

O sistema tem duas variáveis livres que foram denotadas na solução por $\_t1$ e $\_t2$.

```
> evalm(A &* X);      #  calcula A.X para verificar se é igual a B
```

$$[6, -6, 1, -5]$$

Exemplo 3.21 *Neste exemplo, fornecemos o sistema como sendo uma lista de equações. Na notação utilizada, $x[i]$ corresponde a x_i Definimos também a lista de variáveis e, depois, usamos o comando* `GenerateMatrix` *do* `LinearAlgebra` *para construir as matrizes dos coeficientes a partir das listas fornecidas.*

```
> sistema := [ x[1] - x[2] + x[3] = 7, 2*x[1] + 3*x[3] = 11,
               x[1]-2*x[2] + 4*x[3] = 0];
```

$$sistema := [x_1 - x_2 + x_3 = 7, 2x_1 + 3x_3 = 11, x_1 - 2x_2 + 4x_3 = 0]$$

```
> variaveis := [x[1], x[2], x[3]];
```

$$variaveis := [x_1, x_2, x_3]$$

Sem opções adicionais, a função `GenerateMatrix` *retorna duas matrizes de coeficientes:*

```
> with(LinearAlgebra):
> GenerateMatrix(sistema, variaveis);
```

$$\begin{bmatrix} 1 & -1 & 1 \\ 2 & 0 & 3 \\ 1 & -2 & 4 \end{bmatrix}, \begin{bmatrix} 7 \\ 11 \\ 0 \end{bmatrix}$$

É melhor usar a função `GenerateMatrix` *com a opção* `augmented=true` *pois assim ela retornará uma única matriz, conveniente para passar para LinearSolve:*

```
> A := GenerateMatrix(sistema, variaveis, augmented=true );
```

$$A := \begin{bmatrix} 1 & -1 & 1 & 7 \\ 2 & 0 & 3 & 11 \\ 1 & -2 & 4 & 0 \end{bmatrix}$$

```
> LinearSolve(A); # resolve o sistema dado
```

$$\begin{bmatrix} 64/7 \\ -2/7 \\ -17/7 \end{bmatrix}$$

O comando `LinearSolve` admite parâmetros opcionais que permitem definir o método de resolução a ser utilizado, por exemplo, `LinearSolve(A, method = 'Cholesky')`, `LinearSolve(M, method = 'LU')`, `LinearSolve(A, method = 'SparseIterative')`, etc.

Exemplo 3.22 *É possível também resolver sistemas lineares de forma simples e eficiente com o comando* `solve` *da biblioteca padrão. Para isso, basta fornecer o conjunto de equações do sistema no formato usual.*

```
> sist := { x + y + z + w + t = 0, x - 2*y - w + 3*t = 1,
                                   2*x - y + z + 4*t = 1 }:
> solve(sist);
```

$$\{t = t, y = -\frac{1}{3} - \frac{1}{3}z - \frac{2}{3}w + \frac{2}{3}t, x = \frac{1}{3} - \frac{2}{3}z - \frac{1}{3}w - \frac{5}{3}t, w = w, z = z\}$$

Podemos observar que o sistema dado tem duas variáveis livres: z e w.

3.9 Matrizes escalonadas

Matrizes na forma escada (escalonadas) são muitos freqüentes e úteis no estudo da Álgebra Linear.

No `LinearAlgebra` esse tipo de operação pode ser efetuado com os comandos `GaussianElimination(`M`)`, que fornece uma matriz triangular inferior pelo método da eliminação de Gauss, e `ReducedRowEchelonForm(`M`)`, que fornece a forma completamente escalonada de M.

No pacote `linalg` uma matriz triangular inferior pode ser obtida a partir de M com um comando `gausselim(`M`)`.

```
> with(LinearAlgebra):
> A := Matrix( [[ 1, 2, 1, -1], [ 2, 1, 1, 1], [3, 3, 2, 0],
                [ -5, 3, 0, 1]]);
```

$$A := \begin{bmatrix} 1 & 2 & 1 & -1 \\ 2 & 1 & 1 & 1 \\ 3 & 3 & 2 & 0 \\ -5 & 3 & 0 & 1 \end{bmatrix}$$

```
> GaussianElimination(A);
```

$$\begin{bmatrix} 1 & 2 & 1 & -1 \\ 0 & -3 & -1 & 3 \\ 0 & 0 & 2/3 & 9 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

```
> ReducedRowEchelonForm(A);
```

$$\begin{bmatrix} 1 & 0 & 0 & -7/2 \\ 0 & 1 & 0 & -11/2 \\ 0 & 0 & 1 & 27/2 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

3.10 Dependência e independência linear

Dados os vetores $v_1, \ldots v_n$, se a equação (ou melhor, o sistema linear) definida por $x_1v_1 + \ldots x_nv_n = 0$ possuir apenas a solução nula $x_1 = \cdots = x_n = 0$, então os vetores dados chamam-se *linearmente independentes*; caso contrário, eles chamam-se *linearmente dependentes*.

Exemplo 3.23 *Dados quatro vetores (que são matrizes de ordem 2×2) vamos inicialmente verificar se eles são linearmente indepedentes.*

```
> M1 := <<1, 2>|<3,  4>>; M2 := << 0, 0>|<1, 1>>;
> M3 := <<1, 0>|<-2, 1>>; M4 := <<-1, 0>|<0, 3>>;
```

$$M1 := \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$

$$M2 := \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix}$$

$$M3 := \begin{bmatrix} 1 & -2 \\ 0 & 1 \end{bmatrix}$$

$$M4 := \begin{bmatrix} -1 & 0 \\ 0 & 3 \end{bmatrix}$$

O comando `evalm` *calcula a combinação linear $aM1 + bM2 + cM3 + dM4$:*

```
> combinacao_linear := evalm(a*M1 + b*M2 + c*M3 + d*M4);
```

$$combinacao\_linear := \begin{bmatrix} a + c - d & 3a + b - 2c \\ 2a & 4a + b + c + 3d \end{bmatrix}$$

```
> conj := convert(combinacao_linear, set); # converte matriz em conjunto
```

$$conj := \{4a + b + c + 3d,\ 2a,\ 3a + b - 2c,\ a + c - d\}$$

```
> solve(conj);  # iguala a zero as equações e resolve o sistema
```

$$\{a = 0,\ b = 0,\ d = 0,\ c = 0\}$$

Verificamos assim que os vetores $M1, M2, M3$ e $M4$ dados são linearmente independentes.

Agora, dado outro vetor M, vamos escrevê-lo como combinação linear dos quatro vetores dados inicialmente.

```
> M := <<11, 13>|<-7, -5>>;
```

$$M := \begin{bmatrix} 11 & -7 \\ 13 & -5 \end{bmatrix}$$

```
> temp := evalm(combinacao_linear - M);
```

$$temp := \begin{bmatrix} a+c-d-11 & 3a+b-2c+7 \\ 2a-13 & 4a+b+c+3d+5 \end{bmatrix}$$

```
> ConjEq := convert(temp, set);      # converte matriz em conjunto
```

$$ConjEq := \{3a+b-2c+7,\ a+c-d-11,\ 2a-13,\ 4a+b+c+3d+5\}$$

```
> sol: = solve(ConjEq); # iguala a zero e resolve o sistema
```

$$sol := \{a = \frac{13}{2},\ b = -\frac{47}{2},\ c = \frac{3}{2},\ d = -3\}$$

Finalmente, substituímos a solução obtida na combinação linear

```
> subs(sol, m = a*m1 + b*m2 + c*m3 + d*m4);
```

$$\begin{bmatrix} 11 & -7 \\ 13 & -5 \end{bmatrix} = \frac{13}{2}\begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix} - \frac{47}{2}\begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix} + \frac{3}{2}\begin{bmatrix} 1 & -2 \\ 0 & 1 \end{bmatrix} - 3\begin{bmatrix} -1 & 0 \\ 0 & 3 \end{bmatrix}$$

Exemplo 3.24 *Neste exemplo, vamos mostrar que as funções* $y_1 = \cos(x)$, $y_2 = \operatorname{sen}(x)$ *e* $y_3 = e^{-x}$ *são linearmente independentes.*

Fazemos uma combinação linear f *com essas funções (definida em forma de função). Depois, atribuímos três valores quaisquer a* x *e obtemos três equações lineares nas variáveis* a, b *e* c*. Daí, resolvemos o sistema linear obtido e verificamos que tem apenas a solução* $a = b = c = 0$*.*

Deve-se ter bastante cuidado ao digitar a função exponencial e^{-x}*. Para o Maple essa função deve ser digitada na forma* $exp(-x)$*.*

```
> f := x -> a*cos(x) + b*sin(x) + c*exp(-x);
```

$$f := x \to a\cos(x) + b\sin(x) + ce^{(-x)}$$

```
> sist := {f(1)=0, f(2)=0, f(3)=0};   # poderiamos ter usado outros
>                                     # valores no lugar de 1, 2, 3
```

$$sist := \{a\cos(1) + b\sin(1) + ce^{(-1)} = 0, a\cos(2) + b\sin(2) + ce^{(-2)} = 0,$$
$$a\cos(3) + b\sin(3) + ce^{(-3)} = 0\}$$

```
> solve(sist):
```

$$\{a = 0,\ b = 0,\ c = 0\}$$

Obtemos assim somente a solução nula para o sistema e, conseqüentemente, os vetores dados são linearmente independentes.

Exemplo 3.25 *Vamos mostrar agora que as funções polinomiais* $p(x) = 1 + 2x + 3x^2$, $q(x) = x + x^2$ *e* $r(x) = -1 + 2x + x^2$ *são linearmente dependentes.*

```
> p := x -> 1 + 2*x + 3*x^2;
> q := x -> x + x^2;
> r := x -> -1 + 2*x + x^2;
```

$$p := x \to 1 + 2x + 3x^2$$
$$q := x \to x + x^2$$
$$r := x \to -1 + 2x + x^2$$

```
> F := a*p(x) + b*q(x) + c*r(x);
```

$$F := a(1+2x+3x^2)+b(x+x^2)+c(-1+2x+x^2)$$

```
> P := collect(F, x);       # soma termos semelhantes
```

$$P := (3a+b+c)x^2+(2a+b+2c)x+a-c$$

```
> sist := {coeffs(P, x)};  # define conjunto de coeficientes de P
```

$$sist := \{a-c,\ 3a+b+c,\ 2a+b+2c\}$$

```
> solve(sist);    # resolve o sistema
```

$$\{b=-4c,\ a=c,\ c=c\}$$

Como obtivemos outras soluções além da solução nula, as funções dadas são linearmente dependentes.

3.11 Bases

No pacote `LinearAlgebra`, o comando `Basis([`*lista de vetores*`])` escolhe entre os vetores fornecidos numa lista aqueles que são linearmente independents, ou seja, escolhe uma base para o subespaço gerado por essa lista de vetores. O comando equivalente a esse no pacote `linalg` é o `basis([`*lista de vetores*`])`.

No pacote `LinearAlgebra`, o comando `SumBasis([[`*lista1*`], [`*lista2*`], ...])` recebe uma lista de listas de vetores e encontra uma base para a soma dos subespaços gerados pelos vetores de cada lista. De modo semelhante, `IntersectionBasis([[`*lista1*`], [`*lista2*`], ...])` encontra uma base para a interseção dos subespaços gerados pelos vetores. Os comandos equivalentes a esses no pacote `linalg` são `sumbasis({`*lista1*`}, {`*lista2*`}, ...)` e `intbasis({`*lista1*`}, {`*lista2*`}, ...)`.

Exemplo 3.26 *Neste exemplo, definimos seis vetores $v1, v2, v3, v4, v5$ e $v6$ com algumas dependências: $v3 = v1 + v2$ e $v6 = v4 + v5$. Depois, calculamos algumas bases de subespaços gerados por eles.*

```
> restart;
> with(LinearAlgebra):
>
> v1 := < 1 | 1 | 1 | 0 >; v2 := < 0 | 1 | 1 | 1>; v3 := <1 | 2 | 2 | 1>;
> v4 := < 0 | 1 | 0 | 0 >; v5 := < 0 | 0 | 1 | 0>; v6 := <0 | 1 | 1 | 0>;
```

$$v1 := [1,1,1,0]$$

$$v2 := [0,1,1,1]$$

$$v3 := [1,2,2,1]$$

$$v4 := [0,1,0,0]$$

$$v5 := [0,0,1,0]$$

$$v6 := [0,1,1,0]$$

No `linalg` *os vetores deveriam ser definidos na forma* `v1 := vector(1,1,1,0)` *etc.*

Vamos encontrar agora uma base do subespaço gerado por $v1, v2, v3, v4, v5, v6$. Na resposta ao comando, observe que v3 e v6 são descartados.

```
> Basis([v1, v2, v3, v4, v5, v6]);
```

$$[[1,1,1,0],[0,1,1,1],[0,1,0,0],[0,0,1,0]]$$

Base do subespaço gerado por $v1, v2, v4$ *e* $v1+v2+v4$*:*

```
> Basis([v1, v2, v4, v1 + v2 + v4]); # no linalg deveria ser
>                          # basis([v1, v2, v4, evalm(v1 + v2 + v4)]);
```

$$[[1,1,1,0],[0,1,1,1],[0,1,0,0]]$$

Base do subespaço gerado por $v1, 2v1$ *e* $3v1$*:*

```
> Basis([v1, 2*v1, 3*v1]); # no linalg deveria ser
>                           # basis(v1, evalm(2*v1), evalm(3*v1));
```

$$[[1,1,1,0]]$$

Base do subespaço-soma $U+V$*, onde* $U = \langle v1, v2\rangle$ *e* $V = \langle v1, v2, v3, v4\rangle$*:*

```
> SumBasis([[v1, v2], [v1, v2, v3, v4]]); # no linalg deveria ser
>                             # sumbasis({v1, v2}, {v1, v2, v3, v4});
```

$$[[1,1,1,0],[0,1,1,1],[0,1,0,0]]$$

Base do subespaço-soma $R+S$*, onde* $R = \langle v1, v2\rangle$ *e* $S = \langle v4, v5\rangle$*:*

```
> SumBasis([[v1, v2], [v4, v5]]);
```

$$[[1,1,1,0],[0,1,1,1],[0,1,0,0],[0,0,1,0]]$$

Base do subespaço-interseção de $\langle v1, v3\rangle$ *com* $\langle v1, v5, v6\rangle$*:*

```
> IntersectionBasis([[v1, v3], [v1, v5, v6]]); # no linalg deveria ser
>                                   # intbasis({v1, v3}, {v1, v5, v6});
```

$$[[-1,-1,-1,0]]$$

Base do subespaço-interseção de $\langle v2, v3, v4\rangle$ *com* $\langle v3, v4, v5\rangle$*:*

```
> IntersectionBasis([[v2, v3, v4], [v3, v4, v5]]);
```

$$[[0,1,0,0],[1,2,2,1]]$$

3.12 Transformações lineares

Sendo U e V espaços vetoriais, uma transformação linear $T : U \longrightarrow V$ é uma função tal que $T(kv) = kT(v)$ e $T(u+v) = T(u)+T(v)$ para quaisquer $k \in \mathbb{R}$ e $u, v \in U$.

Exemplo 3.27 *Inicialmente, definimos duas funções* $R, S : \mathbb{R}^3 \longrightarrow \mathbb{R}^3$ *tais que* $R(x,y,z) = (x-2y, z, x+y)$ *e* $S(x,y,z) = (xy, z, 0)$*. Depois, implementamos as condições de definição de uma transformação linear em forma de funções booleanas (que retornam um valor* `true` *ou* `false`*). Finalmente, definimos uma terceira função booleana* `E_Linear(`F`)` *que retorna* `true` *se* F *for linear ou* `false` *em caso contrário.*

Usamos a função booleana `equal(`v, w`)` *de* `linalg` *que retorna* `true` *se os vetores* v *e* w *forem iguais ou* `false` *em caso contrário. No pacote* `LinearAlgebra` *a função equivalente a essa é* `Equal(`v, w`)`*.*

```
> restart;
> with(linalg):
> nulo := vector([0, 0, 0]):  # define um vetor nulo
>
> R := (x, y, z) -> vector([x - 2*y, z, x + y]):
> S := (x, y, z) -> vector([x*y, z, 0]):
>
> Cond1 := T -> equal(nulo, simplify(evalm(T(k*x, k*y, k*z)
>                                          - k*T(x, y, z)))):
> Cond2 := T -> equal(nulo, simplify(evalm(T(a, b, c) + T(d, e, f)
>                                          - T(a+d, b+e, c+f)))):
> E_Linear := T -> Cond1(T) and Cond2(T):
>
> E_Linear(R);  # verifica se R é linear
```

$$true$$

```
> E_Linear(S);  # verifica se S é linear
```

$$false$$

Portanto, neste exemplo, somente R é linear.

Exemplo 3.28 *Neste exemplo, definimos uma transformação linear T e calculamos sua matriz A com relação à base canônica do $\mathbb{R}^3$ Depois do cálculo da inversa de A, construímos a transformação linear $S = T^{-1}$.*

```
> with(LinearAlgebra):
> T := (x, y, z) -> <x + 2*y - z, x + y, 2*x + 5*y - 4*z>;
```

$$T := (x, y, z) \to \langle x + 2y - z,\, x + y,\, 2x + 5y - 4z \rangle$$

```
> A := Matrix([T(1, 0, 0), T(0, 1, 0), T(0, 0, 1)]);
```

$$A := \begin{bmatrix} 1 & 2 & -1 \\ 1 & 1 & 0 \\ 2 & 5 & -4 \end{bmatrix}$$

```
> B := A^(-1);
```

$$B := \begin{bmatrix} -4 & 3 & 1 \\ 4 & -2 & -1 \\ 3 & -1 & -1 \end{bmatrix}$$

```
> S := (x, y, z) -> B . <x, y, z>;
```

$$S := (x, y, z) \to B \cdot \langle x, y, z \rangle$$

```
> S(x, y, z);
```

$$\begin{bmatrix} -4x + 3y + z \\ 4x - 2y - z \\ 3x - y - z \end{bmatrix}$$

```
> T(x, y, z);
```

$$\begin{bmatrix} x+2y-z \\ x+y \\ 2x+5y-4z \end{bmatrix}$$

```
> S(7, 11, -33); # testando ...
```

$$\begin{bmatrix} -28 \\ 39 \\ 43 \end{bmatrix}$$

```
> T(-28, 39, 43);
```

$$\begin{bmatrix} 7 \\ 11 \\ -33 \end{bmatrix}$$

Usando o pacote `linalg` *este exemplo poderia ficar assim:*

```
> restart: with(linalg):
> T := (x, y, z) -> vector([x + 2*y - z, x + y, 2*x + 5*y - 4*z]):
> A := matrix([T(1, 0, 0), T(0, 1, 0), T(0, 0, 1)]):
> B := inverse(A):
> S := (x, y, z) -> vector([x, y, z]) &* B:
> evalm(S(x, y, z));
```

$$[-4x+3y+z,\ 4x-2y-z,\ 3x-y-z]$$

```
> evalm(T(x, y, z));
```

$$[x+2y-z,\ x+y,\ 2x+5y-4z]$$

3.13 Núcleo de uma transformação linear

Uma base para o núcleo (*kernel*) de uma transformação linear definida por uma matriz T é calculada com o comando `NullSpace(`T`)` no pacote `LinearAlgebra` e com o comando `kernel(`T`)` no pacote `linalg`. A base calculada é mostrada na forma de lista de vetores.

Exemplo 3.29 *Geramos aleatoriamente uma matriz M de ordem 3×3 com elementos no intervalo $[0, 9]$ e calculamos seu núcleo.*

```
> with(LinearAlgebra):
> M := RandomMatrix(3, 3, generator=0..9);
```

$$M := \begin{bmatrix} 5 & 1 & 8 \\ 1 & 1 & 4 \\ 3 & 0 & 8 \end{bmatrix}$$

```
> NullSpace(M);  # calcula ker(M)
```

$$\{\ \}$$

A resposta obtida significa que o núcleo é formado apenas pelo vetor nulo (que corresponde a uma base vazia).

Exemplo 3.30 *Neste exemplo, determinamos o núcleo da transformação linear* $T : \mathbb{R}^4 \longrightarrow \mathbb{R}^3$ *cuja matriz com relação às bases canônicas é* $\begin{bmatrix} 1 & 2 & 0 & 1 \\ 2 & -1 & 2 & -1 \\ 1 & -3 & 2 & -2 \end{bmatrix}$, *isto é,*

$$T(x, y, z, w) \;=\; (x + 2y + w,\, 2x - y + 2z - w,\, x - 3y + 2z - 2w).$$

```
> with(LinearAlgebra):
> T := Matrix([[1, 2,0, 1], [2,-1,2,-1], [1,-3,2,-2]]);
```

$$T := \begin{bmatrix} 1 & 2 & 0 & 1 \\ 2 & -1 & 2 & -1 \\ 1 & -3 & 2 & -2 \end{bmatrix}$$

Verificando se a matriz T *corresponde ao* $T(x, y, z, w)$ *desejado:*

```
> T . <x, y, z, w>;
```

$$T := \begin{bmatrix} x + 2y + w \\ 2x - y + 2z - w \\ x - 3y + 2z - 2w \end{bmatrix}$$

```
> K := NullSpace(T); # calcula uma base para ker(T)
```

$$K := \left\{ \begin{bmatrix} 0 \\ -2 \\ 1 \\ 4 \end{bmatrix}, \begin{bmatrix} 1 \\ -3 \\ 0 \\ 5 \end{bmatrix} \right\}$$

K *é um conjunto de vetores* $\{K[1], K[2]\}$. *Para verificar a resposta obtida, testamos se o produto* $T \cdot K[i]$ *é nulo*

```
> T . K[1];
```

$$\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

```
> T . K[2];
```

$$\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Usando o pacote `linalg` *obtemos outra resposta para o núcleo de* T *que também é uma resposta válida.*

```
> restart: with(linalg):
> T := matrix([[1,2,0,1],[2,-1,2,-1],[1,-3,2,-2]]):
> K := kernel(T);
```

$$K := \{\left[-2, 1, \frac{5}{2}, 0\right],\ \left[-1, 0, \frac{3}{2}, 1\right]\}$$

Testando os vetores obtidos:

```
> evalm (K[1] &* transpose(T));
```

$$[0, 0, 0]$$

```
> evalm(K[2] &* transpose(T));
```

$$[0, 0, 0]$$

3.14 Polinômio característico

O polinômio característico $p(\lambda) = \det(M - \lambda I)$ de uma matriz pode ser calculado usando-se o comando `CharacteristicPolynomial(`*matriz, variável*`)` do pacote `LinearAlgebra` ou o comando `charpoly(`*matriz, variável*`)` do pacote `linalg`.

O conceito de polinômio característico está relacionado com duas outras definições: a de matriz característica e a de matriz companheira, que são calculadas no pacote `LinearAlgebra` com os comandos `CharacteristicMatrix(`*matriz, variável*`)` e `CompanionMatrix(`*polinômio, variável*`)`, respectivamente, e no pacote `linalg` com os comandos `charmat(`*matriz, variável)* e `companion(`*polinômio, variável)*, respectivamente.

Exemplo 3.31 *Neste exemplo calculamos o polinômio característico de uma matriz A. Definimos também uma matriz M como sendo a matriz característica de A e calculamos o seu determinante. Observe que o determinante de M coincide com o polinômio característico de A.*

```
> with(LinearAlgebra):
> A := Matrix([[1, 2, 3, 4], [2, 3, 4, 1], [3, 4, 1, 2], [4, 1, 2, 3]]);
```

$$A := \begin{bmatrix} 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 1 \\ 3 & 4 & 1 & 2 \\ 4 & 1 & 2 & 3 \end{bmatrix}$$

```
> M := CharacteristicMatrix(A, lambda);
```

$$M := \begin{bmatrix} -\lambda+1 & 2 & 3 & 4 \\ 2 & -\lambda+3 & 4 & 1 \\ 3 & 4 & -\lambda+1 & 2 \\ 4 & 1 & 2 & -\lambda+3 \end{bmatrix}$$

```
> CharacteristicPolynomial(A, lambda);
```

$$\lambda^4 - 8\lambda^3 - 28\lambda^2 + 64\lambda + 160$$

```
> Determinant(M);
```

$$\lambda^4 - 8\lambda^3 - 28\lambda^2 + 64\lambda + 160$$

Uma alternativa interessante a esses nomes de comandos quilométricos é o uso de apelidos:

```
> alias(CAR = CharacteristicPolynomial, MC = CompanionMatrix);
```

$$CAR,\ MC$$

```
> CAR(A, y);
```

$$y^4 - 8y^3 - 28y^2 + 64y + 160$$

Exemplo 3.32 *Inicialmente, construímos a matriz companheira M de um polinômio mônico (coeficiente do termo de maior grau igual a 1). Depois, calculamos o polinômio característico de M e observamos que ele é igual ao polinômio dado.*

```
> with(linalg):
> p := 15 + 4*x - 7*x^2 - 11*x^3 + x^4;
```

$$p := 15 + 4x - 7x^2 - 11x^3 + x^4$$

```
> M := companion(p, x)
```

$$M := \begin{bmatrix} 0 & 0 & 0 & -15 \\ 1 & 0 & 0 & -4 \\ 0 & 1 & 0 & 7 \\ 0 & 0 & 1 & 11 \end{bmatrix}$$

```
> charpoly(M, x);   # igual ao polinômio p
```

$$15 + 4x - 7x^2 - 11x^3 + x^4$$

3.15 Autovalores e autovetores

As raízes do polinômio característico de uma matriz A são chamados *autovalores* da matriz. Com o pacote `LinearAlgebra` do Maple, eles podem ser calculados com um comando `Eigenvalues(`A`)`. Esse comando admite alguns parâmetros opcionais (como por exemplo `output=list`) que servem para especificar o formato da saída dos resultados encontrados.

No pacote `linalg`, os autovalores de A podem ser calculados com um comando `eigenvalues(`A`)`.

Exemplo 3.33 *Inicialmente, vamos calcular os autovalores de uma matriz A. Depois, o polinômio característico dessa mesma matriz e suas raízes. Assim, podemos observar que essas raízes coincidem com os valores fornecidos pelo comando* `eigenvalues(A)`.

```
> with(linalg):
> A := matrix([[4, -1, 0], [-2, 1, 1], [0, 0, 2]]);
```

$$A := \begin{bmatrix} 4 & -1 & 0 \\ -2 & 1 & 1 \\ 0 & 0 & 2 \end{bmatrix}$$

```
> eigenvalues(A); # mostra os autovalores de A
```

$$2,\ \frac{5}{2} + \frac{1}{2}\sqrt{17},\ \frac{5}{2} - \frac{1}{2}\sqrt{17}$$

```
> p := charpoly(A, x);
```

$$p := -4 + 12x - 7x^2 + x^3$$

```
> solve(p);
```

$$2,\ \frac{5}{2} + \frac{1}{2}\sqrt{17},\ \frac{5}{2} - \frac{1}{2}\sqrt{17}$$

Exemplo 3.34 *Duas matrizes de mesma ordem da forma A e $P^{-1}AP$ são chamadas* semelhantes *(ou* conjugadas*). Sabe-se que duas matrizes semelhantes têm muitas coisas em comum: mesmo determinante, mesmo traço, mesmos autovalores, etc.*

Neste exemplo, partindo de uma matriz diagonal A com autovalores dados (-3, 1, 5 e 11) construímos uma outra matriz M que têm os mesmos autovalores que A. A matriz M é construída com a ajuda de uma matriz invertível P de mesma ordem e gerada aleatoriamente.

```
> restart:
> with(LinearAlgebra):
> A := DiagonalMatrix(<-3, 1, 5, 11>);
```

$$A := \begin{bmatrix} -3 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 11 \end{bmatrix}$$

```
> P := RandomMatrix(4, 4, generator=-3..3);
```

$$P := \begin{bmatrix} 1 & 1 & -3 & -3 \\ 1 & 1 & 0 & -3 \\ -3 & 0 & 1 & 1 \\ 1 & 0 & -3 & 0 \end{bmatrix}$$

```
> M := P^(-1) . A . P;   # calcula uma matriz conjugada de A
```

$$M := \begin{bmatrix} 15 & 4 & -42 & -12 \\ 72 & 29 & -312 & -72 \\ 4/3 & 4/3 & -3 & -4 \\ 86/3 & 32/3 & -118 & -27 \end{bmatrix}$$

```
> Eigenvalues(M, output=Vector[row]); # mostra os autovalores em forma
>                                     # de matriz-linha
```

$$[1, -3, 5, 11]$$

Se λ é um autovalor de T e $v \neq 0$ é tal que $Tv = \lambda v$ então v chama-se um *autovetor* de T associado a λ. No Maple, os autovetores de T podem ser calculados com um comando `Eigenvectors(`T`)` do pacote `LinearAlgebra` ou com um comando `eigenvectores(`T`)` do `linalg`.

Se for usado um `Eigenvectors(`T`)` do `LinearAlgebra`, serão mostradas duas matrizes como resposta: uma matriz-coluna P com os autovalores e outra matriz Q cujas colunas são os autovetores correspondentes. Se tiver sido usado um comando `Eigenvectors(T, output=list)` então a resposta será uma lista de listas, cada uma contendo um autovalor, a dimensão do autoespaço e uma base formada pelos autovetores associados.

No pacote `linalg`, um comando `eigenvectores(T)` tem como resposta uma lista de listas, cada uma contendo um autovalor, a dimensão do autoespaço e uma base formada pelos autovetores correspondentes.

Exemplo 3.35 *Calculamos os autovetores de uma matriz A e mostramos a resposta de duas maneiras: como uma par de matrizes e como uma lista de listas.*

```
> restart: with(LinearAlgebra):
> A := Matrix([[1, 5, 5, 5, 5], [5, 1, 5, 5, 5], [5, 5, 1, 5, 5],
                [5, 5, 5, 1, 5], [5, 5, 5, 5, 1]]);
```

$$A := \begin{bmatrix} 1 & 5 & 5 & 5 & 5 \\ 5 & 1 & 5 & 5 & 5 \\ 5 & 5 & 1 & 5 & 5 \\ 5 & 5 & 5 & 1 & 5 \\ 5 & 5 & 5 & 5 & 1 \end{bmatrix}$$

```
> v := Eigenvectors(A);
```

$$v := \begin{bmatrix} -4 \\ -4 \\ -4 \\ -4 \\ 21 \end{bmatrix}, \begin{bmatrix} -1 & -1 & -1 & -1 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

```
> P := v[1]   # P é a matriz-coluna dos autovalores de A
```

$$P := \begin{bmatrix} -4 \\ -4 \\ -4 \\ -4 \\ 21 \end{bmatrix}$$

```
> Q := v[2];   # Q é a matriz cujas colunas são os autovetores de A
```

$$Q := \begin{bmatrix} -1 & -1 & -1 & -1 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

```
> Q^(-1) . A . Q;
```

$$\begin{bmatrix} -4 & 0 & 0 & 0 & 0 \\ 0 & -4 & 0 & 0 & 0 \\ 0 & 0 & -4 & 0 & 0 \\ 0 & 0 & 0 & -4 & 0 \\ 0 & 0 & 0 & 0 & 21 \end{bmatrix}$$

```
> w := Eigenvectors(A, output=list);
```

$$w := \left[\left[21, 1, \left\{ \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \right\} \right], \left[-4, 4, \left\{ \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ -1 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ -1 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ -1 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ -1 \end{bmatrix} \right\} \right] \right]$$

Como entender essa lista de listas? É simples: para cada autovalor de A é mostrado ao lado dele a dimensão do autoespaço associado, seguido da base desse autoespaço. Por exemplo, 21 é um autovalor de A associado a um autoespaço de dimensão 1 e cuja base é formada pelo vetor $(1, 1, 1, 1)$.

Usando o pacote `linalg` *obtemos o seguinte:*

```
> restart: with(linalg):
> A := matrix([[1, 5, 5, 5, 5], [5, 1, 5, 5, 5], [5, 5, 1, 5, 5],
>                 [5, 5, 5, 1, 5], [5, 5, 5, 5, 1]]):
> v := eigenvectors(A);
```

$$v := [21, 1, \{[1, 1, 1, 1, 1]\}], [-4, 4, \{[-1, 0, 1, 0, 0], [-1, 0, 0, 1, 0], [-1, 0, 0, 0, 1], [-1, 1, 0, 0, 0]\}]$$

Exemplo 3.36 *Neste exemplo calculamos os autovetores de uma matriz T de ordem* 4×4. *O resultado é mostrado em forma de lista de listas da forma* [*autovalor, dimensão,* { *base* }] *e, no final, escolhemos um dos autovalores* λ *e um dos autovetores associados* v *e testamos se* $vT^t - \lambda v = 0$ *(que é o mesmo que* $Tv - \lambda v = 0$*). Dessa forma, verificamos se* v *é mesmo um autovalor de T associado ao autovalor* λ.

```
> restart:
> with(linalg):
> T := matrix([[-3, 0, 0, 0], [-8, -19, -24, -24], [6, 12, 15, 8],
               [2, 4, 6, 13]]);
```

$$T := \begin{bmatrix} -3 & 0 & 0 & 0 \\ -8 & -19 & -24 & -24 \\ 6 & 12 & 15 & 8 \\ 2 & 4 & 6 & 13 \end{bmatrix}$$

```
> w := eigenvectors(T);
```

$$w := [-3, 2, \{[-2, 1, 0, 0], [-3, 0, 1, 0]\}], [5, 1, \{[0, 1, -2, 1]\}], [7, 1, \{[0, 0, -1, 1]\}]$$

```
> w[1]; # lista correspondente ao primeiro autovalor
```

$$[-3, 2, [-2, 1, 0, 0], [-3, 0, 1, 0]]$$

```
> lambda := w[1][1];
```

$$\lambda := -3$$

```
> v := w[1][3][1];
```

$$v := [-2, 1, 0, 0]$$

```
> evalm(v &* transpose(T) - lambda*v); # equivale a  T.v - lambda*v
>                                      # no pacote LinearAlgebra
```

$$[0, 0, 0, 0]$$

3.16 Polinômio minimal

O *polinômio minimal* (ou *polinômio mínimo*) de uma matriz A é o polinômio mônico $m(x)$ de menor grau tal que $m(A) = 0$. No Maple, ele pode ser calculado com um comando `MinimalPolynomial(`*matriz, variável*`)` do pacote `LinearAlgebra` ou com um comando `minpoly(`*matriz, variável*`)` do `linalg`.

Exemplo 3.37 *Neste exemplo calculamos os polinômios minimal pmin e característico pcar de uma matriz A e verificamos que pcar é divisível por pmin. A partir de pmin criamos uma função polinomial $m(x)$ e verificamos que $m(A) = 0$.*

```
> with(LinearAlgebra):
> A := <<3,4,4,4> | <4,3,4,4> | <4,4,3,4> | <4,4,4,3>>;
```

$$A := \begin{bmatrix} 3 & 4 & 4 & 4 \\ 4 & 3 & 4 & 4 \\ 4 & 4 & 3 & 4 \\ 4 & 4 & 4 & 3 \end{bmatrix}$$

```
> pmin := MinimalPolynomial(A, x);
```

$$pmin := -15 - 14x + x^2$$

```
> pcar := CharacteristicPolynomial(A, x);
```

$$pcar := -15 - 44x - 42x^2 - 12x^3 + x^4$$

```
> divide(pcar, pmin); # verifica se pcar é divisível por pmin
```

$$true$$

```
> m := unapply( pmin, x );      # define uma função m(x)
```

$$m := x \to -15 - 14x + x^2$$

```
> m(A);
```

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Exemplo 3.38 *Neste exemplo calculamos a forma fatorada dos polinômios minimal e característico de uma matriz A dada.*

```
> with(linalg):
> A := matrix([[-5, 1, 0, 0, 0, 0, 0, 0], [ 0,-5, 0, 0, 0, 0, 0, 0],
>              [ 0, 0,-5, 0, 0, 0, 0, 0], [ 0, 0, 0, 6, 1, 0, 0, 0],
>              [ 0, 0, 0, 0, 6, 0, 0, 0], [ 0, 0, 0, 0, 0, 6, 1, 0],
>              [ 0, 0, 0, 0, 0, 0, 6, 0], [ 0, 0, 0, 0, 0, 0, 0, 6]]);
```

$$A := \begin{bmatrix} -5 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -5 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -5 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 6 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 6 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 6 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 6 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 6 \end{bmatrix}$$

```
> C := factor(charpoly(A, x));
```

$$C := (x+5)^3(x-6)^5$$

```
> M := factor(minpoly(A, x));
```

$$M := (x+5)^2(x-6)^2$$

3.17 Forma canônica de Jordan

A função `JordanForm` do pacote `LinearAlgebra` pode ser usada para calcular a forma canônica de Jordan de uma matriz, bem como a base de vetores associada com essa forma canônica. Se for usada na forma `JordanForm(`M`)` será mostrada a forma de Jordan J da matriz M. Se for usada na forma `JordanForm(`M`, output='Q')`, será mostrada uma matriz Q cujas colunas formam uma base de vetores associada à forma de Jordan e, neste caso, $Q^{-1}MQ = J$.

No pacote `linalg`, a forma de Jordan de M é calculada com um comando `jordan(`M`)`. Se for usado na forma `jordan(`M`, 'P')`, será calculado uma matriz P tal que $P^{-1}MP$ é a forma de Jordan de M.

Exemplo 3.39 *Neste exemplo calculamos a forma de Jordan J de uma matriz dada M de ordem 3×3 e uma matriz Q tal que $Q^{-1}MQ = J$.*

```
> with(LinearAlgebra):
> M := <<5, 6, -7> | <-9, -11, 13> | <-4, -5, 6>>;
```

$$M := \begin{bmatrix} 5 & -9 & -4 \\ 6 & -11 & -5 \\ -7 & 13 & 6 \end{bmatrix}$$

```
> J := JordanForm(M);
```

$$J := \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

```
> Q := JordanForm(M, output='Q');
```

$$Q := \begin{bmatrix} -1 & 5 & 1 \\ -1 & 6 & 0 \\ 1 & -7 & 0 \end{bmatrix}$$

```
> Q^(-1) . M . Q;
```

$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

Exemplo 3.40 *Apresentamos neste exemplo duas matrizes A e B de ordem 4×4 que têm o mesmo polinômio característico, o mesmo polinômio minimal, mas não têm a mesma forma de Jordan. Pode-se deduzir daí que elas não são semelhantes.*

```
> with(linalg):
> A := matrix([[ 43,  24, 12,  4], [-95, -55, -30, -10], [76, 48, 29, 8],
>              [-19, -12, -6,  3 ]]);
>
> B:= matrix([[53/7, 15/7, -1/7, 2/7],    [45/7, 76/7, 1/7, 5/7 ],
>             [-45/7, -41/7, 34/7, -5/7], [-522/7, -463/7, 1/7, -23/7]]);
```

$$A := \begin{bmatrix} 43 & 24 & 12 & 4 \\ -95 & -55 & -30 & -10 \\ 76 & 48 & 29 & 8 \\ -19 & -12 & -6 & 3 \end{bmatrix}$$

$$B := \begin{bmatrix} 53/7 & 15/7 & -1/7 & 2/7 \\ 45/7 & 76/7 & 1/7 & 5/7 \\ -45/7 & -41/7 & 34/7 & -5/7 \\ -522/7 & -463/7 & 1/7 & -23/7 \end{bmatrix}$$

```
> charpoly(A, x);
```

$$625 - 500x + 150x^2 - 20x^3 + x^4$$

```
> charpoly(B, x);
```

$$625 - 500x + 150x^2 - 20x^3 + x^4$$

```
> minpoly(A, x);
```

$$25 - 10x + x^2$$

```
> minpoly(B, x);
```

$$25 - 10x + x^2$$

```
> J1 := jordan(A);
```

$$J1 := \begin{bmatrix} 5 & 1 & 0 & 0 \\ 0 & 5 & 0 & 0 \\ 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 5 \end{bmatrix}$$

```
> J2 := jordan(B);
```

$$J2 := \begin{bmatrix} 5 & 1 & 0 & 0 \\ 0 & 5 & 0 & 0 \\ 0 & 0 & 5 & 1 \\ 0 & 0 & 0 & 5 \end{bmatrix}$$

O pacote `LinearAlgebra` possui outra função relacionada com a forma de Jordan de uma matriz. É a função `JordanBlockMatrix([`*lista*`])`, onde *lista* é uma seqüência da forma $[\lambda_1, n_1], [\lambda_2, n_2], \ldots$. Essa função constrói uma matriz diagonal de blocos baseada na lista passada como parâmetro. Cada $[\lambda_i, n_i]$ corresponde a uma matriz (bloco de Jordan) de ordem $n_i \times n_i$ da forma

$$\begin{bmatrix} \lambda_i & 1 & 0 & \cdots & 0 \\ 0 & \lambda_i & 1 & \cdots & 0 \\ 0 & 0 & \lambda_i & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \lambda_i \end{bmatrix}$$

No pacote `linalg`, um bloco de Jordan de ordem n associado a *lambda* pode ser construído com um comando `JordanBlock(`*lambda, n*`)`. Uma matriz diagonal de blocos pode ser construída com um comando `diag(`*bloco1, bloco2, ...*`)`.

Exemplo 3.41 *Construir duas matrizes com a função* `JordanBlockMatrix`*, formadas por blocos de Jordan previamente escolhidos.*

```
> with(LinearAlgebra):
> J1 := JordanBlockMatrix([[lambda, 4]]);            # é equivalente  a
>                          #  JordanBlock(lambda, 4)   no pacote linalg
```

$$J1 := \begin{bmatrix} \lambda & 1 & 0 & 0 \\ 0 & \lambda & 1 & 0 \\ 0 & 0 & \lambda & 1 \\ 0 & 0 & 0 & \lambda \end{bmatrix}$$

```
> J2 := JordanBlockMatrix([[4, 3], [ 5, 2], [ -3, 1]]);   # equivale no
> # linalg a diag(JordanBlock(4,3), JordanBlock(5,2), JordanBlock(-3,1))
```

$$J2 := \begin{bmatrix} 4 & 1 & 0 & 0 & 0 & 0 \\ 0 & 4 & 1 & 0 & 0 & 0 \\ 0 & 0 & 4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 5 & 1 & 0 \\ 0 & 0 & 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 0 & 0 & -3 \end{bmatrix}$$

Exemplo 3.42 *Vamos construir uma matriz que tenha uma forma de Jordan previamente escolhida. Assim, tentamos responder à seguinte questão: dada uma matriz diagonal de blocos de Jordan, qual é uma matriz que tem essa diagonal de blocos dada como forma de Jordan?*

Para isso, geramos uma matriz diagonal de blocos J com o `JordanBlockMatrix`*. Depois, com o auxílio de uma matriz invertível P de mesma ordem que J, calculamos uma matriz $M = P^{-1}JP$ que é semelhante a J. Logo, a forma de Jordan de M é a matriz J dada.*

```
> restart;
> with(LinearAlgebra):
> J := JordanBlockMatrix([[4, 2], [-7, 3]]);
```

$$J := \begin{bmatrix} 4 & 1 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 & 0 \\ 0 & 0 & -7 & 1 & 0 \\ 0 & 0 & 0 & -7 & 1 \\ 0 & 0 & 0 & 0 & -7 \end{bmatrix}$$

```
> P := Matrix([[ 1, 1, 1, 1, 1], [1, 2, 2, 2, 2], [1, 2, 3, 3, 3],
>                                 [1, 2, 3, 4, 4], [1, 2, 3, 4, 5]]);
```

$$P := \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 2 & 2 & 2 \\ 1 & 2 & 3 & 3 & 3 \\ 1 & 2 & 3 & 4 & 4 \\ 1 & 2 & 3 & 4 & 5 \end{bmatrix}$$

```
> A := P^(-1) . J . P;
```

$$A := \begin{bmatrix} 6 & 4 & 4 & 4 & 4 \\ 9 & 22 & 28 & 27 & 27 \\ -10 & -20 & -26 & -18 & -19 \\ 1 & 2 & 3 & -3 & 6 \\ -1 & -2 & -3 & -4 & -12 \end{bmatrix}$$

```
> JordanForm(A);
```

$$\begin{bmatrix} 4 & 1 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 & 0 \\ 0 & 0 & -7 & 1 & 0 \\ 0 & 0 & 0 & -7 & 1 \\ 0 & 0 & 0 & 0 & -7 \end{bmatrix}$$

Para obter esses mesmos resultados com o pacote `linalg`*, basta usar os seguintes comandos:*

```
> with(linalg):
> J := diag(JordanBlock(4, 2), JordanBlock(-7, 3));
> P := matrix([[ 1, 1, 1, 1, 1], [1, 2, 2, 2, 2], [1, 2, 3, 3, 3],
>                                 [1, 2, 3, 4, 4], [1, 2, 3, 4, 5]]);
> A := evalm(P^(-1) &* J &* P);
> jordan(A);
```

3.18 Ortogonalização

Dada um conjunto de vetores $v_1, v_2, \ldots v_n$, é possível construir uma base ortogonal para o subespaço gerado por esses vetores. Esse processo é chamado *Ortogonalização de Gram-Schmidt* e, no pacote `LinearAlgebra`, ele é executado pelo comando `GramSchimidt(`$[v_1, v_2, \ldots, v_n]$`)`. Se, além disso, for acrescentado a esse comando uma opção `normalized`, então a base obtida será ortonormal, ou seja, formada por vetores ortogonais unitários.

O pacote `linalg` possui um comando com mesmo nome que esse e que funciona de forma idêntica.

Exemplo 3.43 *Neste exemplo construímos uma base ortogonal e uma base ortonormal para o subespaço vetorial gerado por quatro vetores dados.*

```
> with(LinearAlgebra):
> v1 := <0, 1, 2, 1>:  v2 := <-1,0,1,0>:  v3 := <0, 2,2,1>:
                                           v4 := <1, -1, 1, -1>:
> # no pacote linalg devemos usar v1 := vector([0, 1, 2, 1]) etc.
>
> base1 := GramSchmidt([v1, v2, v3, v4]);
```

$$base1 := \left[\begin{bmatrix} 0 \\ 1 \\ 2 \\ 1 \end{bmatrix}, \begin{bmatrix} -1 \\ -1/3 \\ 1/3 \\ -1/3 \end{bmatrix}, \begin{bmatrix} -1/4 \\ 3/4 \\ -1/4 \\ -1/4 \end{bmatrix}, \begin{bmatrix} 2/3 \\ 0 \\ 2/3 \\ -4/3 \end{bmatrix}\right]$$

Verificamos agora se os vetores $base1[1]$ *e* $base1[2]$ *obtidos são ortogonais:*

```
> base1[1] . base1[2];
```

$$0$$

Verificamos se $base1[3]$ *e* $base1[4]$ *são ortogonais:*

```
> base1[3] . base1[4];
```

$$0$$

Calculamos uma base ortonormal para $\langle v1, v2, v3, v4\rangle$ *:*

```
> base2 := GramSchmidt([v1, v2, v3, v4], normalized);
```

$$base2 := \left[\begin{bmatrix} 0 \\ \frac{1}{6}\sqrt{6} \\ \frac{1}{3}\sqrt{6} \\ \frac{1}{6}\sqrt{6} \end{bmatrix}, \begin{bmatrix} -\frac{1}{2}\sqrt{3} \\ -\frac{1}{6}\sqrt{3} \\ \frac{1}{6}\sqrt{3} \\ -\frac{1}{6}\sqrt{3} \end{bmatrix}, \begin{bmatrix} -\frac{1}{6}\sqrt{3} \\ \frac{1}{2}\sqrt{3} \\ -\frac{1}{6}\sqrt{3} \\ -\frac{1}{6}\sqrt{3} \end{bmatrix}, \begin{bmatrix} \frac{1}{6}\sqrt{6} \\ 0 \\ \frac{1}{6}\sqrt{6} \\ -\frac{1}{3}\sqrt{6} \end{bmatrix}\right]$$

3.19 Exponencial de uma matriz

O comando `exponential(`A`)` do pacote `linalg` permite calcular a exponencial de uma matriz A, que é definida por $e^A = I + A + \frac{1}{2!}A^2 + \frac{1}{3!}A^3 + \frac{1}{4!}A^4 + \cdots$. Essa matriz é útil na resolução de alguns tipos de sistemas de equações diferenciais.

Exemplo 3.44 *Calcular a exponencial da matriz* $\begin{bmatrix} 6 & 4 & 4 \\ -4 & -4 & -11 \\ 3 & 6 & 13 \end{bmatrix}$ *e seus autovalores.*

```
> with(linalg):
> A := matrix([[6, 4, 4], [-4, -4, -11], [3, 6, 13]]);
```

$$A := \begin{bmatrix} 6 & 4 & 4 \\ -4 & -4 & -11 \\ 3 & 6 & 13 \end{bmatrix}$$

```
> B := exponential(A);
```

$$B := \begin{bmatrix} 3e^4 & 4e^4 & 4e^4 \\ -e^7 & e^4 - 2e^7 & -3e^7 + e^4 \\ e^7 - e^4 & 2e^7 - 2e^4 & -2e^4 + 3e^7 \end{bmatrix}$$

```
> eigenvalues(A);
```

$$7,\ 4,\ 4$$

```
> eigenvalues(B);
```

$$e^7,\ e^4,\ e^4$$

Exemplo 3.45 *O comando* `exponential(`M, t`)` *permite calcular* e^{Mt}*. Desse modo, vamos calcular* e^{Mt} *sendo* $M = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$.

```
> with(linalg):
> M := matrix([[0, 1], [-1, 0]]);
```

$$M := \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

```
> exponential(M, t);
```

$$\begin{bmatrix} \cos(t) & \sin(t) \\ -\sin(t) & \cos(t) \end{bmatrix}$$

3.20 Exercícios

1) Sendo $\vec{a}$, $\vec{b}$, $\vec{c}$ e $\vec{d}$ vetores do $\mathbb{R}^3$, mostre que:
a) $(\vec{a} \times \vec{b}) \cdot (\vec{c} \times \vec{d}) = (\vec{a} \cdot \vec{c})(\vec{b} \cdot \vec{d}) - (\vec{b} \cdot \vec{c})(\vec{a} \cdot \vec{d})$;
b) $(\vec{a} \times \vec{b}) \times \vec{c} = (\vec{a} \times \vec{c})\vec{b} - (\vec{b} \times \vec{c})\vec{a}$.

2) Considere M como sendo a seguinte matriz 8×8:

$$M = \begin{bmatrix} a & b & c & d & l & m & n & p \\ b & -a & -d & c & m & -l & p & -n \\ c & d & -a & -b & n & -p & -l & m \\ d & -c & b & -a & p & n & -m & -l \\ l & -m & -n & -p & -a & b & c & d \\ m & l & p & -n & -b & -a & d & -c \\ n & -p & l & m & -c & -d & -a & b \\ p & n & -m & l & -d & c & -b & -a \end{bmatrix}.$$

a) Calcule e fatore o determinante de M;
b) Calcule $N = MM^t$, verifique que N é uma matriz diagonal, calcule seu determinante e, mais uma vez, o determinante de M.

3) Seja $f(x,y) = x + \dfrac{(x+y-1)(x+y-2)}{2}$. Defina várias matrizes quadradas de diversas ordens cujo termo geral a_{ij} seja igual a $f(i,j)$. Você consegue observar alguma lei de formação dos elementos dessas matrizes?

4) Calcule as inversas das matrizes M_1 e M_2. Inicialmente, suponha matrizes de ordens $4\times 4, 5\times 5, \cdots$; depois tente generalizar.

a) $M_1 = \begin{bmatrix} 0 & 1 & 1 & \cdots & 1 \\ 1 & 0 & 1 & \cdots & 1 \\ 1 & 1 & 0 & \cdots & 1 \\ \cdots & \cdots & \cdots & \ddots & \vdots \\ 1 & 1 & 1 & \cdots & 0 \end{bmatrix}$; b) $M_2 = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & 0 & 1 & \cdots & 1 \\ 1 & 1 & 0 & \cdots & 1 \\ \cdots & \cdots & \cdots & \ddots & \vdots \\ 1 & 1 & 1 & \cdots & 0 \end{bmatrix}$.

5) Determine a solução geral do sistema linear

$$\begin{cases} 38x - 74y + 46z + 84t &= 90 \\ -95x + 185y - 115z - 210t &= -225 \\ 57x - 111y + 69z + 126t &= 135. \end{cases}$$

6) Para alguns valores particulares de n, verifique que os determinantes das matrizes M_3 e M_4 de ordens $n\times n$ são respectivamente iguais a $\dfrac{4^{n+1}-1}{3}$ e a $\cos(nx)$.

a) $M_3 = \begin{bmatrix} 5 & 2 & 0 & \cdots & 0 & 0 \\ 2 & 5 & 2 & \cdots & 0 & 0 \\ 0 & 2 & 5 & \cdots & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 2 & 5 \end{bmatrix}$;

b) $M_4 = \begin{bmatrix} \cos\alpha & 1 & 0 & \cdots & 0 & 0 \\ 1 & 2\cos\alpha & 1 & \cdots & 0 & 0 \\ 0 & 1 & 2\cos\alpha & \cdots & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 2\cos\alpha \end{bmatrix}$.

7) Atribuindo valores particulares a n, verifique que os autovalores da matriz M de ordem $n\times n$ definida por

$$M = \begin{bmatrix} a & b & b & \cdots & b & b \\ b & a & b & \cdots & b & b \\ b & b & a & \cdots & b & b \\ \cdots & \cdots & \cdots & \cdots & \ddots & \vdots \\ b & b & b & \cdots & b & a \end{bmatrix}$$

são iguais a $a-b$ com multiplicidade $n-1$ e $a+(n-1)b$ com multiplicidade 1. Determine os autovetores correspondentes.

8) Calcule os autovalores e os autovetores de

$$P = \begin{bmatrix} 5 & -2 & -3 & -4 \\ 6 & 13 & 7 & 8 \\ -8 & -8 & -1 & -8 \\ 4 & 4 & 4 & 11 \end{bmatrix} \text{ e de } Q = \begin{bmatrix} 7 & 0 & 0 & 0 \\ 4 & 11 & 4 & 4 \\ -8 & -8 & -1 & -8 \\ 4 & 4 & 4 & 11 \end{bmatrix}.$$

Capítulo 4

Gráficos

O Maple possui muitos comandos e opções para construção de gráficos, desde os gráficos planos mais simples até gráficos tridimensionais mais sofisticados. Possui também recursos para construção de animações envolvendo esses tipos de gráficos. Neste capítulo apresentamos uma pequena parte desses recursos.

4.1 Gráficos de funções $y = f(x)$

4.1.1 Gráficos simples

O gráfico de uma função definida por uma expressão algébrica $y = f(x)$ na variável x pode ser construído com o comando `plot`:

`plot(`*f(x), x=a..b, y=c..d, op1, op2, ...*`)`

onde

f(x)	é uma função real
x=a..b	é a variação do x (domínio)
y=c..d	é a variação do y (opcional)
op1, op2, ...	outras opções

Na variação do x ou do y pode aparecer `-infinity` ou `infinity` ($-\infty$ ou ∞).

Exemplo 4.1 *Neste exemplo, construímos o gráfico da parábola $y = x^2$. Definimos a variação do x como sendo o intervalo $[-2, 2]$.*

```
> plot(x^2, x = -2..2);
```

O gráfico é mostrado na Figura 4.1.

Exemplo 4.2 *O comando a seguir constrói o gráfico da função polinomial $y = -x^5 + 3x - 1$. Definimos o intervalo de variação do x como sendo o intervalo $[-2, 3/2]$ e o da variação do y como $[-4, 10]$. O gráfico é o da Figura 4.2.*

```
> plot(-x^5 + 3*x -1, x = -2..3/2, y = -4..10);
```

4.1.2 Opções do comando `plot`

As opções do `plot` são fornecidas em forma de igualdades do tipo *opção = valor*. As possibilidades são:

Figura 4.1:

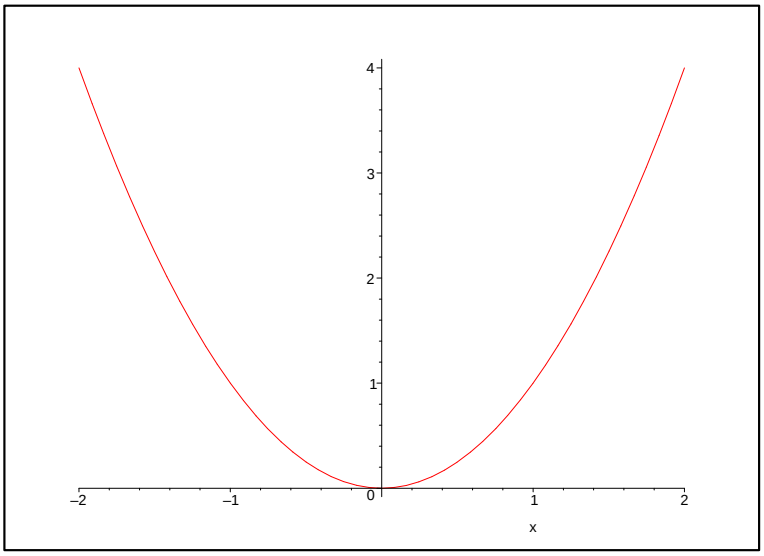
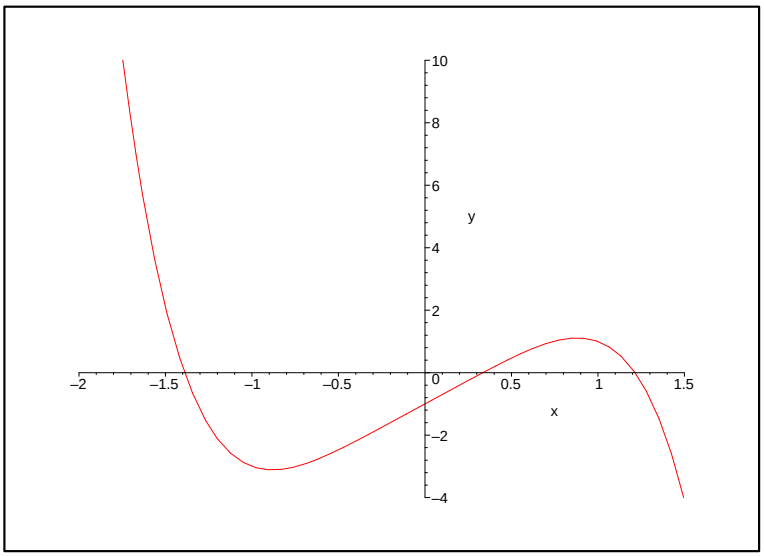


Figura 4.2:



adaptive Pode ser `true` ou `false`. Se `adaptive=true` (o padrão) o gráfico será construído com maior número de pontos onde ele tiver uma maior curvatura. Se `adaptive=false` então será usado um espaçamento uniforme entre os pontos do gráfico.

axes Especifica o tipo dos eixos utilizados: `FRAME`, `BOXED`, `NORMAL` ou `NONE`.

axesfont Especifica o tipo de letra para marcar cada eixo. Pode ser em forma de lista [*fonte, estilo, tamanho*], onde *fonte* é um dos tipos `TIMES`, `COURIER`, `HELVETICA` ou `SYMBOL`. Para o tipo `TIMES`, o estilo pode ser `ROMAN`, `BOLD`, `ITALIC` ou `BOLDITALIC`. Para os tipos `HELVETICA` ou `COURIER` o estilo pode ser omitido ou ser escolhido entre `BOLD`, `OBLIQUE` ou `BOLDOBLIQUE`. O tipo `SYMBOL` não precisa de especificação de estilo. Exemplo: `axesfont = [TIMES, ROMAN, 20]`.

color O mesmo que `colour`. Especifica a cor do gráfico a ser desenhado. Pode ser

aquamarine	black	blue	navy	coral	cyan	brown
gold	green	gray	grey	khaki	magenta	maroon
orange	pink	plum	red	sienna	tan	turquoise
violet	wheat	white	yellow			

ou ser definida na forma `COLOR(RGB,` *r, g, b*`)` onde *r, g, b* são valores reais no intervalo [0, 1] que especificam a proporção de vermelho, verde e azul que compõem a cor. Exemplos: `color = yellow`; `color = COLOR(RGB, 0.3, 0.5, 0.8)`.

coords Indica o sistema de coordenadas utilizado. Pode ser `bipolar, cardioid, cartesian, elliptic, polar`, entre outros. O padrão é o `cartesian`.

discont Se for ajustado para `true`, evitará que sejam desenhadas linhas verticais nos pontos de descontinuidade do gráfico. Se for `false` (que é o padrão) as descontinuidades podem aparecer ligadas por segmentos de retas verticais.

filled Se for ajustado em `true`, a região entre o gráfico e o eixo x será pintada. Na forma padrão, é ajustado em `false`.

font Tipo de letra eventualmente utilizado no gráfico (como um título). Veja a opção `axesfont` citada anteriormente para uma listagem das opções.
Exemplo: `font = [TIMES,ITALIC,14]`

labels=[x,y] Define os nomes dos eixos. O padrão é utilizar os mesmos nomes das variáveis usadas na função fornecida.

labeldirections=[x,y] Especifica a direção dos rótulos que aparecem nos eixos. Os valores de x e y devem ser `HORIZONTAL` or `VERTICAL`. O padrão é `HORIZONTAL`.

labelfont Tipo de letra para os nomes dos eixos. Veja a opção `axesfont` anterior.

legend Legenda do gráfico. É útil quando desenham-se vários gráficos simultaneamente.

linestyle Define se o gráfico é construído pontilhado ou sólido. Pode ser um inteiro de 1 a 4 ou, equivalentemente, uma das palavras: `SOLID, DOT, DASH, DASHDOT`. O padrão é o estilo `SOLID`. Exemplo: `linestyle = 3` (que é o mesmo que `linestyle = DASH`).

numpoints Especifica o número de pontos utilizados para construir o gráfico. O padrão é 50 pontos. Exemplo: `numpoints = 200`

scaling Controla a proporção entre as escalas dos eixos. Pode ser `CONSTRAINED` ou `UNCONSTRAINED`. O padrão é `UNCONSTRAINED`.

style Se for ajustado em `POINT`, o gráfico será construído com pontos isolados. Se for ajustado em `LINE` (que é o padrão) os pontos serão ligados por segmentos de reta.

symbol Símbolo utilizado para marcar os pontos isolados. Pode ser um dos seguintes: `BOX, CROSS, CIRCLE, POINT` ou `DIAMOND`.

symbolsize Tamanho do símbolo utilizado para marcar pontos isolados no gráfico. O padrão é o valor 10.

thickness Largura das linhas usadas para desenhar o gráfico. Pode ser um valor inteiro maior ou igual a 0. O padrão é o valor 0.

tickmarks=[m,n] Especifica as quantidades mínimas de pontos marcados sobre os eixos. Para especificar a quantidade de determinado eixo, deve ser usado uma das opções `xtickmarks = m` ou `ytickmarks = n`.

title Título do gráfico fornecido como *string* (portanto, entre aspas). Se o título contiver mais de uma linha um sinal de \n pode ser usado para sinalizar o final de cada linha. Exemplo: `title=`*"Gráfico da função f(x) = cos(5x)"*

titlefont Especifica o tipo de letra do título. Exemplo: `titlefont=[COURIER,BOLD,5]`

view=[xmin..xmax, ymin..ymax] Especifica as coordenadas máximas e mínimas da curva a ser mostrada. O padrão é mostrar toda a curva.
Exemplo: `view = [-2..1, -1..0.5]`

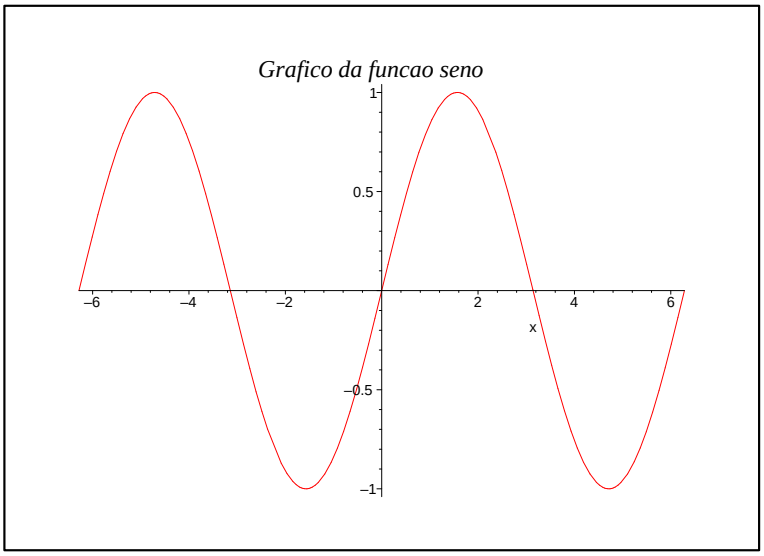
xtickmarks Indica a quantidade mínima de pontos a ser marcada sobre o eixo horizontal ou uma lista de coordenadas.
Exemplo: `xtickmarks = [-2, -1, 1, 3, 5, 7]`

ytickmarks Indica a quantidade mínima de pontos a ser marcada sobre o eixo vertical ou uma lista de coordenadas.
Exemplo: `ytickmarks = 4`

Exemplo 4.3 *O comando a seguir constrói o gráfico da função seno em* $[-2\pi, 2\pi]$. *O resultado é mostrado na Figura 4.3. É mostrado um título do gráfico usando o tipo de letra times itálico de tamanho 20.*

```
> plot(sin(x),x = -2*Pi..2*Pi, title="Grafico da funcao seno",
>                                    titlefont=[TIMES, ITALIC, 20]);
```

Figura 4.3:

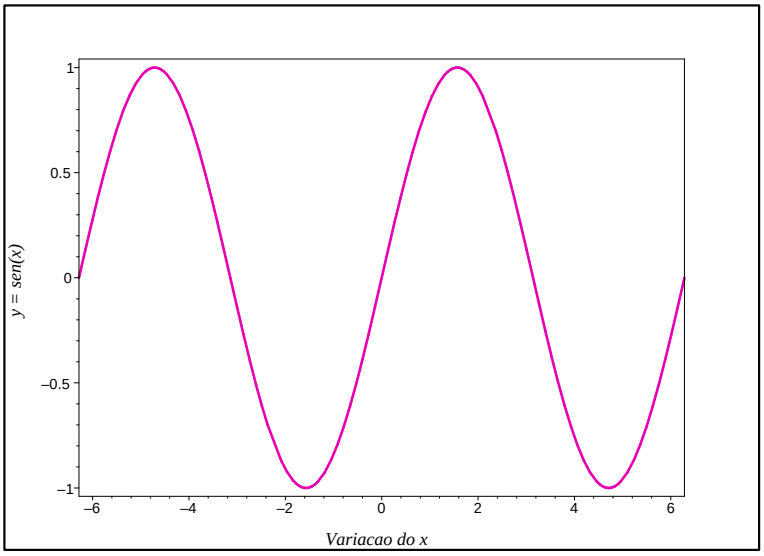


Exemplo 4.4 *O comando a seguir constrói o gráfico da mesma função seno do exemplo anterior, no intervalo* $[-2\pi, 2\pi]$. *Agora, fazemos um gráfico mais grosso (thickness = 4) com uma cor formada por uma mistura de vermelho com azul (color = COLOR(RGB, 0.9, 0.0, 0.7)), os eixos são mostrados como se fossem uma moldura envolvendo o gráfico (axes = BOXED). Além disso, os eixos são rotulados com as expressões "Variacao do*

x" e "y = sen(x)" escritas com tipo de letra TIMES itálico de tamanho 15, com o rótulo do eixo x escrito na direção horizontal e o do eixo y na direção vertical. O gráfico é mostrado na Figura 4.4.

```
> plot(sin(x), x = -2*Pi..2*Pi, axes=BOXED, thickness=4,
>                     color = COLOR(RGB, 0.9, 0.0, 0.7),
>                     labeldirections=[HORIZONTAL, VERTICAL],
>                     labels=["Variacao do x", "y = sen(x)"],
>                     labelfont=[TIMES, ITALIC, 15]);
```

Figura 4.4:



Exemplo 4.5 *O comando a seguir constrói o gráfico da função tangente no intervalo* $[-7, 7]$. *O resultado é muito ruim e é mostrado na Figura 4.5.*

```
> plot(tan(x), x = -7..7);
```

Na Figura 4.6 é mostrado o gráfico da mesma função tangente, limitando-se a variação do y ao intervalo $[-10, 10]$. *Note que ficou com um aspecto bem melhor do que o gráfico anterior, onde deixou-se "livre" a variação do y, ou seja, a critério do Maple.*

```
> plot(tan(x), x=-7..7, y=-10..10);
```

Exemplo 4.6 *O comando a seguir constrói o gráfico da função* `floor(x)` *(maior inteiro menor ou igual a x) no intervalo* $[-7, 7]$. *Como ela é descontínua em* $\mathbb{Z}$, *usamos a opção* `discont=true` *para evitar que as descontinuidades sejam ligadas entre si com segmentos verticais. O resultado é mostrado na Figura 4.7.*

```
> plot(floor(x), x=-3..3, discont=true, thickness=4,
>      title="y = floor(x) = [x]", axesfont=[HELVETICA, OBLIQUE, 12],
>      titlefont=[COURIER, 15], labelfont=[TIMES, ITALIC, 20]);
```

Exemplo 4.7 *O comando a seguir constrói o gráfico da função* $f(x) = sen(1/x)$ *no intervalo* $[-1/2, 1/2]$ *(Figura 4.8) . Não mostramos nenhum tipo de marca nos eixos e aumentamos para* 300 *a quantidade de pontos utilizada para construir o gráfico.*

Figura 4.5:

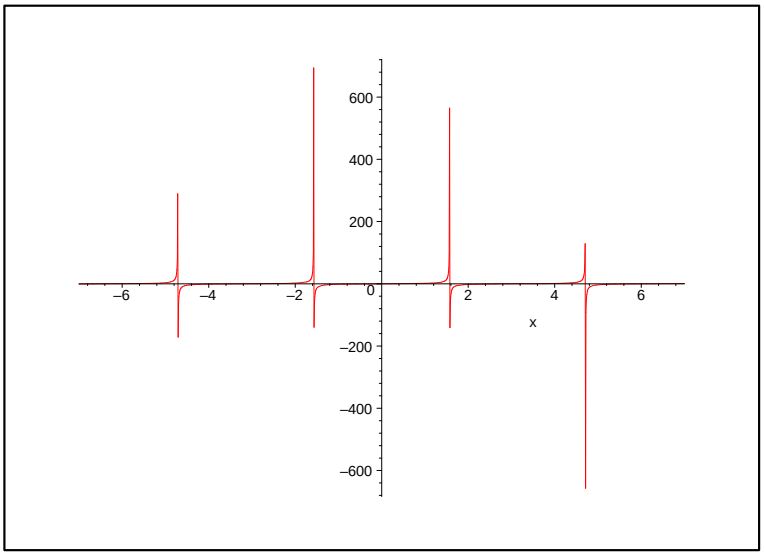
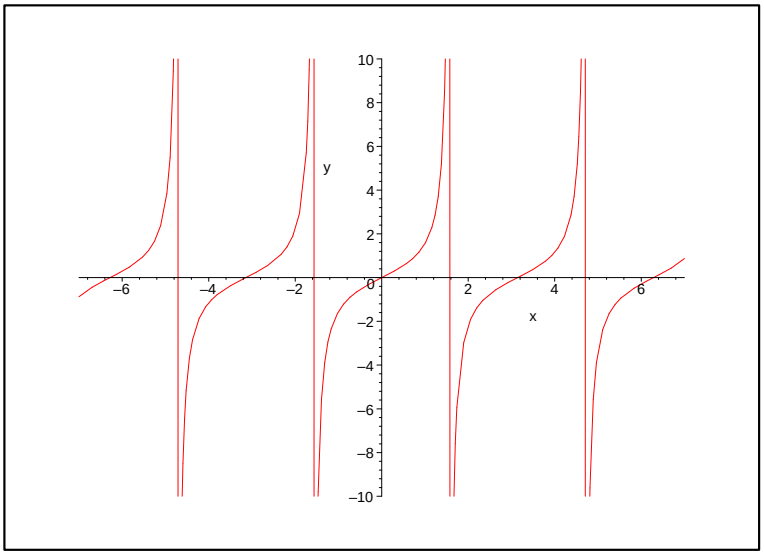


Figura 4.6:




```
> plot(sin(1/x), x = -0.5 .. 0.5, y = -1.3 .. 1.3, numpoints=300,
>                     discont=true, xtickmarks=0, ytickmarks=0);
```

Exemplo 4.8 *O comando a seguir constrói o gráfico da função* $f(x) = \dfrac{x}{1-\cos(x)}$ *no intervalo* $[-30, 30]$ *(Figura 4.9). Construímos o gráfico com pontos isolados usando uma cruz azul de tamanho 15 para assinalar os pontos e aumentamos a quantidade de pontos utilizada para construir o gráfico.*

```
> plot(x/(1-cos(x)), x=-30..30, y=-20..20, numpoints=500,
>       style=point, symbol=cross, symbolsize=15, color=blue);
```

Exemplo 4.9 *O comando a seguir constrói o gráfico da função* $f(x) = 2\,\mathrm{sen}\,x + \mathrm{sen}\,(15x)$ *no intervalo* $[-8, 8]$ *(Figura 4.10). Construimos o gráfico com uma linha de largura 2 e aumentamos para 200 a quantidade de pontos. Além disso, usamos a mesma escala tanto no eixo horizontal quanto no vertical (*`scaling=CONSTRAINED`*).*

Figura 4.7:

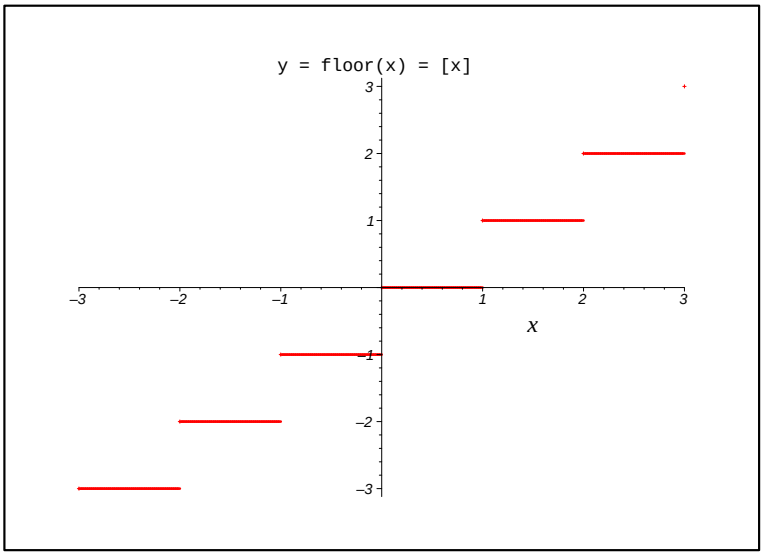
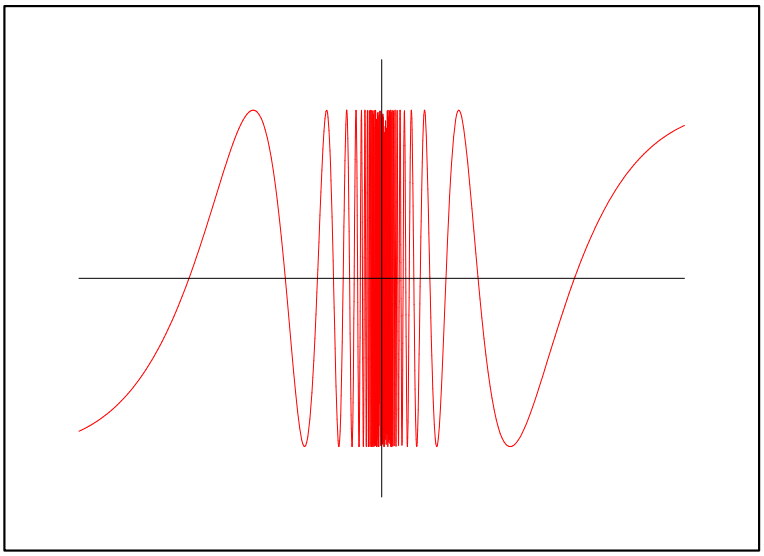


Figura 4.8:



```
> plot(2*sin(x)+sin(15*x), x=-8..8, numpoints=200, linestyle=SOLID,
>       thickness=2, scaling=CONSTRAINED);
```

Exemplo 4.10 *Na figura 4.11 construímos o gráfico da função contínua por partes definida por*

$$y = \begin{cases} x^2 - 9 & \text{se} \quad x < 0 \\ 4x + 3 & \text{se} \quad 0 \leq x < 2 \\ 5 - (x-2)^2 & \text{se} \quad 2 \leq x \end{cases}$$

```
> y := piecewise(x < 0, x^2-9, 0 <= x and x < 2, 4*x + 3,
>                               x >= 2, 5 - (x-2)^2):
> plot(y, x=-4..5, discont=true, thickness=3,
>                          ytickmarks = [-9,0,3,5,11]);
```

Figura 4.9:

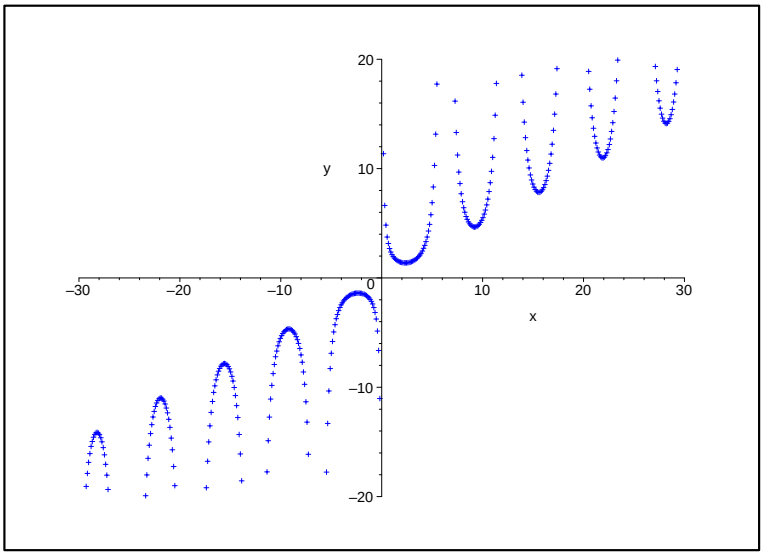
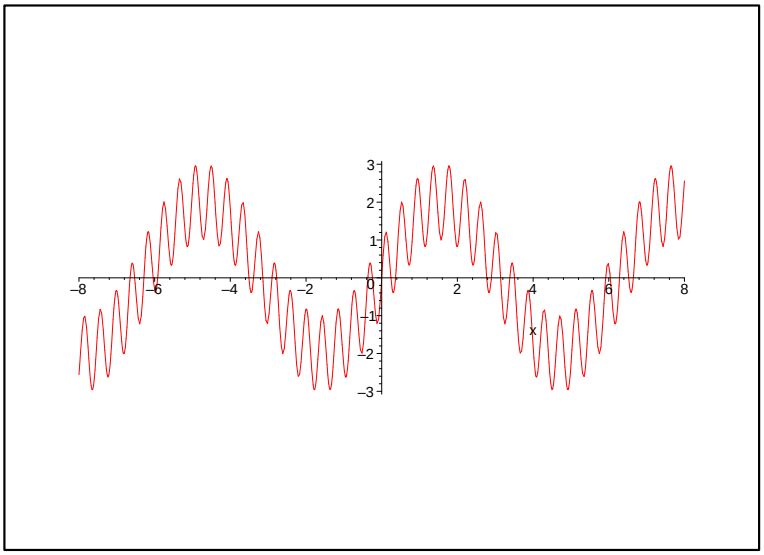


Figura 4.10:



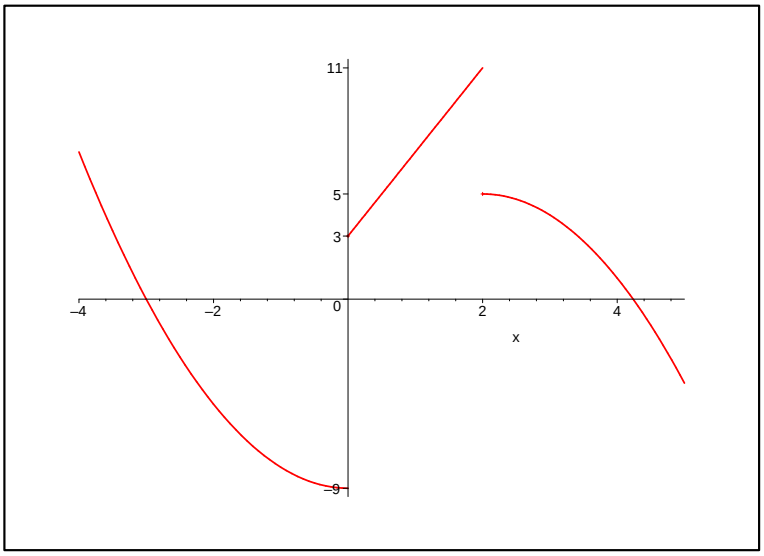
Note que escolhemos precisamente quais são as marcas que queremos que apareçam no eixo y.

4.1.3 Construindo vários gráficos simultaneamente

Para construir vários gráficos em um mesmo sistema de eixos coordenados, a maneira mais simples é fornecer uma lista de funções `[f1, f2, ...]` em vez de só uma função. Nesses casos as funções devem ter um domínio comum. As outras opções fornecidas ao comando `plot` também podem ser na forma de lista (`opção = [op1, op2, ...]`).

Exemplo 4.11 *Neste exemplo construímos em um mesmo sistema de eixo as funções seno e cosseno (Figura 4.12). O gráfico da função seno é contruído com um traçado mais fino (*`thickness = 0`*) e cor azul (*`color = blue`*, que é o mesmo que* `color = COLOR(RGB, 0, 0, 1)`*), enquanto que a função cosseno é construída com um*

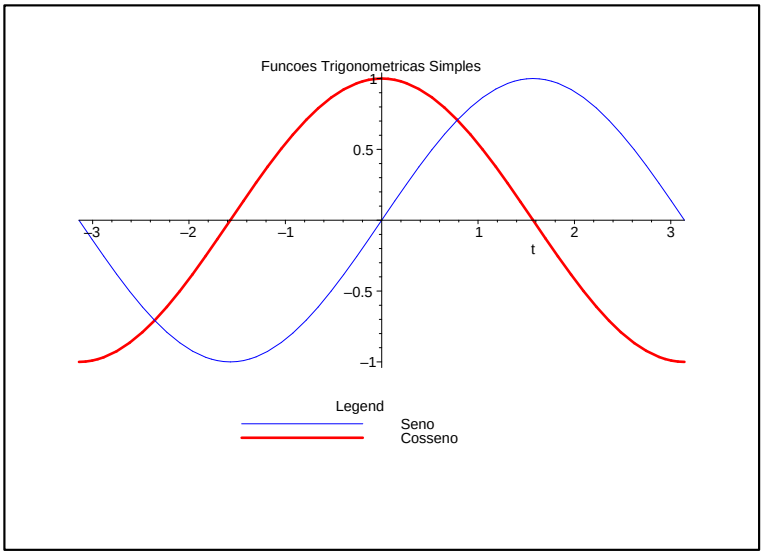
Figura 4.11:



traçado mais grosso (`thickness = 4`*) e cor vermelha (*`color = red`*, o mesmo que* `color = COLOR(RGB, 1, 0, 0)`*).*

```
> plot([sin(t),cos(t)],t=-Pi..Pi,title="Funcoes Trigonometricas Simples",
>      thickness=[0, 4], color = [blue, red], legend=["Seno", "Cosseno"]);
```

Figura 4.12:



Outra maneira de construir vários gráficos simultaneamente é constuir cada um deles separadamente e mostrá-los na tela usando o comando `display` que faz parte do pacote `plots`. Cada gráfico pode ser construído com comandos do tipo

```
variável := plot(...):
```

para que os pontos obtidos sejam guardados em uma variável. É conveniente usar dois pontos no final desse comando para que a extensa relação dos pontos do gráfico não seja mostrada na tela. No final, é só usar um comando

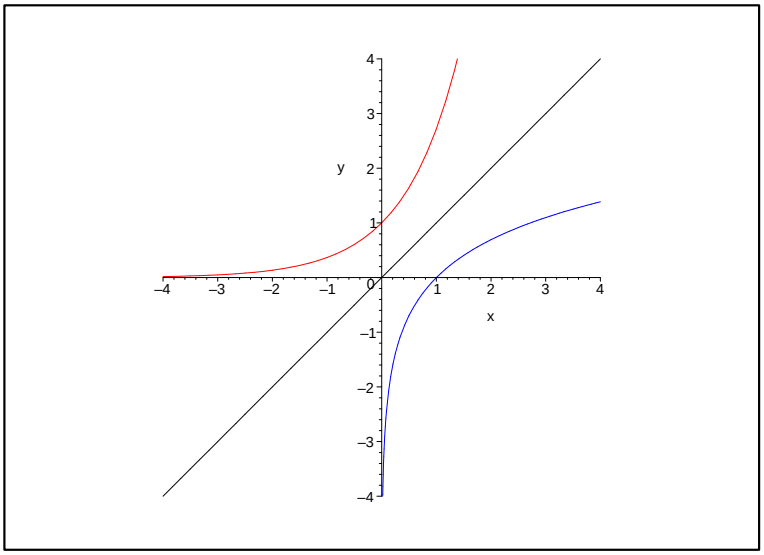
`display(`*variável1, variável2, ...*`)`

para que os gráficos sejam mostrados.

Exemplo 4.12 *Neste exemplo construímos separadamente os gráficos das funções* $f(x) = e^x$ *e* $g(x) = \ln(x)$ *em um mesmo sistema de eixo (Figura 4.13). A função* $h(x) = x$ *também é construída com o traçado do gráfico mais fino.*

```
> graf1 := plot(exp(x), x=-4..4, y=-4..4, scaling=constrained,
>                                  color=red, thickness=2):
> graf2 := plot(ln(x), x=0..4, y=-4..4, scaling=constrained,
>                                  color=blue, thickness=2):
> graf3 := plot(x, x=-4..4, y=-4..4, scaling=constrained,
>                                  color=black, thickness=0):
> with(plots): display(graf1, graf2, graf3);
```

Figura 4.13:



No pacote `plots` tem um comando bastante útil chamado `textplot`. Sua função é colocar mensagens (textos) em um sistema de coordenadas. Sua sintaxe é:

`textplot([`*x1, y1, "mensagem1"*`], [`*x2, y2, "mensagem2"*`],` *opções*`);`

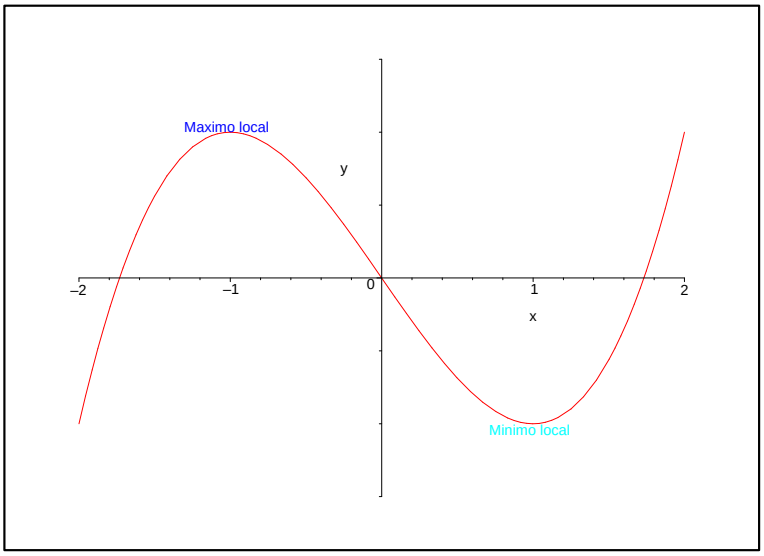
onde *(x1, y1)* corresponde às coordenadas onde será mostrada a *mensagem1*, etc.

As opções são as mesmas do comando `plot` com apenas um acréscimo: a opção `align` que pode assumir valor em um conjunto formado por `BELOW` (abaixo), `ABOVE` (acima), `LEFT` (esquerda) ou `RIGHT` (direita). O `align` corresponde ao alinhamento do texto com relação a um ponto escolhido. Exemplo: `align = ABOVE` .

Exemplo 4.13 *Neste exemplo construímos separadamente o gráfico da função* $y = x^3 - 3x$ *e duas mensagens indicando as posições dos pontos de máximo e mínimo locais (Figura 4.14). As posições foram calculadas usando-se a teoria conveniente.*

```
> with(plots):
> A := plot(x^3 - 3*x, x=-2..2, y=-3..3, ytickmarks=2):
> B := textplot([-1, 2, "Maximo local"], color = blue, align=ABOVE):
> C := textplot([1, -2, "Minimo local"], color= cyan, align=BELOW):
> display(A, B, C);
```

Figura 4.14:



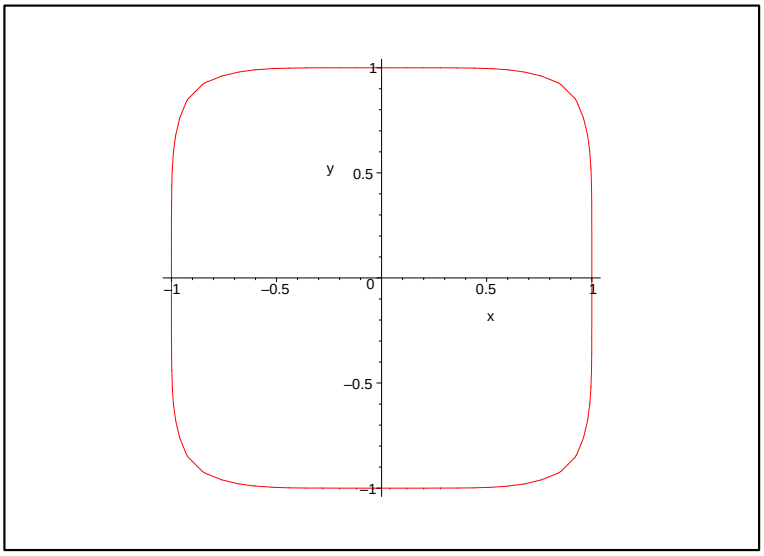
4.2 Gráficos definidos implicitamente

Gráficos definidos implicitamente podem ser construídos com o comando `implicitplot` que pertence ao pacote `plots`. Seu uso é muito parecido com o do comando `plot` já visto anteriormente.

Exemplo 4.14 *Neste exemplo construímos o gráfico da curva definida implicitamente por* $x^6 + y^6 = 1$ *(Figura 4.15).*

```
> with(plots):
> implicitplot(x^6 + y^6 = 1, x=-1..1, y=-1..1, scaling=constrained);
```

Figura 4.15:

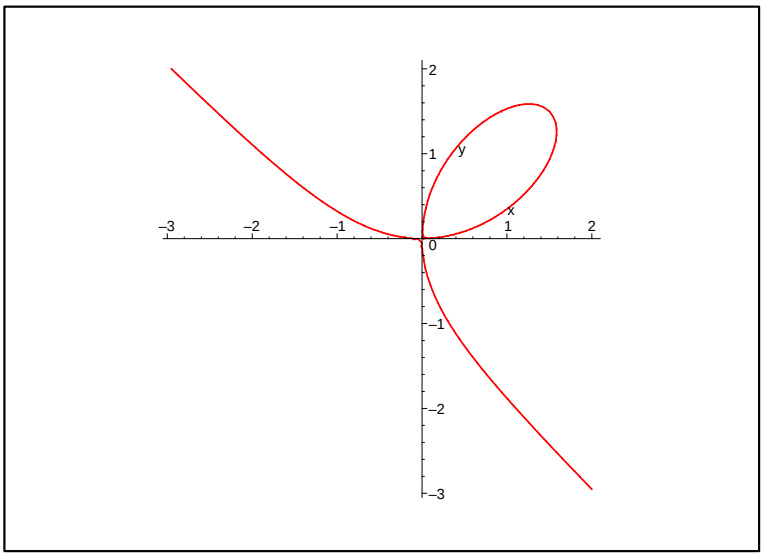


Exemplo 4.15 *Neste exemplo construímos o gráfico da curva definida implicitamente por* $x^3 + y^3 = 3xy$ *(Figura 4.16). Para aumentar a quantidade de pontos do gráfico*

construído com o `implicitplot` *podemos usar a opção* `grid = [`n_x, n_y `]`, *onde* n_x *e* n_y *são inteiros.*

```
> with(plots):
> implicitplot(x^3 + y^3 = 3*x*y, x=-5..2, y=-5..2, scaling=constrained,
>                          thickness=3, grid=[100,100], tickmarks=[4,4]);
```

Figura 4.16:



4.3 Gráficos de curvas parametrizadas

Se a curva for definida por equações paramétricas, então seu gráfico pode ser construído com o comando `plot`. Para isso, basta formar uma lista com as equações envolvidas e a variação do parâmetro:

> `plot([`$f1(t), f2(t), \ldots, t = a..b$`],` *opções*, `...);`

Exemplo 4.16 *Na Figura 4.17 construímos o gráfico da* astróide *definida parametricamente por*

$$\begin{cases} x(t) &= \cos^3 t \\ y(t) &= \operatorname{sen}^3 t \end{cases}, \ 0 \leq t \leq 2\pi.$$

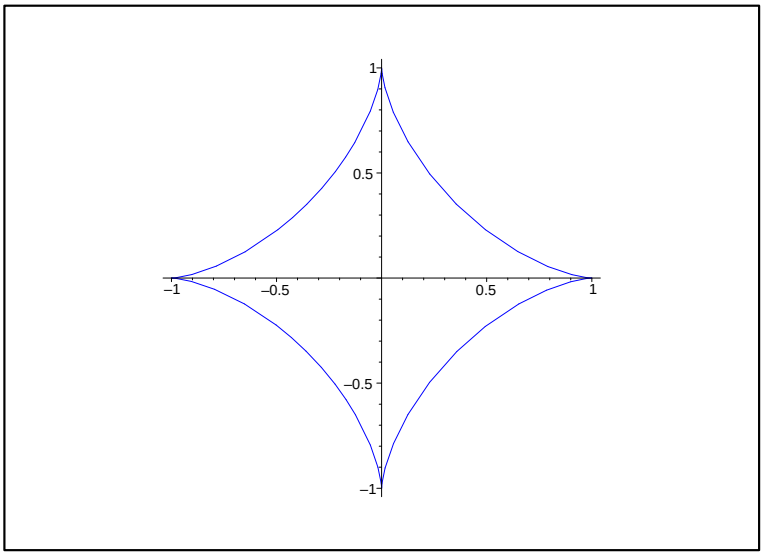
```
> plot([cos(t)^3, sin(t)^3, t=0..2*Pi], thickness=2, color=blue,
>                                       scaling=constrained);
```

É importante não esquecer de colocar a variação do t *entre colchetes. Por exemplo, o comando*

```
> plot([cos(t)^3, sin(t)^3], t=0..2*Pi, thickness=2, color=blue);
```

é muito parecido com o anterior, mas seu resultado é bem diferente: são construídos os gráficos das funções $y = \cos^3 t$ *e* $y = \operatorname{sen}^3 t$ *em um mesmo sistema de eixos.*

Figura 4.17:



Exemplo 4.17 *Na Figura 4.18, construímos o gráfico da* hipociclóide *definida parametricamente por*

$$\begin{cases} x(t) &= (a-b)\cos t + r\cos\left(\dfrac{(a-b)t}{b}\right) \\ y(t) &= (a-b)\,\mathrm{sen}\, t - r\,\mathrm{sen}\left(\dfrac{(a-b)t}{b}\right) \end{cases}, \; a=10,\; b=\frac{17}{2},\; r=1,\; 0\le t\le 34\pi.$$

```
> a := 10; b := 17/2; r := 1;
> plot([(a - b)*cos(t) + r*cos(t*(a - b)/b),
> (a - b)*sin(t) - r*sin(t*(a - b)/b), t=0..34*Pi], scaling=constrained);
```

Variando-se os valores de a, b e r obtemos uma diversidade de formatos das curvas. Refaça este exemplo considerando $a = 11, b = 3, r = 23/10$ e depois $a = 11, b = 8, r = 5$ e compare os resultados. Devemos ter $0 \le t \le \frac{2nb\pi}{a-b}$, onde n é o menor inteiro positivo tal que $\frac{nb}{a-b}$ é um inteiro.

4.4 Gráficos em coordenadas polares

Gráficos em coordenadas polares podem ser construídos com o comando `polarplot` que faz parte do pacote `plots`. Suas opções são as mesmas do `plot`.

Exemplo 4.18 *Neste exemplo, construímos o gráfico de um rosáceo de 8 pétalas definido em coordenadas polares por $r = \mathrm{sen}\,(4\theta)$ (Figura 4.19).*

```
> with(plots):
> polarplot(sin(4*teta), teta=0..2*Pi, axes=none, scaling=constrained);
```

Exemplo 4.19 *Na Figura 4.20 construímos o gráfico de $r = \frac{1}{2} + \cos(t), \quad 0 \le t \le 2\pi$. Para isso, depois do* `with(plots)`*, usamos o comando:*

```
> polarplot(1/2 + cos(t), t=0..2*Pi, scaling=constrained, thickness=3);
```

Figura 4.18:

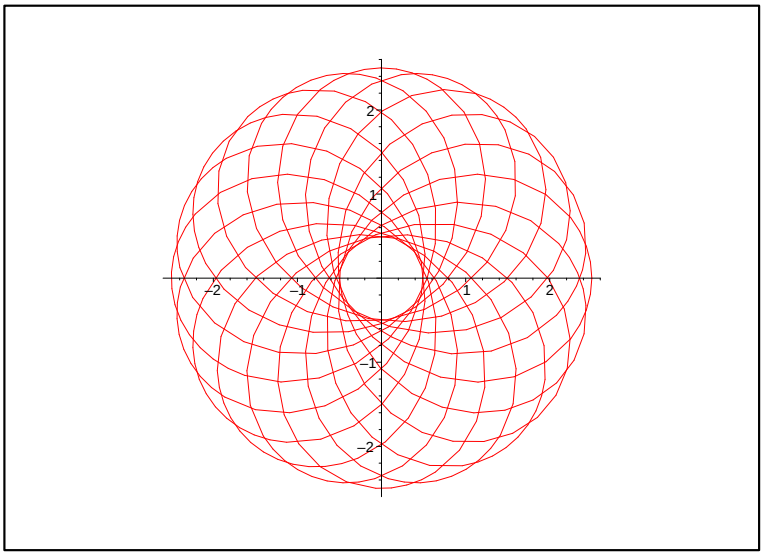
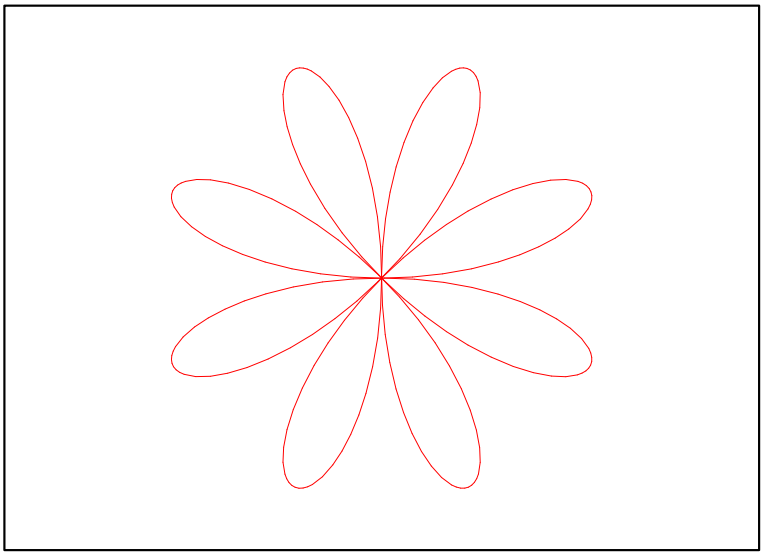


Figura 4.19:



4.5 Gráficos tridimensionais

O Maple possui uma grande variedade de comandos para construção de gráficos tridimensionais. Apresentamos aqui somente alguns comandos básicos.

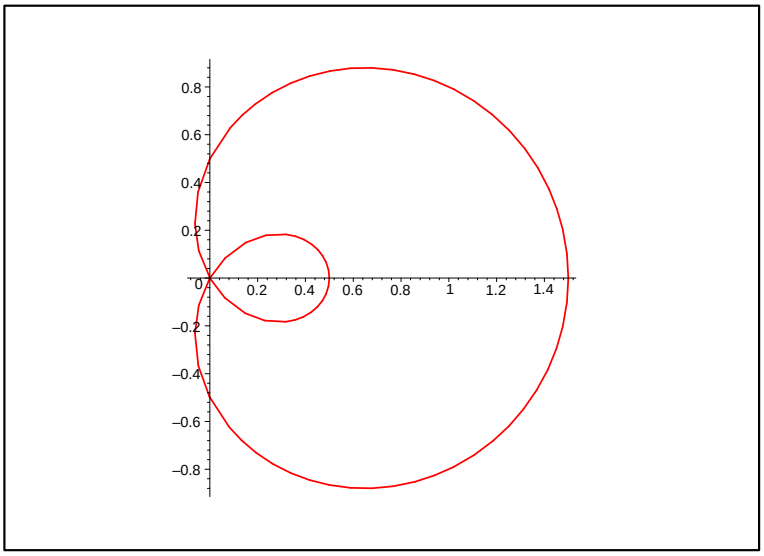
4.5.1 Gráficos de funções $z = f(x, y)$

O gráfico de uma função de duas variáveis reais x, y definida por uma expressão algébrica $f(x, y)$ pode ser construído com um comando

`plot3d(`*f(x, y), x = a..b, y = c..d, op1, op2, ...*`)`

onde

Figura 4.20:

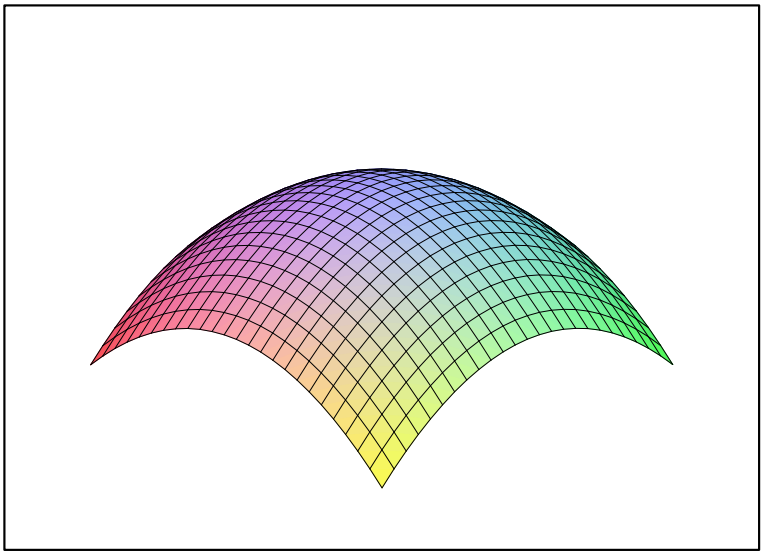


f(x, y)	é uma função real
$x = a..b$	é a variação da variável 1
$y = c..d$	é a variação da variável 2
op1, op2, ...	outras opções

Exemplo 4.20 *Neste exemplo, construímos o gráfico do parabolóide* $z = -x^2 - y^2$. *Definimos o intervalo de variação do* x *e do* y *como sendo o intervalo* $[-2, 2]$. *O resultado é mostrado na Figura 4.21.*

```
> plot3d(-x^2 - y^2, x = -2..2, y=-2..2);
```

Figura 4.21:



Após a construção do gráfico na tela, ele pode ser girado com o auxílio do mouse: basta dar um simples clique com o botão esquerdo e, mantendo o botão esquerdo preso, girar o gráfico para uma nova posição. Pode-se também modificar o gráfico construído com os botões da barra de contexto (abaixo da barra de ferramentas).

4.5.2 As opções do comando `plot3d`

O comando `plot3d` admite as opções `axes, axesfont, color, font, labelfont, labels, linestyle, numpoints, scaling, symbol, symbolsize, thickness, title, titlefont` que são semelhantes às do comando `plot`, definidas na seção 4.1.2.

Opções específicas do `plot3d` são:

contours Especifica o número de curvas de nível utilizado. O valor padrão é 10.

coords Especifica o sistema de coordenadas utilizado. O padrão é o sistema `cartesian`. Algumas das opções são: bispherical, cardioidal, conical, cylindrical, ellipsoidal, logcylindrical, paraboloidal, spherical.

filled=truefalse Pode ser `TRUE` ou `FALSE`. Se for `TRUE`, a região entre o gráfico e o plano xOy é preenchida.

grid=[m,n] Especifica as dimensões da grade retangular com espaçamento uniforme utilizada na construção do gráfico.

lightmodel Especifica a quantidade de fontes luminosas que iluminam o gráfico. Pode ser `none`, `light1`, `light2`, `light3` ou `light4` conforme o modelo tenha 0, 1, 2, 3 ou 4 luzes iluminando, respectivamente.

orientation=[theta,phi] Especifica ângulos de rotações em graus (cujos valores padrões são 45^o) sob as quais o gráfico é mostrado. O valor de `theta` corresponde a uma rotação em torno do eixo `z` e `phi` corresponde a rotação em torno do eixo `y`.

projection Especifica o tipo de projeção utilizado. Pode ser `FISHEYE`, `NORMAL` ou `ORTHOGONAL`. O padrão é a projeção `ORTHOGONAL`.

shading Especifica o modo com o qual a superfície é pintada. Pode ser `XYZ`, `XY`, `Z`, `ZGRAYSCALE`, `ZHUE` ou `NONE`.

style Especifica como o gráfico será desenhado. Pode ser `POINT`, `HIDDEN`, `PATCH`, `WIREFRAME`, `CONTOUR`, `PATCHNOGRID`, `PATCHCONTOUR` ou `LINE`. O valor padrão é `PATCH`.

tickmarks=[l,n,m] Especifica o número de subdivisões utilizadas em cada eixo.

view=zmin..zmax *ou* **[xmin..xmax,ymin..ymax,zmin..zmax]** Indica a parte da superfície para ser mostrada. O padrão é mostrar toda a superfície.

Para as opções `axes, style, projection, shading` e `scaling` os valores podem ser em letras maiúsculas ou minúsculas. Por exemplo `axes=BOXED` é considerado como sendo o mesmo que `axes=boxed`.

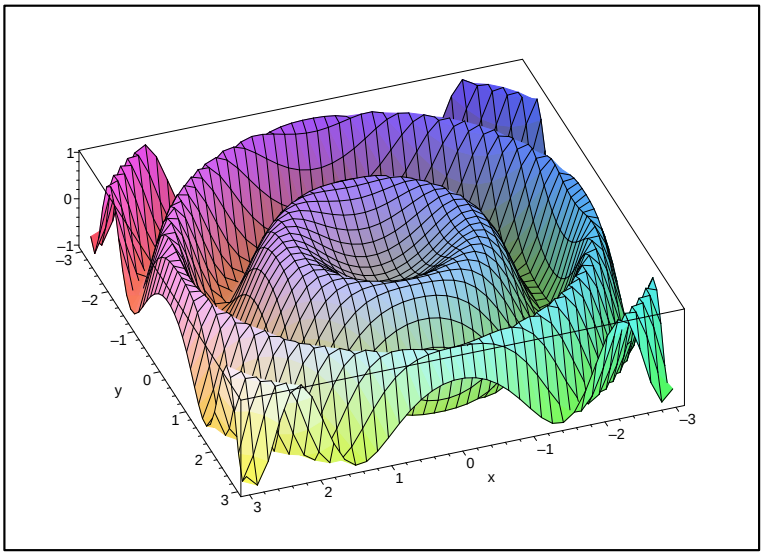
Exemplo 4.21 *Neste exemplo construímos o gráfico de* $z = \operatorname{sen}(x^2 + y^2)$ *com* $-3 \leq x \leq 3$ *e* $-3 \leq y \leq 3$ *(Figura 4.22). É usada uma caixa envolvendo o gráfico (axes = boxed) e ângulos de rotação como sendo 70 e 20 graus e uma grade de* 40×40 *subdivisões.*

```
> plot3d(sin(x^2 + y^2), x=-3..3, y=-3..3, axes=boxed, grid=[40,40],
>                                          orientation=[70,20]);
```

Experimente reconstruir este exemplo alterando os valores da grade e também dos ângulos de rotação.

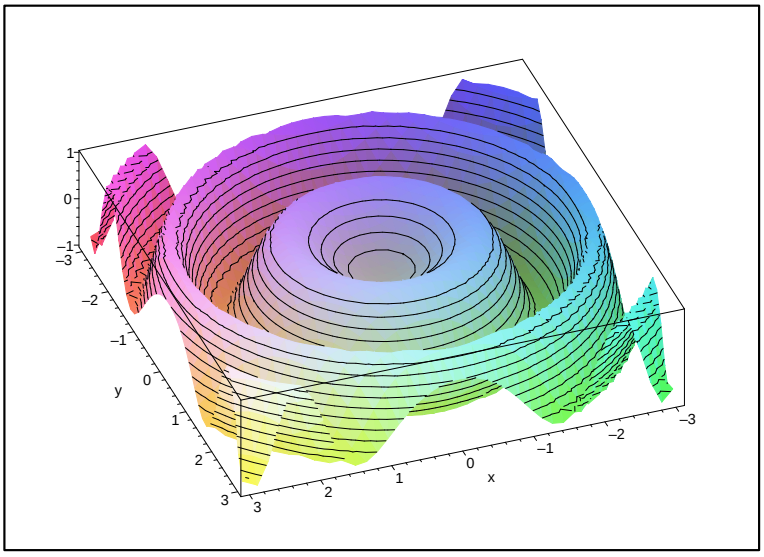
Agora, veja quanto muda o aspecto do mesmo gráfico se acrescentarmos uma opção `style=PATCHCONTOUR` *ao comando* `plot3d` *(Figura 4.23).*

Figura 4.22:




```
> plot3d(sin(x^2 + y^2), x=-3..3, y=-3..3, axes=boxed, grid=[40,40],
>                          orientation=[70,20], style=PATCHCONTOUR);
```

Figura 4.23:

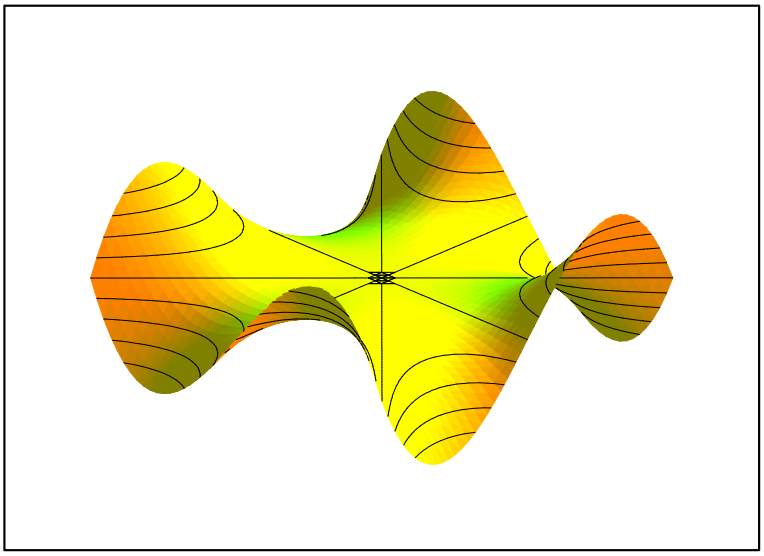


Exemplo 4.22 *Os gráficos normalmente são mostrados com muitas cores. Para que eles sejam mostrados em tonalidades de uma mesma cor, basta acrescentar uma opção* `color` *ao comando* `plot3d`*. Por exemplo, acrescentando uma opção* `color=grey` *o gráfico será todo em tons de cinza.*

Neste exemplo (Figura 4.24) , construímos o gráfico $z = x^3y - xy^3$ *predominantemente amarelo, com duas fontes de luz (*`lightmodel = light2`*), no estilo* `PATCHCONTOUR` *com grade* 45×45*.*

```
> plot3d(x^3*y - x*y^3, x=-4..4, y=-4..4, grid=[45,45],
>        style=PATCHCONTOUR, lightmodel=light2, color=yellow);
```

Figura 4.24:

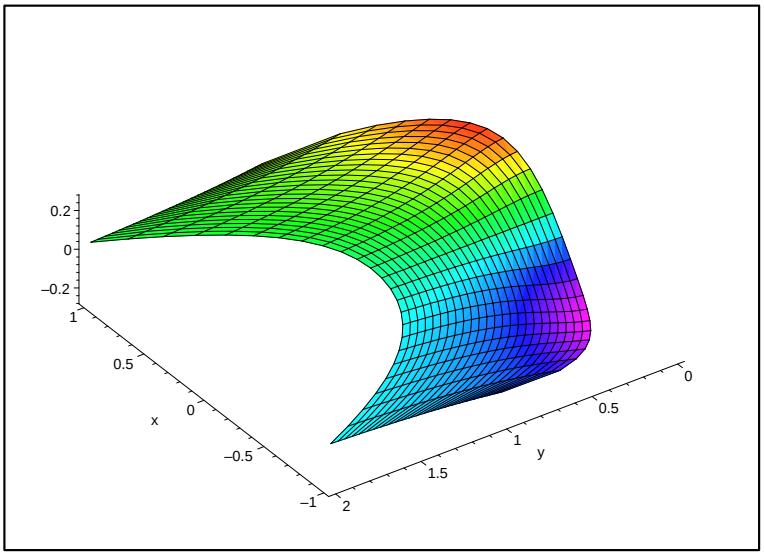


Exemplo 4.23 *Neste exemplo, construímos o gráfico de* $z = \dfrac{x}{1+3x^2+5y^2}$, *com* x *de* -1 *a* 1 *e* y *variando de* $1-\sqrt{1-x^2}$ *a* $1+x^2$. *(Figura 4.25). Note que usamos um domínio que não é retangular.*

Usamos a opção de sombreamento como sendo `ZHUE` *(*`shading=zhue`*) – com isso o gráfico é colorido como um arco-íris, sendo o vermelho a parte mais alta e o violeta a parte mais baixa do gráfico.*

```
> plot3d(x/(1 + 3*x^2 + 5*y^2), x=-1..1, y=1-sqrt(1-x^2)..1+x^2,
>                 axes=frame, shading=zhue, orientation=[145, 25]);
```

Figura 4.25:



4.5.3 Curvas de nível

A biblioteca `plots` possui dois comandos para a construção das curvas de nível do gráfico de uma função $z = f(x, y)$: `contourplot` e `contourplot3d`. A diferença entre elas é que `contourplot` mostra as curvas em um único plano (como é usual nos livros de Cálculo) enquanto que `contourplot3d` mostra as curvas dispostas em vários planos, cada uma na altura original.

A sintaxe desses comandos é semelhante à do comando `plot3d`:

`contourplot(`*f(x,y), x = a..b, y = c..d, op1, op2, ...*`)`

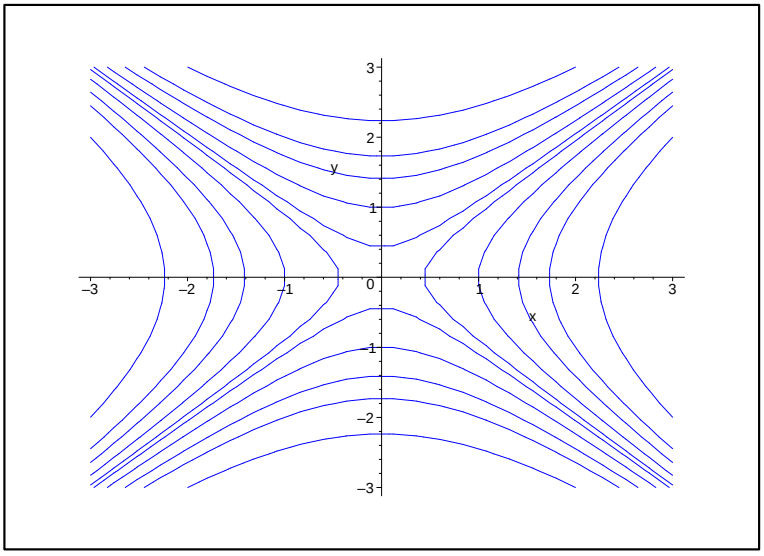
As opções também são parecidas. Destacamos aqui apenas duas opções:

- `filled` que pode ser `TRUE` ou `FALSE`. Se for `TRUE`, então os espaços entre curvas de nível consecutivas são preenchidos gradativamente usando cores fornecidas com a opção `coloring=[cor1, cor2]`.
- A quantidade padrão de curvas mostradas é 8. Esse número pode ser alterado para n curvas bastando para isso acrescentar uma opção `contours = n` ou então fornecer a lista de alturas das curvas de nível `contours = [z1, z2, z3, ...]`.

Exemplo 4.24 *Consideremos a sela $z = x^2 - y^2$ com x e y variando de -3 a 3. Vamos construir na cor azul suas curvas de nível que correspondam aos valores $z = -5$, $z = -3$, $z = -2$, $z = -1$, $z = -1/5$, $z = 1/5$, $z = 1$, $z = 2$, $z = 3$ e $z = 5$ (Figura 4.26).*

```
> with(plots):
> contourplot(x^2 - y^2, x=-3..3, y=-3..3, contours = [-5, -3, -2, -1,
>                                  -1/5, 1/5, 1, 2, 3, 5], color=blue);
```

Figura 4.26:



Exemplo 4.25 *Neste exemplo, construímos as curvas de nível de $z = \operatorname{sen}(x)\operatorname{sen}(y)$, x e y variando no intervalo $[-3, 3]$, usando os comandos `contourplot` e `contourplot3d`. Preenchemos os espaços entre as curvas de nível com tons de cinza e ciano (`filled = true, coloring = [grey, cyan]`).*

```
> with(plots):
> contourplot(sin(x)*sin(y), x=-3..3, y=-3..3, filled=true,
>                                                coloring=[grey,cyan]);
>
> contourplot3d(sin(x)*sin(y), x=-3..3, y=-3..3, filled=true,
>                          orientation=[80,40], coloring=[grey,cyan]);
```

Os resultados estão nas Figuras 4.27 e 4.28. É necessário que tenha sido usado antes um `with(plots)`*, a não ser que se queira escrever a* forma completa *dos nomes desses comandos que são* `plots[contourplot]` *e* `plots[contourplot3d]`*.*

Figura 4.27:

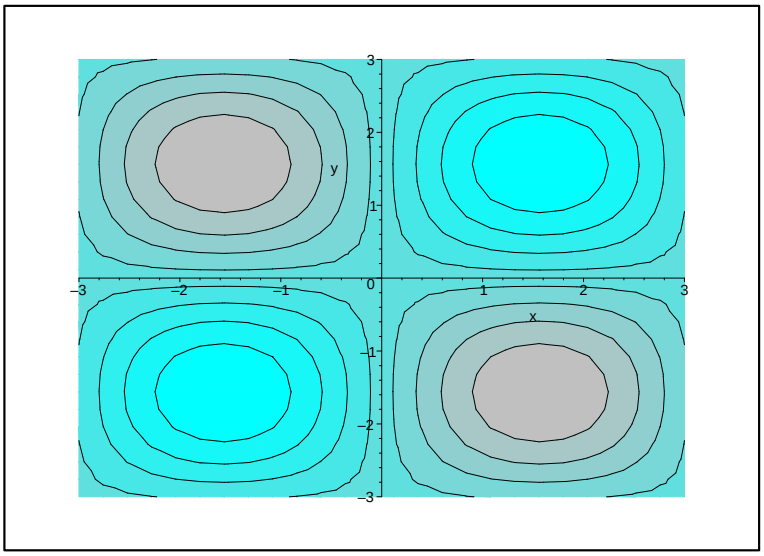
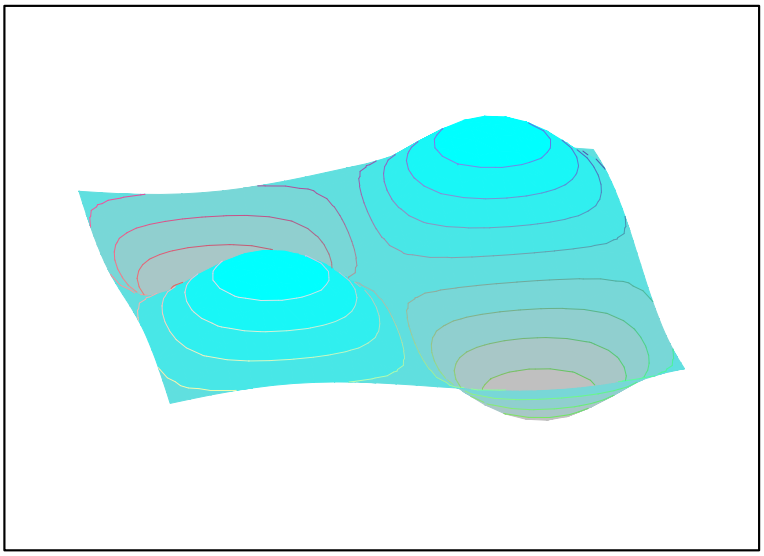


Figura 4.28:



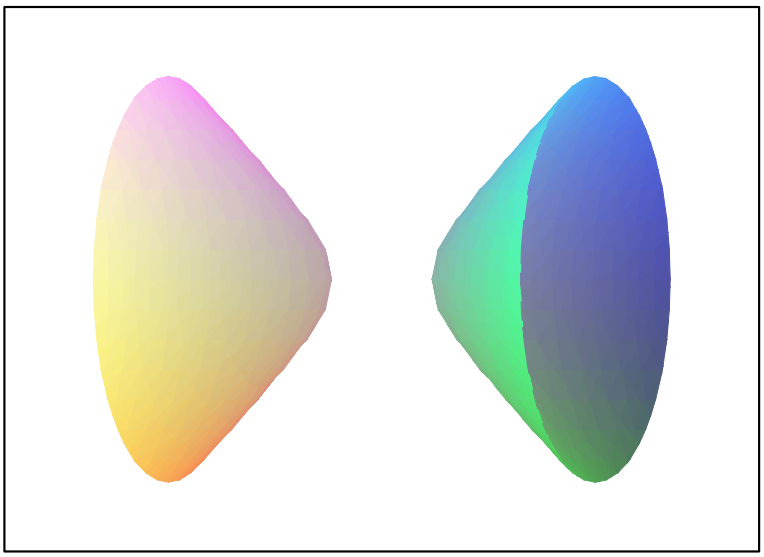
4.5.4 Gráficos definidos implicitamente

Gráficos de superfícies definidas implicitamente são muito comuns nos estudos de Cálculo ou Geometria Analítica. As primeiras superfícies estudadas são as quádricas cujas equações são dadas na forma implícita: $x^2+y^2+z^2=R^2$ (esfera), $\frac{x^2}{a^2}-\frac{y^2}{b^2}+\frac{z^2}{c^2}=1$ (hiperbolóide de uma folha), $\frac{x^2}{a^2}-\frac{y^2}{b^2}-\frac{z^2}{c^2}=1$ (hiperbolóide de duas folhas), etc. Esses tipo de gráfico pode ser facilmente construído pelo Maple com o comando `implicitplot3d` que faz parte do pacote `plots`. Seu uso é muito parecido com outros comandos já vistos anteriormente de forma que vamos nos limitar aqui a fazer apenas dois exemplos.

Exemplo 4.26 *Consideremos o hiperbolóide de duas folhas $x^2-y^2-z^2=1$, com x, y, z variando no intervalo $[-4,4]$. Vamos construir seu gráfico usando uma grade de dimensões $15\times15\times15$ (Figura 4.29). Optamos pelo estilo* `PATCHNOGRID` *com ângulos de rotações iniciais iguais a 110 e 90 graus.*

```
> with(plots):
> implicitplot3d( x^2 - y^2 - z^2 = 1, x=-4..4, y=-4..4, z=-4..4,
>          grid=[15,15,15], style=patchnogrid, orientation=[110, 90]);
```

Figura 4.29:



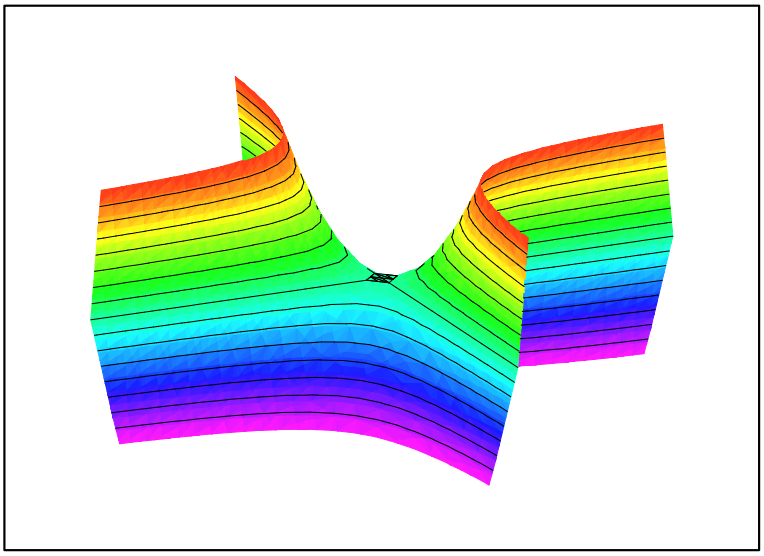
Exemplo 4.27 *Vamos construir o gráfico da sela $x^2-y^2=z$, colorindo o gráfico de acordo com a altura e no estilo que coloca as curvas de nível desenhadas no gráfico (Figura 4.30).*

```
> with(plots):
> implicitplot3d(x^2 - y^2 = z, x=-4..4, y=-4..4, z=-4..4, orientation =
>          [110,60], grid=[20,20,20], shading=zhue, style=patchcontour);
```

4.5.5 Gráficos de superfícies parametrizadas

Em algumas áreas de estudo como Geometria Diferencial e Computação Gráfica é muito comum o uso de superfícies definidas por equações paramétricas. Nestes casos, os

Figura 4.30:



pontos (x, y, z) que pertencem ao gráfico da superfície satisfazem equações do tipo

$$\begin{cases} x &= f(u,v) \\ y &= g(u,v) \\ z &= h(u,v) \end{cases}$$

com (u, v) pertencente a um domínio do no plano que depende de cada superfície.

Esse tipo de gráfico pode ser construído de modo semelhante aos construídos com o comando `plot3d` (seções 4.5.1 e 4.5.2), devendo-se apenas fornecer as equações paramétricas em forma de lista como no exemplo: `plot3d([f, g, h], x=a..b, y=c..d)`.

Cuidado: se usarmos chaves no lugar dos colchetes, como em `plot3d({f, g, h}, x=a..b, y=c..d)`, então serão construídos simultaneamente os gráficos das três funções $f(x, y)$, $g(x, y)$ e $h(x, y)$ que não têm nada a ver com a superfície parametrizada por $(f(x, y), g(x, y), h(x, y))$.

Exemplo 4.28 *Construamos o* helicóide $(u\cos v, u\,\mathrm{sen}\, v, v)$ *com* $-6 \leq u \leq 6$ *e* $-8 \leq v \leq 8$ *(Figura 4.31).*

```
> plot3d([u*cos(v), u*sin(v), v], u = -6..6, v = -8..8,
>                                 grid=[25, 50], shading=zhue);
```

Exemplo 4.29 *Neste exemplo, construímos uma superfície que tem a aparência de uma mola:*

$$F(u, v) = ((11 + 2\cos(u))\cos(v),\ (11 + 2\cos(u))\,\mathrm{sen}\,(v),\ 2\,\mathrm{sen}\,(u) + v)$$

com $0 \leq u \leq 2\pi$ *e* $-12 \leq v \leq 12$ *(Figura 4.32).*

```
> plot3d([(11+2*cos(u))*cos(v), (11+2*cos(u))*sin(v), 2*sin(u)+v],
>         u=0..2*Pi, v=-12..12, scaling=constrained, grid=[20,70],
>         orientation=[80,70]);
```

O pacote `plots` contém um comando `textplot3d` com sintaxe

`textplot3d([`*x, y, z, "mensagem"*`],` *opções*`)`

Figura 4.31:

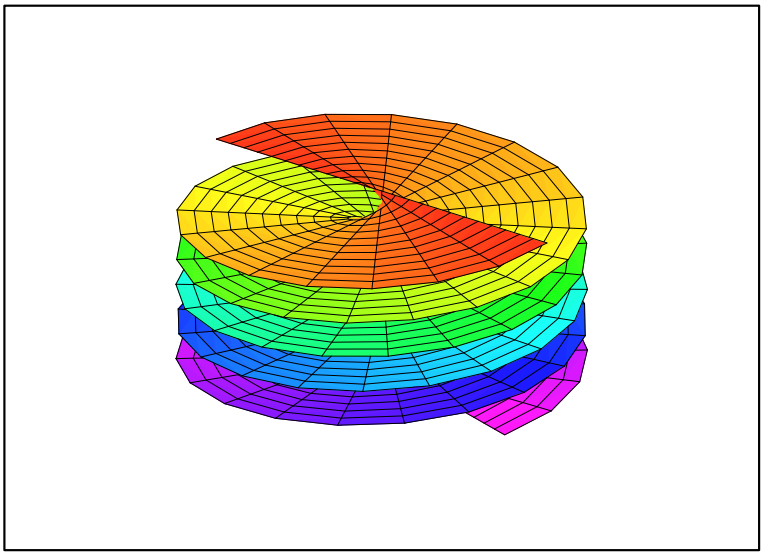
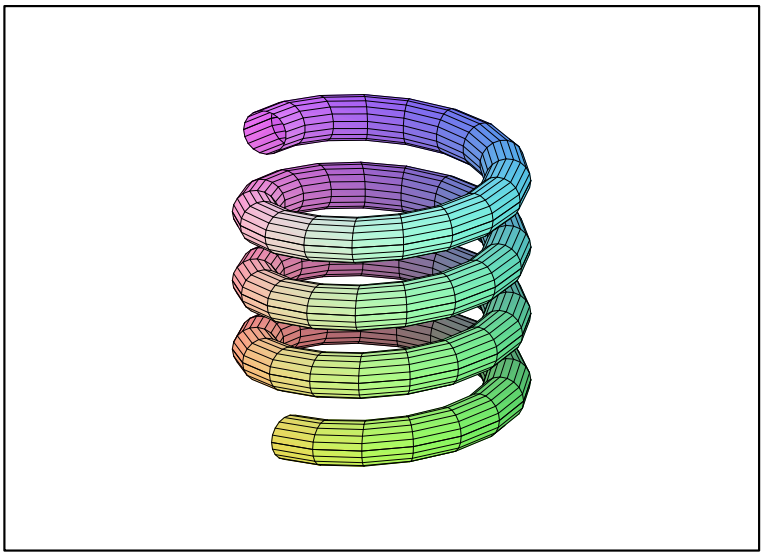


Figura 4.32:



cujo uso é semelhante ao `textplot` e é utilizado para imprimir mensagens em um sistema de eixos tridimensional.

Exemplo 4.30 *Construímos o gráfico do* hiperbolóide de uma folha *parametrizado por* $(\cos u - v \operatorname{sen} v,\ \operatorname{sen} u + v \cos u,\ v)$. *O gráfico não é mostrado na tela de imediato, pois ele é guardado em uma variável auxiliar chamada* `hiperb`.

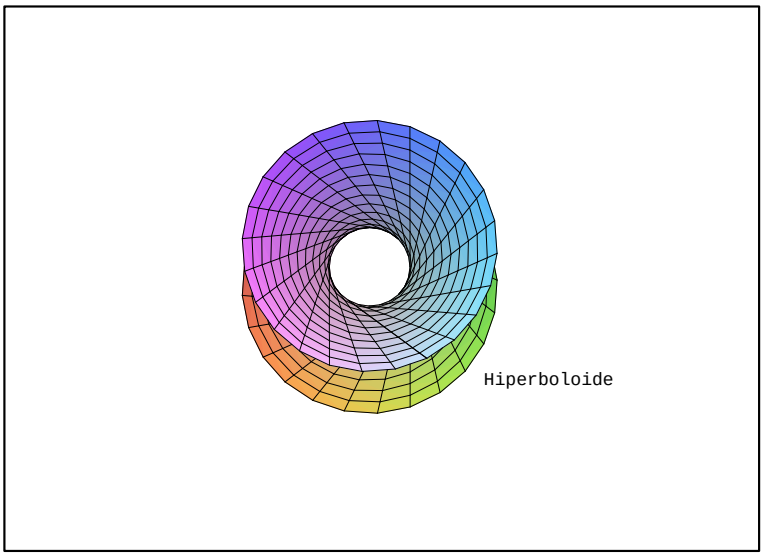
Depois, o `textplot3d` *é usado para imprimir a mensagem "Hiperboloide" à direita da posição* $(0, 4, 0)$ *e o resultado guardado na variável auxiliar* `mens`.

Finalmente o comando `display` *mostra na tela os conteúdos das variáveis auxiliares* `hiperb` *e* `mens`.

```
> hiperb := plot3d([cos(u) - v*sin(u), sin(u) + v*cos(u), v],
>          u=0..2*Pi, v=-3..3, scaling=constrained, orientation=[45,10]):
>
> with(plots):
```

```
> mens := textplot3d([0, 4, 0, "Hiperboloide"], align=RIGHT,
>                                          color=black, font=[COURIER, 15]):
> display(mens, hiperb);
```

Figura 4.33:



4.6 Outros gráficos

Além dos tipos mostrados nas seções anteriores, o Maple dispõe ainda de vários outros comandos e pacotes para construção de gráficos, por exemplo, para a construção de poliedros e outros tipos de objetos tridimensionais.

Destacamos nesta seção mais um comando do pacote `plots`: o `tubeplot` que pode ser usado para desenhar superfícies tubulares. Dada uma curva no espaço, o `tubeplot` constrói uma superfície que parece um cano seguindo o formato da curva dada. Em geral, gera gráficos muito curiosos.

O gráfico da curva $\alpha(t) = (f(t), g(t), h(t))$ com $a \leq t \leq b$ pode ser construído com um comando `spacecurve([f(t), g(t), h(t))], t=a..b)`.

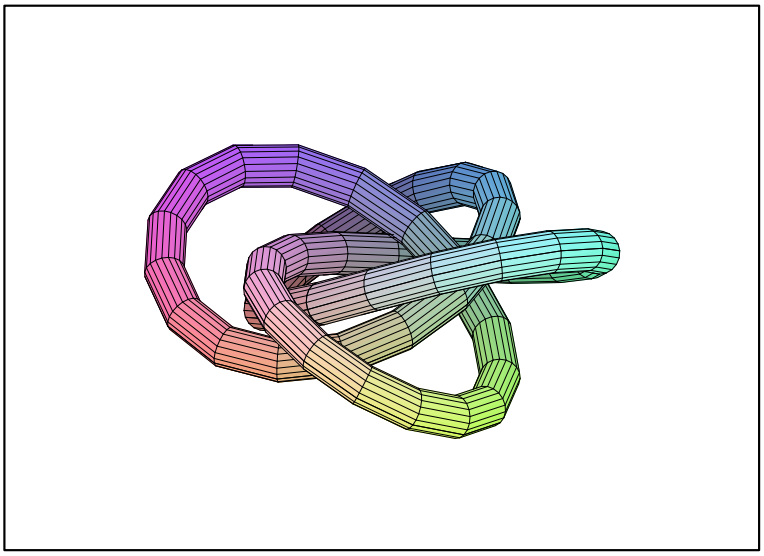
Exemplo 4.31 *Aqui, criamos um tubo de raio* $0,4$ *envolvendo a curva espacial parametrizada por*

$$f(t) = (-r\cos(at) + s\cos(bt),\ r\,\mathrm{sen}\,(at) + s\,\mathrm{sen}\,(bt),\ \mathrm{sen}\,(dt) + sen(ct)/2),$$

onde escolhemos $a = 1, b = 3$, $c = 2, d = 4$, $r = 4/3$ *e* $s = 3$.

```
> with(plots):
> a := 1: b := 3: c := 2: d := 4: r := 4/3: s := 3:
> curva := [-r*cos(a*t) + s*cos(b*t), r*sin(a*t) + s*sin(b*t),
>                                      sin(d*t) + sin(c*t)/2]:
> tubeplot(curva, radius = 0.4, t = 0..2*Pi, grid = [50, 20]);
```

Figura 4.34:



4.7 Animações gráficas

O pacote `plots` possui três comandos para a construção de animações: `animate`, `animate3d` e `animatecurve`. Nesta seção descrevemos brevemente cada um desses comandos e acrescentamos vários exemplos.

Uma animação consiste simplesmente em uma seqüência de gráficos, mostrados consecutivamente, um após o outro. Isso gera uma ilusão de movimento e normalmente chama muito a atenção. Fica parecendo que os gráficos se movem na tela.

4.7.1 Animações bidimensionais

A sintaxe do `animate` é

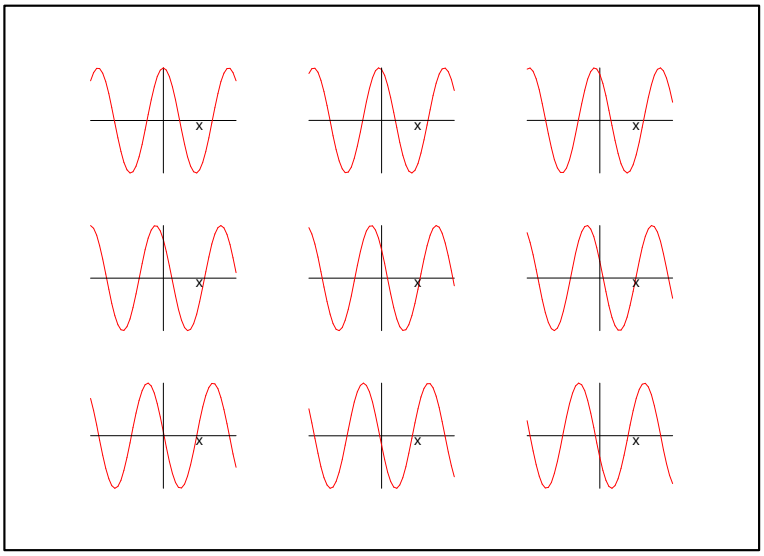
`animate`(*F(x, t), x = a..b, t = c..d, opções*)

onde $F(x,t)$ é uma expressão (ou função) de duas variáveis que nos nossos exemplos consideraremos sempre como x e t. O $F(x,t)$ deve ser seguido pela variação do x e do t e opcionalmente por opções (em forma de igualdades) que são as mesmas do comando `plot`. A variação do x corresponde ao domínio das funções envolvidas na animação, enquanto que a variação do t corresponde às posições intermediárias. O valor $t = c$ corresponde ao gráfico inicial e $t = d$ corresponde ao gráfico final. O total de n gráficos contruídos é controlado com uma opção `frames =` n.

Exemplo 4.32 *Neste exemplo criamos uma animação que corresponde a uma seqüência de 9 gráficos iniciando com* $y = cos(x)$ *e terminando com* $y = cos(x+2)$. *Essa seqüência corresponde aos gráficos de* $cos(x + t)$ *com* t *variando no conjunto* {0, 2/8, 4/8, 6/8, 1, 10/8, 12/8, 14/8, 2}, *assumindo assim 9 valores igualmente espaçados de 0 a 2 (Figura 4.35). Escolhendo o domínio como sendo o intervalo* $[-7, 7]$ *e gráficos sem marcas nos eixos* x *e* y (`tickmarks = [0, 0]`), *digitamos o seguinte comando no aviso do Maple:*

```
> with(plots):
> animate(cos(x + t), x =-7..7, t=0..2, frames=9, tickmarks=[0,0]);
```

Figura 4.35:



Para iniciar a animação, deve-se pressionar em cima do gráfico inicial com o botão esquerdo do mouse. *Com isso, a barra de contexto muda e é possível dar início à animação pressionando-se o botão "*Play the animation*". Neste exemplo, a animação dá a impressão de que o gráfico do cosseno está se deslocando para a esquerda.*

Outra maneira de iniciar a animação, é pressionar com o botão direito do mouse *em cima do gráfico inicial da animação. Com isso, deve aparecer um menu com as opções: Copy, Style, Legend, Axes, Projection, Animation e Export As. Escolhendo Animation aparece outro menu de opções: Play, Next, Backward, Faster, Slower, Continuous. Escolhendo Play terá início a animação na tela. A opção Continuous permite que a animação seja repetida indefinidamente – neste caso, para parar é só repetir esse procedimento que no lugar da opção Play aparecerá uma opção Stop.*

Ao pressionar com o botão direito do mouse em cima do gráfico inicial e escolhendo a opção "*Export As*" é possível salvar a animação em vários formatos, inclusive GIF animado.

Exemplo 4.33 *Para fazer $f(x)$ "transformar-se" em $g(x)$, basta usar no* `animate` *uma função $F(x,t) = (1-t)\,f(x) + t\,g(x)$ e $0 \leq t \leq 1$. Neste exemplo, "transformamos" a função $|x| - 1$ na função $-x^2 + 1$, usando um total de 16 gráficos (Figura 4.36).*

```
> with(plots):
> animate((abs(x) - 1)*(1 - t) + (-x^2 + 1)*t, x=-2..2, t=0..1,
>                                    frames = 16, tickmarks = [0, 0]);
```

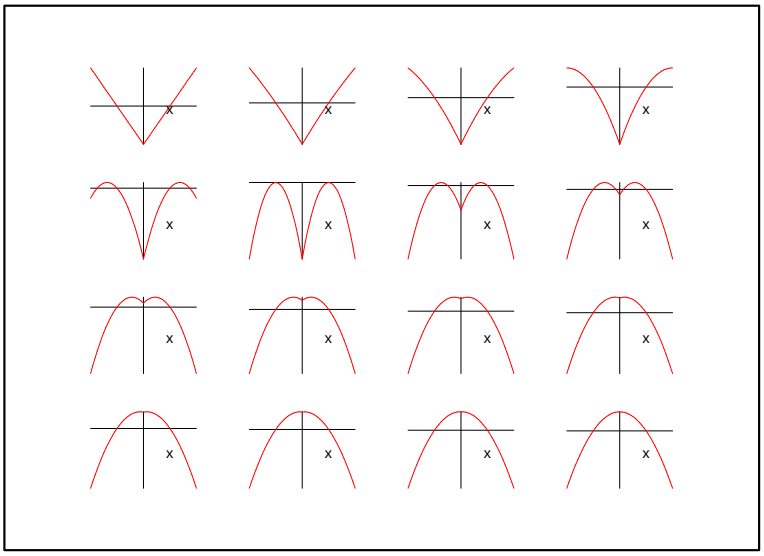
Como no exemplo anterior, é só escolher a opção Animation/Play ao pressionar com o botão direito do mouse em cima do gráfico inicial da animação.

4.7.2 Animações tridimensionais

O comando `animate3d` pode ser usado para gerar animações tridimensionais. Seu uso é semelhante ao do comando `animate`:

`animate3d`(*F(x, y, t), x = a..b, y = c..d, t = e..f, opções*)

Figura 4.36:



Exemplo 4.34 *O seguinte comando gera uma animação que inicia com o gráfico de* $\operatorname{sen}(x)\operatorname{sen}(y)$ *(que corresponde a* $t = 1$*) e termina com o gráfico de* $\operatorname{sen}(2x)\operatorname{sen}(2y)$ *(que corresponde a* $t = 2$*).*

```
> with(plots):
> animate3d( sin(x*t)*sin(y*t), x=-3..3, y=-3..3, t=1..2, frames=10,
>                                                  scaling=constrained);
```

Neste caso é usado um total de 10 gráficos na animação.

Com o `animate3d` é possível "deformar" o gráfico de uma superfície parametrizada por

$$X_1(u,v) = (f_1(u,v),\, g_1(u,v),\, h_1(u,v))$$

em outra superfície parametrizada por

$$X_2(u,v) = (f_2(u,v),\, g_2(u,v),\, h_2(u,v)).$$

Para isso, basta usar no `animate3d` uma função $F(u,v,t) = (1-t)X_1(u,v) + tX_2(u,v)$ $= ((1-t)f_1(u,v) + tf_2(u,v),\ (1-t)g_1(u,v) + tg_2(u,v),\ ((1-t)h_1(u,v) + th_2(u,v))$ com $0 \leq t \leq 1$.

Exemplo 4.35 *Para deformar* $(u, v, \operatorname{sen}(u^2+v^2))$ *em* $(u, v, 5-\cos(u))$ *com um total de 20 gráficos na animação e grade de tamanho* 30×30*, basta usar depois do* `with(plots)` *o seguinte comando:*

```
> animate3d([(1-t)*u+t*u, (1-t)*v+t*v, (1-t)*sin(u^2+v^2)+t*(5-cos(u))],
>                   u=-4..4, v=-4..4, t=0..1, frames=20, grid=[30,30]);
```

Com o `animate3d` é possível girar na tela o gráfico de qualquer superfície parametrizada por $X(u,v) = (f(u,v),\, g(u,v),\, h(u,v)$. Para isso, basta fazer o t variar no intervalo $[0, 2\pi]$ e usar uma função $F(u,v,t)$ nos seguintes formatos:

- $[f(u,v), g(u,v)\cos t - h(u,v)\operatorname{sen} t, g(u,v)\operatorname{sen} t + h(u,v)\cos t]$, para girar o gráfico em torno do eixo Ox

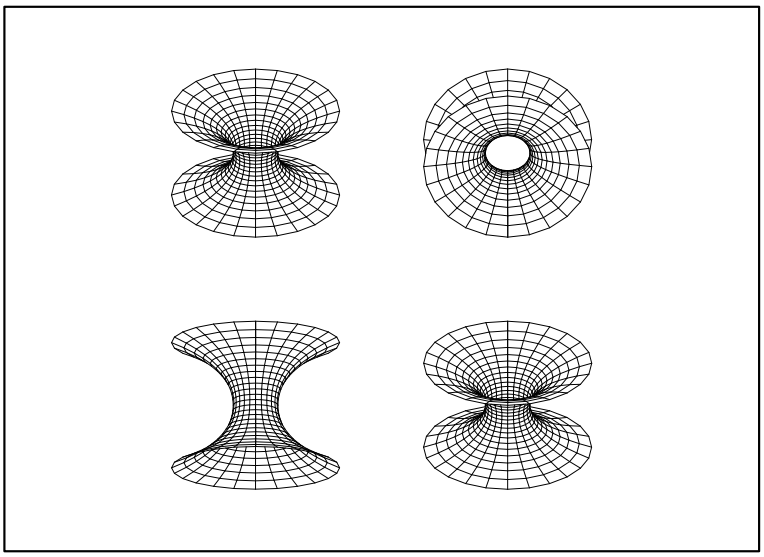
- $[f(u,v)\cos t - h(u,v)\operatorname{sen} t, g(u,v), f(u,v)\operatorname{sen} t + h(u,v)\cos t]$, para girar o gráfico em torno do eixo Oy
- $[f(u,v)\cos t - g(u,v)\operatorname{sen} t, f(u,v)\operatorname{sen} t + g(u,v)\cos t, h(u,v)]$, para girar o gráfico em torno do eixo Oz

Exemplo 4.36 *Vamos girar o gráfico de um* catenóide *em torno do eixo Oy:*

```
> f := (u, v) -> cosh(v)*cos(u);
> g := (u, v) -> cosh(v)*sin(u);
> h := (u, v) -> v;
> F := (u,v,t) ->
> [f(u,v)*cos(t)-h(u,v)*sin(t), g(u,v), f(u,v)*sin(t)+h(u,v)*cos(t)];
> teste:=plots[animate3d](F(u,v,t), u=-Pi..Pi, v=-2..2, t=0..2*Pi, frames=4,
>           orientation=[0,45],scaling=constrained,shading=none):
>
> display(teste);  # mostra todos os gráficos
> teste;           # executa a animação
```

A Figura 4.37 representa a animação. Ela foi construída com um `display(teste);`

Figura 4.37:



4.7.3 Outras animações

Com o comando `animatecurve` é possível observarmos a construção de de uma curva por etapas. Sua sintaxe é parecida com a do `plot`:

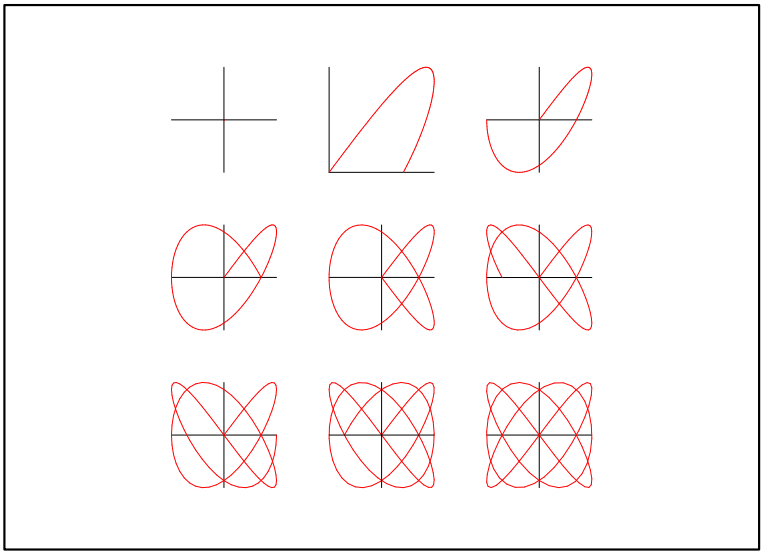
`animatecurve(`*f, domínio, opções,* `frames =` *n*`)`

onde f pode ser uma função de uma variável real ou uma lista de funções (neste caso, representando uma curva em forma paramétrica).

Exemplo 4.37 *Neste exemplo, construímos uma animação com a construção da curva definida parametricamente por* $f(t) = (\operatorname{sen}(3t), \operatorname{sen}(4t))$, *com* $0 \leq t \leq 2\pi$. *Esta animação está representada na Figura 4.38 (construída com um comando* `display`*).*

```
> with(plots):
> X := animatecurve([sin(3*t), sin(4*t), t=0..2*Pi], numpoints=100,
        thickness=2,frames=9,tickmarks=[0,0], scaling=constrained):
> X;
```

Figura 4.38:



4.8 Salvando gráficos em diferentes formatos

O Maple não só mostra gráficos na tela, mas também pode salvá-los em disco nos mais diferentes formatos: EPS, GIF, BMP, JPG e outros.

A maneira mais fácil de fazer isso é pressionar com o botão direito do mouse em cima do gráfico e escolher depois a opção "Export As", escolher o formato desejado e digitar o nome do arquivo a ser criado.

Mas, é possível também fazer isso de forma mais eficiente, com uma maior variedade de opções com comandos específicos do Maple. Para isso, basta usar um comando `plotsetup`:

```
plotsetup(plotdevice, plotoutput, plotoptions)
```

onde `plotoutput` corresponde ao nome do arquivo de saída (por exemplo, `"A:\\teste.gif"` ou `"C:\\temp\\maple\\grafico.jpg"`) e `plotdevice` pode ser:

ps para uma saída em um arquivo Encapsulated PostScript (EPS). Neste caso, `plotoptions` pode incluir as opções: `color, noborder, portrait, landscape, width, height, bottommargin, leftmargin, axiswidth, axisheight.`

gif para a saída ser em forma de imagem GIF. Neste caso, `plotoptions` pode incluir valores para a altura, largura e número de cores como no exemplo: `plotoptions="colors=256, height=200, width=400"`

jpeg para a saída ser em forma de arquivo JPG. Neste caso, `plotoptions` pode incluir informações sobre a altura e largura da imagem, como em `plotoptions="height=200,width=320"`

bmp para a saída ser em forma de arquivo BMP (imagem *bitmap true color* do Windows)

pov para a saída ser na forma de arquivo POV do POVRay. Neste caso, `plotoptions` pode incluir informações sobre ângulos de rotações como no exemplo `plotoptions="phi=20,theta=30"`.

Exemplo 4.38 *Com o comando a seguir, os gráficos gerados pelo Maple não vão ser mostrados na tela, mas vão ser gravados em disco em um arquivo (imagem digital) de extensão GIF:*

```
> plotsetup(gif, plotoutput="c:\\graf\\teste.gif", plotoptions="width=300");
```

Exemplo 4.39 *Aqui é gerado um arquivo EPS, ideal para ser inserido em textos em LaTeX:*

```
> plotsetup(ps,plotoutput="plot.eps",plotoptions="color,portrait,noborder");
```

Para fazer com que os gráficos voltem a ser mostrados na tela novamente, basta digitar:

```
> plotsetup(default);
```

Digite `?plotsetup` ou `?plot,device` para mais informações.

4.9 Exercícios

1) Construa os gráficos das seguintes funções $y = f(x)$:
   a) $y = x^2 \operatorname{sen}(2/x)$, $-0,01 \leq x \leq 0,01$, $-0,0001 \leq y \leq 0,0001$;
   b) $y = \displaystyle\sum_{k=1}^{n} \frac{\operatorname{sen}((2k-1)x)}{2k-1}$ com $n = 7$, $-10 \leq x \leq 10$, $-1 \leq y \leq 1$;
   c) $y = \{1/x\}$ = parte fracionária de $1/x$, $-1 \leq x \leq 1$, $-1 \leq y \leq 1$;
   d) $y = \operatorname{tg}(\operatorname{sen}(x)) - \operatorname{sen}(\operatorname{tg}(x))$ com $-2\pi \leq x \leq 2\pi$.

2) Construa os gráficos das curvas definidas implicitamente pelas equações:
   a) $x^{2/3} + y^{2/3} = 1$, $-1 \leq x \leq 1$, $-1 \leq y \leq 1$;
   b) $(x^2 + y^2)^2 = x^2 - y^2$, $-1 \leq x \leq 1$, $-1 \leq y \leq 1$.

3) Construa o gráfico da *"borboleta"* cujas equações paramétricas são $x = f(t)\operatorname{sen}(t)$, $y = f(t)\cos(t)$, onde $f(t) = e^{\cos(t)} - 2\cos(4t) + \operatorname{sen}^5(t/12)$ e $t \in [0,\, 50]$.

4) Construa os seguintes gráficos em coordenadas polares $r = f(\theta)$. Use as opções `scaling=constrained`, `numpoints=400` e `axes=none`.
   a) $r = \theta - 4\cos(9\pi\theta)$, $0 \leq \theta \leq 15\pi$;
   b) $r = \theta + \operatorname{sen}(2\pi\theta) + 5\cos(2\pi\theta)$, $0 \leq \theta \leq 40$;
   c) $r = \sqrt{\theta}\operatorname{sen}(\theta) - \cos(17\theta)$, $0 \leq \theta \leq 15\pi$;

d) $r = (\theta + 9)/2 - 3\,\mathrm{sen}\,(11\,\theta) + 5\cos(4\theta),\ \ 0 \le \theta \le 40$;
e) $r = \theta\cos(8\theta),\ \ 0 \le \theta \le 20\pi$.

5) As curvas definidas pelas equações paramétricas

$$x = (a+b)\cos t - R\cos\left(\frac{t(a+b)}{b}\right),$$

$$y = (a+b)\,\mathrm{sen}\,t - R\,\mathrm{sen}\,\left(\frac{t(a+b)}{b}\right)$$

são chamadas *epiciclóides*. Construa o gráfico da epiciclóide em que $a = 11$, $b = 4$ e $R = 7/2$.

6) Construa os gráficos das seguintes curvas em que $0 \le t \le 2\pi$:
a) $x = (\,\mathrm{sen}\,t)^3$, $y = (\cos t)^3$;
b) $x = \,\mathrm{sen}\,(4t) + \cos(5t)$, $y = (\,\mathrm{sen}\,(3t))^3$;
c) $x = \,\mathrm{sen}\,t + \,\mathrm{sen}\,(3t)$, $y = \cos(5t)$.

7) Construa o gráfico das funções $z = f(x,y),\ \ -3 \le x \le 3,\ \ -3 \le y \le 3$.
a) $z = (\frac{1}{2}\cos x\cos y - \cos(x^2+y^2))e^{-x^2-y^2}$;
b) $z = \dfrac{8\,\mathrm{sen}\,(xy^2 - yx^2)}{1 + x^2 + y^2}$;
c) $z = \dfrac{10(x^2-y^2)}{2 + x^4 + y^4}$
d) $z = \mathrm{Re}((x+iy)^n)$ com $n = 5$

8) Construa os gráficos das superfícies parametrizadas por:
a) $F(u,v) = ((3 - v\,\mathrm{sen}\,(u/2))\,\mathrm{sen}\,(u), (3 - v\,\mathrm{sen}\,(u/2))\cos(u), v\cos(u/2))$ (*faixa de Möbius*), $0 \le u \le 4\pi$, $-1 \le v \le 1$;
b) $F(u,v) = ((4+(2+\cos(v))\cos(5u) - \,\mathrm{sen}\,(v)\,\mathrm{sen}\,(5u))\cos(u), (4+(2+\cos(v))\cos(5u) - \,\mathrm{sen}\,(v)\,\mathrm{sen}\,(5u))\,\mathrm{sen}\,(u), (2+\cos(v))\,\mathrm{sen}\,(5u) + \,\mathrm{sen}\,(v)\cos(5u))$, $-\pi \le u \le \pi$, $-\pi \le v \le \pi$;
c) $F(u,v) = (u\,\mathrm{sen}\,(u)\cos(v), u\cos(u)\cos(v), -u\,\mathrm{sen}\,(v))$, $0 \le u \le 3\pi$, $0 \le v \le 2\pi$;
d) $F(u,v) = (u - \frac{u^3}{3} + uv^2, v - \frac{v^3}{3} + u^2v, u^2 - v^2)$, $-2 \le u \le 2$, $-2 \le v \le 2$.

9) Use um comando `tubeplot` com opções `numpoints = 300` e `radius = 3` para traçar o gráfico de

$$f(t) = (t\cos t,\ t\,\mathrm{sen}\,t,\ t), \quad -10\pi \le t \le 10\pi.$$

10) Use o comando `spacecurve` para traçar o gráfico da curva $f(t)$, $0 \le t \le 2\pi$, e o `tubeplot` com opções `radius=10`, `tubepoints=12` e `numpoints=100` em cada um dos seguintes casos:
a) $f(t) = (-22\cos(t) - 128\,\mathrm{sen}\,(t) - 44\cos(3t) - 78\,\mathrm{sen}\,(3t),\ 11\cos(t) - 43\cos(3t) + 34\cos(5t) - 39\,\mathrm{sen}\,(5t),\ 70\cos(3t) - 40\,\mathrm{sen}\,(3t) + 8\cos(5t) - 9\,\mathrm{sen}\,(5t))$;
b) $f(t) = (32\cos(t) - 51\,\mathrm{sen}\,(t) - 104\cos(2t) - 34\,\mathrm{sen}\,(2t) + 104\cos(3t) - 91\,\mathrm{sen}\,(3t),\ 94\cos(t) + 41\,\mathrm{sen}\,(t) + 113\cos(2t) - 68\cos(3t) - 124\,\mathrm{sen}\,(3t),\ 16\cos(t) + 73\,\mathrm{sen}\,(t) - 211\cos(2t) - 39\,\mathrm{sen}\,(2t) - 99\cos(3t) - 21\,\mathrm{sen}\,(3t))$;
c) $f(t) = (-22\cos(t) - 128\,\mathrm{sen}\,(t) - 44\cos(3t) - 78\,\mathrm{sen}\,(3t),\ -10\cos(2t) - 27\,\mathrm{sen}\,(2t) + 38\cos(4t) + 46\,\mathrm{sen}\,(4t),\ 70\cos(3t) - 40\,\mathrm{sen}\,(3t))$.

Capítulo 5

Cálculo Diferencial e Integral

Este capítulo pressupõe que o leitor está familiarizado com conceitos de Cálculo tais como limites, derivadas, integrais e séries. A maior parte desses assuntos é vista em cursos de Cálculo 1, com uma pequena parte vista em Cálculo 2 ou Cálculo 3.

5.1 Limites

5.1.1 Limites de funções $f(x)$

No Maple, o limite de uma função quando a variável tende a um certo valor é calculado com um comando do tipo `limit(`*função, variável = valor*`)`. Por exemplo, $\lim_{t\to a} f(t)$ é calculado com um comando `limit(f(t), t = a)`.

Exemplo 5.1 *Os comandos a seguir podem ser usados para calcular os limites:* $R = \lim_{x\to -1}(x^2 - 5x + 3)$, $S = \lim_{t\to 0} \dfrac{\sqrt[5]{1+t} - \sqrt[5]{1-t}}{\sqrt[3]{1+t} - \sqrt[3]{1-t}}$, $T = \lim_{\theta\to \pi/2}\left(\dfrac{\ln(\cos(\theta))}{\ln(\tan(\theta))}\right)$ *e* $U = \lim_{\alpha\to 0} \dfrac{\cos(m\alpha) - \cos(n\alpha)}{\alpha^2}$.

```
> R := limit (x^2 - 5*x + 3, x = -1);
```

$$R := 9$$

```
> S := limit( ( root[5](1 + t) - root[5](1 - t))/
>                            ( root[3](1+t)-root[3](1-t)),t=0);
```

$$S := \frac{3}{5}$$

```
> T := limit ( ln(cos(theta))/ln(tan(theta)), theta = Pi/2);
```

$$T := -1$$

```
> U := limit( (cos(m*alpha) - cos(n*alpha))/alpha^2, alpha=0);
```

$$U := -\frac{1}{2}m^2 + \frac{1}{2}n^2$$

Exemplo 5.2 *Vamos calcular o limite* $\lim_{x\to 0}(x+1)^{\mathrm{cotg}(x)}$. *Inicialmente, o comando é digitado com erro, mas parece funcionar.*

```
> limit((x + 1)^cotg(x), x=0); # errado
```

$$1$$

A resposta assim obtida está **errada** *porque a função* `cotg(x)` *não é uma função pré-definida do programa. O que ele fez foi calcular a potência* $1^{cotg(x)}$ *onde* `cotg(x)` *é um considerado como sendo um símbolo qualquer. A função cotagente para o Maple é* `cot(x)`*, conforme digitado a seguir.*

```
> limit((x+1)^cot(x), x=0);
```

$$e$$

Obtemos assim a resposta correta: $\lim_{x\to 0}(x+1)^{\mathrm{cotg}(x)} = e.$

O Maple possui outro comando chamado `Limit` (iniciando com L maiúsculo), com sintaxe idêntica à de `limit`, cujo objetivo é mostrar o limite em formato usual, sem calculá-lo. É chamada *forma inercial* ou *forma inerte* ou *forma não avaliada* do limite.

Exemplo 5.3 *Neste exemplo mostramos um limite na sua forma inercial. O valor de uma expressão desse tipo pode ser calculado se for aplicado um comando* `value` *à expressão.*

```
> Limit (( x - a)/(sqrt(x) - sqrt(a)), x = a);
```

$$\lim_{x\to a}\frac{x-a}{\sqrt{x}-\sqrt{a}}$$

```
> value(%);
```

$$2\sqrt{a}$$

O uso do `Limit` combinado com o `limit` melhoram a apresentação dos resultados, conforme mostramos nos seguintes exemplos.

Exemplo 5.4

```
> Limit(sin(a*x)/(b*x), x = 0) = limit( sin(a*x)/(b*x), x = 0);
```

$$\lim_{x\to 0}\frac{\sin(ax)}{bx} = \frac{a}{b}$$

Exemplo 5.5

```
> F := (sin(3*x) - 3*x*exp(x) + 3*x^2)/(arctan(x) - sin(x) - x^3/6):
> L := Limit (F, x = 0):
> l := limit (F, x = 0):
> L = l;
```

$$\lim_{x\to 0}\frac{\sin(3x)-3xe^x+3x^2}{\arctan(x)-\sin(x)-\frac{1}{6}x^3} = 18$$

5.1.2 Limites no infinito

Os limites no infinito, isto é, com a variável tendendo a $+\infty$ ou a $-\infty$, são calculados usando-se um valor `-infinity` ou `infinity` para a variável.

Exemplo 5.6 *Vamos calcular* $\lim_{x\to\infty}(3+x-4x^3+x^4)$.

```
> limit ( 3 + x - 4*x^3 + x^4, x = -infinity);
```

$$\infty$$

Exemplo 5.7 *Calcular* $\lim_{x\to-\infty}\dfrac{5x^7-3x^4+2x^2+11}{3x^7+x^6-x^3+10x^2+4}$.

```
> F := (5*x^7-3*x^4+2*x^2+11)/(3*x^7+x^6-x^3+10*x^2+4):
> lim := limit(F, x=-infinity):
> Lim :=  Limit(F, x=-infinity):
> Lim = lim;
```

$$\lim_{x\to(-\infty)}\frac{5x^7-3x^4+2x^2+11}{3x^7+x^6-x^3+10x^2+4}=\frac{5}{3}$$

Exemplo 5.8 *O Maple conhece a regra que afirma que no produto de duas funções, se uma tender a zero e a outra for limitada, então o limite do produto dessas funções é zero. Exemplificamos isso a seguir.*

```
> limit ( 1/x * cos(x), x = infinity);
```

$$0$$

Mas, se uma tender a zero e a outra função não for limitada então nada podemos afirmar a respeito do limite do produto dessas funções. Aqui, quando $x\to\infty$ *temos que* $1/x$ *tende a zero mas o produto* $1/x\,.\,\cosh(x)$ *tende a infinito. Isso acontece porque a função* $\cosh(x)$ *não é limitada.*

```
> limit( 1/x * cosh(x), x = infinity);
```

$$\infty$$

Exemplo 5.9 *Aqui usamos mais uma vez a forma inercial do limite para obtermos uma igualdade onde aparece o limite proposto seguido do seu valor calculado.*

```
> Limit(((x+7)/(x+3))^x,x=infinity):  % = value(%);
```

$$\lim_{x\to\infty}\left(\frac{x+7}{x+3}\right)^x = e^4$$

5.1.3 Limites laterais

Limites laterais são calculados acrescentando-se uma opção `left` ou `right` aos comandos `limit` ou `Limit`. Se for acrescentada a opção `left` então será calculado o limite lateral à esquerda. Se for acrescentado `right` então o limite será lateral à direita.

Exemplo 5.10 *Neste exemplo calculamos os limites à esquerda e à direita do 0 da função* $\cos(x)/x$. *Como esses limites não são iguais, concluímos observando que o limite em* $x=0$ *é indefinido, ou melhor, não existe.*

```
> Limit(cos(x)/x , x=0, left) = limit (cos(x)/x, x =0 , left);
```

$$\lim_{x\to 0-} \frac{\cos(x)}{x} = -\infty$$

```
> Limit(cos(x)/x, x=0, right) = limit (cos(x)/x, x=0, right);
```

$$\lim_{x\to 0+} \frac{\cos(x)}{x} = \infty$$

```
> limit( cos(x)/x, x=0);
```

$$undefined$$

Exemplo 5.11 *Definamos agora uma função contínua por partes com o* `piecewise`*:*

```
> y := piecewise(x < 0, sin(x), x >=0 and x < 2, 2*x - 1, x >= 2, 2);
```

$$y := \begin{cases} \sin(x) & x < 0 \\ 2x - 1 & -x <= 0 \text{ and } x < 2 \\ 2 & 2 <= x \end{cases}$$

Calculemos agora os limites nos pontos de descontinuidade. É recomendável acompanhar cada resposta obtida no gráfico da Figura 5.1.

```
> limit(y, x = 0, left);
```

$$0$$

```
> limit(y, x = 0, right);
```

$$-1$$

```
> limit(y, x = 0);
```

$$undefined$$

```
> limit(y, x = 2, left);
```

$$3$$

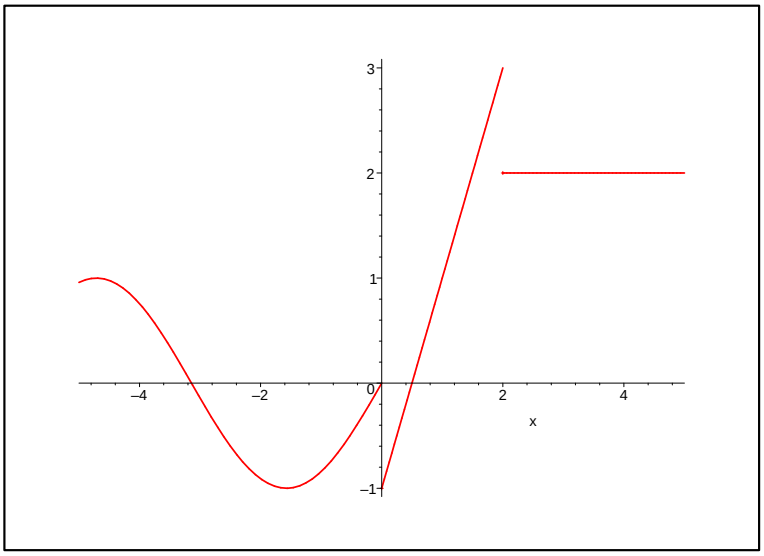
```
> limit(y, x = 2, right);
```

$$2$$

```
> limit(y, x = 2);
```

$$undefined$$

Figura 5.1:



5.1.4 Limites de funções de várias variáveis

Apesar de muito eficiente no cálculo de limites de funções de uma variável, o Maple mostra-se ineficiente no cálculo de limites de funções de várias variáveis. Ele só consegue calcular esse tipo de limite em pouquíssimos casos.

A sintaxe para o cálculo desse tipo de limite é idêntica à do caso de uma única variável, tendo-se o cuidado de fornecer os valores das variáveis em forma de conjunto:

$$\text{limit}\ (\ \textit{função}, \{\ \textit{variável1} = \textit{valor1}, \textit{variável2} = \textit{valor2}, \ldots\})$$

Exemplo 5.12 *Consideremos o limite* $\lim\limits_{(x,y)\to(0,0)} \dfrac{x^2-y^2}{x^2+y^2}$. *É um dos poucos casos que o Maple consegue resolver. A resposta é* undefined, *ou seja, o limite não existe.*

```
> limit( (x^2 - y^2)/(x^2 + y^2), {x = 0, y = 0});
```

$$undefined$$

Exemplo 5.13 *Neste exemplo, estudamos o limite* $\lim\limits_{(x,y)\to(0,0)} \dfrac{xy}{x^2+y^2}$. *O Maple devolve como resposta o próprio limite proposto, ou seja, ele não consegue decidir nada. Mas, o usuário pode dar uma ajuda ao programa fornecendo o limite de forma iterada limit(limit(...)) e indicando algum tipo de dependência entre as variáveis. Se fizermos isso mais de uma vez e obtivermos respostas diferentes, então o limite não existirá.*

```
> limit(x*y/(x^2 + y^2), {x = 0, y = 0});
```

$$\text{limit}\left(\frac{xy}{x^2+y^2}, \{x=0, y=0\}\right)$$

```
> Limit(Limit ( x*y/(x^2 + y^2), x = y), y = 0);
```

$$\lim_{y\to 0}\lim_{x\to y}\frac{xy}{x^2+y^2}$$

```
> limit(limit ( x*y/(x^2 + y^2), x = y), y = 0);
```

$$\frac{1}{2}$$

```
> limit(limit ( x*y/(x^2 + y^2), x = 2*y), y = 0);
```

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

Concluímos então que o limite dado não existe pois obtivemos as respostas $1/2$ *e* $2/5$ *quando fizemos* $x = y$ *e* $x = 2y$*, respectivamente.*

5.2 Funções contínuas

O Maple possui três comandos relacionados com a continuidade de uma função:

discont(f(x), x) Calcula os pontos de descontinuidade da função definida pela expressão algébrica $f(x)$ na variável x.

fdiscont(f(x), domínio, opções) Calcula aproximadamente, usando métodos numéricos, as descontinuidades de $f(x)$ no domínio indicado. Podem ser feitas diversas opções, como escolher o nome do método numérico utilizado.

iscont(f(x), x=a..b, opção) Função *booleana* que testa a continuidade de $f(x)$ no intervalo indicado. A opção pode decidir se o intervalo é aberto ('open') ou fechado ('closed') – os apóstrofos aqui utilizados são opcionais.

Exemplo 5.14 *Vamos testar a descontinuidade de* $f(x) = \dfrac{1}{x} + \dfrac{1}{x+5}$ *em cada intervalo dado.*

```
> iscont(1/x + 1/(x + 5), x = -10..1); # descontínua em ]-10, 1[
```

$$false$$

```
> iscont(1/x + 1/(x + 5), x = 0..1); # contínua em ]0, 1[
```

$$true$$

```
> iscont(1/x + 1/(x + 5), x = 0..1, closed); # descontínua em [0, 1]
```

$$false$$

```
> iscont(1/x + 1/(x + 5), x = -10..-6); # contínua em ]-10, -6[
```

$$true$$

```
> iscont(1/x + 1/(x + 5), x = -10..-5, open); # contínua em ]-10, -5[
```

$$true$$

```
> iscont(1/x + 1/(x + 5), x=-10..-5, closed); # descontínua em [-10,-5]
```

$$false$$

Exemplo 5.15 *O* `discont` *é bastante eficiente para achar as descontinuidades de funções não muito complicadas. Ele fornece um conjunto com todas as descontinuidades encontradas. Inicialmente usamos esse comando para determinar as descontinuidades de* $f(x) = 1/x + 1/(x + 5)$*.*

```
> discont( 1/x + 1/(x + 5), x);
```

$$\{-5,\, 0\}$$

Agora, vamos determinar as descontinuidades de $g(x) = \dfrac{x^5 + x - 2}{x^3 - 5x^2 + 6x}$.

```
> discont((x^5 + x - 2)/(x^3 - 5*x^2 + 6*x), x);
```

$$\{0,\, 2,\, 3\}$$

Exemplo 5.16 *Vamos determinar as descontinuidades da função* $\mathrm{tg}(3x)$. *O Maple representa inteiros genéricos por* _Z1~, _Z2~, *etc. Em livros de Cálculo, o usual seria apresentar a resposta dada na forma* $\frac{1}{3}k\pi + \frac{1}{6}\pi$, *onde* $k \in \mathbb{Z}$.

```
> discont(tan(3*x), x);
```

$$\left\{\frac{1}{3}\pi \_Z1\sim + \frac{1}{6}\pi\right\}$$

O comando `about` *fornece informações sobre o* _Z1, *confirmando que ele é um número inteiro.*

```
> about(_Z1);
```

Originally _Z1, *renamed* _Z1~: *is assumed to be: integer*

E agora vamos determinar as descontinuidades de $\dfrac{1 + x + e^x}{1 + 3\cos(x)}$

```
> discont( (1 + x + exp(x))/(1 + 3*cos(x)), x);
```

$$\left\{\pi - \arccos(\frac{1}{3}) - 2\_B1\sim\, \pi + 2\_B1\sim\, \arccos(\frac{1}{3}) + 2\pi \_Z2\sim\right\}$$

O comando `about` *fornece informações sobre* _Z2 *e* _B1. *Neste caso, o* _Z2~ *é um número inteiro e o* _B1~ *só pode assumir dois valores:* 0 *ou* 1.

```
> about(_Z2, _B1);
```

Originally _Z2, *renamed* _Z1~: *is assumed to be: integer*
Originally _B1, *renamed* _B1~: *is assumed to be: OrProp(0,1)*

Exemplo 5.17 *Usamos o* `fdiscont` *para determinar de forma aproximada as descontinuidades das funções* $f(x) = \dfrac{1}{e^x + \mathrm{tg}\, x}$ *no intervalo* $[-4,\, 3]$ *e* $g(x) = \dfrac{x^2 + x + 1}{x^7 - 3x + 10}$ *no intervalo* $[-20,\, 20]$. *Utilizamos o método de Newton para calcular cada resposta.*

```
> fdiscont( 1/(exp(x) + tan(x)), x = -4..3, newton=true);
```

$$[-3.18302881311210760,\, -.531390856652157129,\, 1.74388634680748878]$$

```
> fdiscont((x^2 + x + 1)/(x^7 - 3*x + 10), x = -20..20, newton=true);
```

$$[-1.46366450027293738]$$

5.3 Derivadas

5.3.1 Derivadas de funções de uma variável

A derivada de uma função definida por uma expressão algébrica $f(x)$ na variável x é calculada com um comando `diff(`*f(x), x*`)`.

Exemplo 5.18 *Sejam f, g, h as funções definidas por $f(x) = 3x^4 + x^2 + 3$, $g(t) = \sqrt{t+1} + \sqrt[3]{t-1}$ e $h(w) = w^2 + 2^w$. Vamos calcular $f'(x)$, $g'(t)$ e $h'(w)$.*

```
> f(x) := 3*x^4 + x^2 + 3;
```

$$f(x) := 3x^4 + x^2 + 3$$

```
> diff(f(x), x);   # derivada de f(x) com relação a x
```

$$12x^3 + 2x$$

```
> g(t) := sqrt(t + 1) + root[3](t - 1);
```

$$g(t) := \sqrt{t+1} + (t-1)^{\left(\frac{1}{3}\right)}$$

```
> diff(g(t), t);   # derivada de g(t) com relação a t
```

$$\frac{1}{2}\frac{1}{\sqrt{t+1}} + \frac{\frac{1}{3}}{(t-1)^{\left(\frac{2}{3}\right)}}$$

```
> h(w) := w^2 + 2^w;
```

$$h(w) := w^2 + 2^w$$

```
> diff(h(w), w);   # derivada de h(w) com relação a w
```

$$2w + 2^w \ln(2)$$

Exemplo 5.19 *É óbvio que o Maple conhece as regras de derivação como a derivada do produto, derivada do quociente e a Regra da Cadeia, conforme exemplificamos a seguir.*

```
> diff(x^5 * cos(x), x);            # derivada de um produto
```

$$5x^4\cos(x) - x^5\sin(x)$$

```
> diff(cos(cos(ln(x))), x);         # Regra da Cadeia
```

$$\frac{\sin(\cos(\ln(x)))\sin(\ln(x))}{x}$$

```
> diff(exp(4*x + 1)/(x^2 + 1), x); # derivada de um quociente
```

$$4\frac{e^{(4x+1)}}{1+x^2} - \frac{2e^{(4x+1)}x}{(1+x^2)^2}$$

```
> simplify(%);
```

$$2\frac{e^{(4x+1)}(2+2x^2-x)}{(1+x^2)^2}$$

Exemplo 5.20 *Definimos agora uma função f e tentamos calcular sua derivada com o comando* `diff`*. Note que* `diff(`*f, x*`)` *não funciona, pois o* `diff` *não calcula derivadas de funções. Mas,* `diff(`*f(x), x*`)` *funciona da maneira que se espera, porque $f(x)$ é considerado pelo Maple como sendo uma expressão algébrica.*

```
> f:= x-> x^3 + sin(x);
```

$$f := x \to x^3 + \sin(x)$$

```
> diff(f, x); # não funciona para calcular f´(x)
```

$$0$$

```
> diff(f(x), x); # agora, sim, calcula f´(x)
```

$$3x^2 + \cos(x)$$

Exemplo 5.21 *A resposta obtida com o* `diff` *nem sempre é a mais simples. Neste exemplo, o* `simplify` *faz uma simplificação bastante significativa no resultado fornecido pelo* `diff`*.*

```
> y := x *sqrt(x^2 + 1)/2 + 1/2*ln(x + sqrt(x^2 +1));
```

$$y := \frac{1}{2}x\sqrt{x^2+1} + \frac{1}{2}\ln(x + \sqrt{x^2+1})$$

```
> diff(y, x);
```

$$\frac{1}{2}\sqrt{x^2+1} + \frac{\frac{1}{2}x^2}{\sqrt{x^2+1}} + \frac{\frac{1}{2}\left(1 + \frac{x}{\sqrt{x^2+1}}\right)}{x + \sqrt{x^2+1}}$$

```
> simplify(%);
```

$$\sqrt{x^2+1}$$

Exemplo 5.22 *Se o Maple não conhece a derivada da função, então ele "deixa-a indicada", conforme exemplificamos aqui com as derivadas das funções* $Q(x)$ *e* $R(x)$*).*

```
> P(x) := (x^2 + x + 1)*Q(x) + R(x);
```

$$P(x) := (x^2 + x + 1)Q(x) + R(x)$$

```
> diff(P(x), x);
```

$$(2x+1)Q(x) + (x^2+x+1)\left(\frac{\partial}{\partial x}Q(x)\right) + \left(\frac{\partial}{\partial x}R(x)\right)$$

Nesses casos o Maple usa a notação $\frac{\partial}{\partial x}f(x)$ *para denotar a a derivada de* $f(x)$*. Seria melhor ter utilizado a notação* $\frac{d}{dx}f(x)$*, assim ficaria de acordo com a notação usual dos livros de Cálculo.*

Exemplo 5.23 *O comando* `define` *pode ser usado para definir novas derivadas. Aqui definimos a derivada de* $F(x)$ *como sendo a função* $G(x)$*.*

```
> define(F, diff(F(x),x)=G(x));
> diff(F(cos(x)), x);
```

$$-G(\cos(x))\sin(x)$$

```
> diff(cos(F(x)), x);
```

$$-\sin(F(x))G(x)$$

```
> diff(exp(F(x^2 + x - F(x))), x);
```

$$G(x^2+x-F(x))(2x+1-G(x))e^{(F(x^2+x-F(x)))}$$

O `diff` tem uma forma inercial que é o `Diff`. As sintaxes do `diff` e do `Diff` são idênticas. Como toda forma inercial, o objetivo do `Diff` é mostrar a derivada a ser calculada. Para efetuar o cálculo deve-se usar o `diff` ou aplicar um comando `value` ao `Diff`.

Exemplo 5.24 *Vamos calcular a derivada de* $-\ln(\cos(x))$ *aplicando o comando* `value` *à forma inercial* `Diff`.

```
> Diff(-ln(cos(x)), x);
```

$$\frac{\partial}{\partial x}(-\ln(\cos(x)))$$

```
> value(%);
```

$$\frac{\sin(x)}{\cos(x)}$$

Exemplo 5.25 *A forma inercial é útil para escrever fórmulas, conforme mostramos neste exemplo.*

```
> Diff(F(x)^G(x), x) = diff(F(x)^G(x), x);
```

$$\frac{\partial}{\partial x}F(x)^{G(x)} = F(x)^{G(x)}\left(\left(\frac{\partial}{\partial x}G(x)\right)\ln(F(x)) + \frac{G(x)\left(\frac{\partial}{\partial x}F(x)\right)}{F(x)}\right)$$

```
> Diff(sin(F(x)) + cos(G(x)), x):  % = value(%);
```

$$\frac{\partial}{\partial x}(\sin(F(x))+\cos(G(x))) = \cos(F(x))\left(\frac{\partial}{\partial x}F(x)\right) - \sin(G(x))\left(\frac{\partial}{\partial x}G(x)\right)$$

5.3.2 Derivadas de ordem superior

Derivadas de ordem n podem ser calculadas escrevendo-se a variável n vezes no comando `diff`. Mas, uma maneira mais prática é escrever um cifrão e a ordem da derivada ao lado da variável. Assim, para calcular a derivada enésima de $f(x)$ com relação a x, basta usar um comando `diff(`*f(x), x$n*`)`.

Outra maneira não muito prática de calcular uma derivada de ordem superior é encaixar vários comandos `diff`. A quantidade de comandos encaixados deve ser a mesma ordem da derivada. Assim, a derivada segunda de $f(x)$ pode ser calculada com um comando `diff( diff(`*f(x), x*`), x)`.

A forma inercial `Diff` também pode ser usada com derivadas de ordem superior.

Exemplo 5.26 *Calculemos agora de quatro maneiras a derivada terceira de* x^7+4x^4+1.

```
> y := x^7 + 4*x^4 + 1:
> diff(y, x, x, x);
```

$$210x^4 + 96x$$

```
> diff(y, x$3);
```

$$210x^4 + 96x$$

```
> diff( diff( diff(y, x), x), x);
```

$$210x^4 + 96x$$

```
> Diff(y, x$3);
```

$$\frac{\partial^3}{\partial x^3}(x^7 + 4x^4 + 1)$$

```
> value(%);
```

$$210x^4 + 96x$$

Exemplo 5.27 *Vamos mostrar agora que* $y(t) = e^{-4t}(a\cos(3t) + b\sin(3t))$ *satisfaz* $y'' + 25y' + 8y = 0$*, usando uma equação onde do lado esquerdo aparece a forma inercial* `Diff` *e do lado direito o* `diff` *que é quem realmente efetua os cálculos. É necessário uma chamada ao comando de simplificação, o* `simplify`*, para poder obtermos* 0 *no segundo membro da equação.*

```
> y := exp(-4*t)*(a*cos(3*t) + b*sin(3*t));
```

$$y := e^{(-4t)}(a\cos(3t) + b\sin(3t))$$

Para evitar que o Maple substitua o valor de y no primeiro membro, vamos colocar o y entre apóstrofos:

```
> Diff('y', t$2) + 8*Diff('y', t) + 25*'y' =
                  simplify(diff(y, t$2) + 8*diff(y, t) + 25*y);
```

$$\left(\frac{\partial^2}{\partial t^2}y\right) + 8\left(\frac{\partial}{\partial t}y\right) + 25y = 0$$

Exemplo 5.28 *Para encerrar esta seção, um exemplo que é útil para chamar a atenção para o potencial da Computação Algébrica. Calculamos aqui a derivada décima quinta da função tangente trigonométrica, algo muito incoveniente para ser feito sem o auxílio desse tipo de recurso. Note que a maioria dos números inteiros obtidos como coeficientes ultrapassam 1 quatrilhão, um cálculo que não demora mais do que 3 centésimos de segundo se for usado um computador que não esteja muito ultrapassado.*

```
> Diff(tan(x), x$15) = diff(tan(x), x$15);
```

$$\frac{\partial^{15}}{\partial x^{15}}\tan(x) = 134094848(1+\tan(x)^2)^2\tan(x)^{12} + 13754155008(1+\tan(x)^2)^3\tan(x)^{10}$$

$$+182172651520(1+\tan(x)^2)^4\tan(x)^8 + 559148810240(1+\tan(x)^2)^5\tan(x)^6$$

$$+460858269696(1+\tan(x)^2)^6\tan(x)^4 + 89702612992(1+\tan(x)^2)^7\tan(x)^2$$

$$+1903757312(1+\tan(x)^2)^8 + 16384\tan(x)^{14}(1+\tan(x)^2)$$

5.3.3 Derivadas de funções de várias variáveis

O cálculo de derivadas de funções com mais de uma variável é idêntico ao cálculo das funções de apenas uma variável.

Exemplo 5.29 *Neste exemplo calculamos várias derivadas da função de duas variáveis definida por* $F(x, y) = x^7y^5 + 10x$.

```
> F := x^7 * y^5 + 10*x;
```

$$F := x^7y^5 + 10x$$

```
> Diff(F, x) = diff(F, x);
```

$$\frac{\partial}{\partial x}\left(x^7y^5 + 10x\right) = 7x^6y^5 + 10$$

```
> Diff(F, y) = diff(F, y);
```

$$\frac{\partial}{\partial y}\left(x^7y^5 + 10x\right) = 5x^7y^4$$

```
> Diff(F, x, y) = diff(F, x, y);
```

$$\frac{\partial^2}{\partial y \partial x}\left(x^7y^5 + 10x\right) = 35x^6y^4$$

```
> Diff(F, y, x) = diff(F, y, x);
```

$$\frac{\partial^2}{\partial x \partial y}\left(x^7y^5 + 10x\right) = 35x^6y^4$$

```
> Diff(F, x$2, y$3) = diff(F, x$2, y$3);
```

$$\frac{\partial^5}{\partial y^3 \partial x^2}\left(x^7y^5 + 10x\right) = 2520x^5y^2$$

```
> Diff(F, x, x, y, y, y) = diff(F, x, x, y, y, y);
```

$$\frac{\partial^5}{\partial y^3 \partial x^2}\left(x^7y^5 + 10x\right) = 2520x^5y^2$$

```
> Diff(F, y$6, x$8) = diff(F, y$10, x$10);
```

$$\frac{\partial^{14}}{\partial x^8 \partial y^6}\left(x^7y^5 + 10x\right) = 0$$

Note que `diff(F, x$2, y$3)` *é o mesmo que* `diff(F, x, x, y, y, y)`.

Exemplo 5.30 *Reforçamos aqui nossa advertência de que o operador* `diff` *não aceita funções como argumentos, mas expressões algébricas.*

```
> f := (x, y, z) -> x^3 + cos(y*z^4);
```

$$f := (x, y, z) \to x^3 + \cos(yz^4)$$

```
> 'f(x, y, z)' = f(x, y, z); # checando a definição da função
```

$$f(x, y, z) = x^3 + \cos(yz^4)$$

```
> Diff(f, z, y) = diff(f, z, y); # não é o que se espera
```

$$\frac{\partial^2}{\partial y \partial z} f = 0$$

```
> Diff('f(x, y, z)', z, y) = diff(f(x, y, z), z, y);
```

$$\frac{\partial^2}{\partial y \partial z} f(x, y, z) = -4\cos(yz^4)z^7y - 4\sin(yz^4)z^3$$

Os apóstrofos foram utilizados para evitar que o Maple substituísse o valor de $f(x, y, z)$

5.3.4 O operador diferencial

A derivada de uma função é calculada com o *operador diferencial* D. Derivadas de ordem n podem ser calculadas na forma $D@@n$ ou na forma $D(D(\dots(D)\dots))$.

Exemplo 5.31 *Usamos o operador diferencial D para calcular aqui as derivadas de algumas funções conhecidas. Observe que o resultado retornado é sempre outra função.*

```
> D(cos);
```

$$-\sin$$

```
> D(sec)(x);
```

$$\sec(x)\ \tan(x)$$

```
> D(ln);
```

$$a \to \frac{1}{a}$$

```
> D(arctan)(z);
```

$$\frac{1}{1+z^2}$$

```
> D(arcsin);
```

$$a \to \frac{1}{\sqrt{1-a^2}}$$

Exemplo 5.32 *Definimos aqui uma função $f(x) = \ln(1+x+x^2)$ e calculamos sua derivada $g(x) = f'(x)$. Note que g é realmente uma função (e não uma expressão algébrica) pois, por exemplo, é possível calcular um valor como $g(2)$.*

```
> f := x -> ln(1 + x + x^2);
```

$$f := x \to \ln(1+x+x^2)$$

```
> g := D(f);
```

$$g := x \to \frac{1+2x}{1+x+x^2}$$

```
> g(2);
```

$$\frac{5}{7}$$

Exemplo 5.33 *Vamos definir agora um operador H que é uma soma da função dada com suas derivadas, até a terceira ordem.*

```
> Id := x -> x; # Definição da função identidade
```

$$Id := x \to x$$

```
> H := Id + D + D@@2 + D@@3 ;
```

$$H := Id + D + D^{(2)} + D^{(3)}$$

O operador H assim definido deve atuar em funções. Por exemplo, definimos uma função f dada por $f(x) = x^7$ e calculamos $H(f)$. Observe que o resultado é uma soma de funções.

```
> f := x ->x^7;
```

$$f := x \to x^7$$

```
> H(f);
```

$$f + (x \to 7x^6) + (x \to 42x^5) + (x \to 210x^4)$$

Agora, como $H(f)$ é uma função, podemos calcular o valor dessa função em um ponto dado, por exemplo no ponto a:

```
> H(f)(a);
```

$$a^7 + 7a^6 + 42a^5 + 210a^4$$

Usando o operador D, a derivada parcial com relação à i-ésima variável x_i de uma função $f(x_1, \ldots, x_i, \ldots)$ é denotada por $D[i](f)$.

Derivadas mistas como $\dfrac{\partial^2}{\partial x_i \partial x_j} f$ podem ser indicadas por $D[i, j](f)$. Em geral, a derivada de ordem n da função $f(x_1, \ldots, x_i, \ldots)$ com relação às variáveis $x_{s_1}, x_{s_2}, \ldots, x_{s_n}$ é $D[s_1, s_2, \ldots, s_n](f)$.

A repetição de variáveis pode ser abreviada com o uso do cifrão. Por exemplo, $D[\underbrace{i, i, \ldots, i}_{n \text{ vezes}}, \underbrace{j, j, \ldots, j}_{m \text{ vezes}}, \cdots]$ é o mesmo que $D[i\$n, j\$m, \cdots]$.

Exemplo 5.34 *Neste exemplo definimos uma função $f(x, y)$ e calculamos várias derivadas dela usando o operador diferencial. Usamos a forma inercial* `Diff` *para mostrar o que está sendo calculado a cada passo.*

```
> f := (x, y) -> x^10 - y^20 + x^3*y^4 + 1;
```

$$f := (x, y) \to x^{10} - y^{20} + x^3 y^4 + 1$$

```
> Diff('f(x,y)',x,y) - Diff('f(x,y)',y,x) = (D[1,2] - D[2,1])(f)(x, y);
```

$$\left(\frac{\partial^2}{\partial y \partial x} f(x, y)\right) - \left(\frac{\partial^2}{\partial x \partial y} f(x, y)\right) = 0$$

```
> Diff('f(x,y)', x$3, y$2) = D[1$3, 2$2](f)(x, y);
```

$$\frac{\partial^5}{\partial y^2 \partial x^3} f(x, y) = 72y^2$$

```
> Diff('f(x,y)', x$7) = D[1$7](f)(x, y);
```

$$\frac{\partial^7}{\partial x^7} f(x, y) = 604800x^3$$

Exemplo 5.35 *Definamos agora uma função laplaciano $\mathrm{L}(f) = D_{1,1}f + D_{2,2}f + D_{3,3}f$. A vantagem em usar o operador D (no lugar do* `diff`*, por exemplo) é que podemos derivar sem saber os nomes das variáveis, não importando que elas se chamem x, y, a, b, c, α, β, etc.*

```
> L := f -> (D[1$2] + D[2$2] + D[3$2])(f);
```

$$L := D[1, 1] + D[2, 2] + D[3, 3]$$

```
> g := (a, b, c) -> a^3 + b^4 + c^5;
```

$$g := (a, b, c) \to a^3 + b^4 + c^5$$

```
> L(g);
```

$$((a, b, c) \to 6a) + ((a, b, c) \to 12b^2) + ((a, b, c) \to 20c^3)$$

```
> L(g)(1, -2, 0);
```

$$54$$

```
> L(g)(x, y, z);
```

$$6x + 12y^2 + 20z^3$$

```
> L(g)(alpha, beta, omega);
```

$$6\alpha + 12\beta^2 + 20\omega^3$$

5.3.5 Derivadas de funções definidas implicitamente

Funções de uma ou várias variáveis definidas implicitamente por equações podem ter suas derivadas calculadas com o uso de um comando do tipo

`implicitdiff(`*equação, variável1, variável2, ...*`)`.

Exemplo 5.36 *Suponhamos que y seja definido implicitamente como função diferenciável de x através da equação $x^2 + y^2 = 1$. Então, a derivada de y com relação a x é calculada com o seguinte comando:*

```
> implicitdiff(x^2 + y^2 = 1, y, x);
```

$$-\frac{x}{y}$$

Exemplo 5.37 *O cálculo de derivadas de funções definidas implicitimente também pode ser feito passo a passo, se os comandos* `diff` *e* `solve` *forem utilizados, conforme mostramos a seguir. Neste caso, é preciso escrever $y(x)$, em vez de simplesmente y.*

```
> eq1 := x^2 + y(x)^2 = 1;
```

$$eq1 := x^2 + y(x)^2 = 1$$

```
> deq1 := diff(eq1, x);
```

$$deq1 := 2x + 2y(x)\left(\frac{\partial}{\partial x}\, y(x)\right) = 0$$

```
> solve(deq1, diff(y(x), x));
```

$$-\frac{x}{y(x)}$$

Exemplo 5.38 *Funções de várias variáveis também podem ser derivadas implicitamente com o* `implicitdiff` *de modo semelhante às funções de uma variável. Supondo que a equação $xyz + e^z - 9 = 0$ defina implicitamente z como uma função diferenciável de x e y, vamos calcular suas derivadas.*

```
> eq := x*y*z + exp(z) - 9 = 0;
```

$$eq := xyz + e^z - 9 = 0$$

```
> Diff(z, x) = implicitdiff(eq, z, x);
```

$$\frac{\partial}{\partial x} z = -\frac{yz}{xy + e^z}$$

```
> Diff(z, y) = implicitdiff(eq, z, y);
```

$$\frac{\partial}{\partial y} z = -\frac{xz}{xy + e^z}$$

Se uma função for definida implicitamente por um sistema de equações, então o `implicitdiff` também calcula sua derivada. Para isso, listam-se as equações do sistema em forma de conjunto $\{eq1, eq2, \dots\}$ e escreve-se o resultado como primeiro parâmetro do `implicitdiff`. Depois, escreve-se o segundo parâmetro que deve ser um conjunto $\{d_1, d_2, \dots\}$ com as variáveis que são dependentes e, finalmente, duas variáveis v_1 e v_2 para indicar a derivada de v_1 com relação v_2. Assim, o comando `implicitdiff` neste caso deve ter a forma

`implicitdiff(`{*eq1, eq2, ...*}, {d_1, d_2, ...}, v_1, v_2 `)`

Exemplo 5.39 *Suponhamos que y e z sejam funções diferenciáveis de x, definidas implicitamente pelo sistema de equações*

$$\begin{cases} x^3 + y^3 + z^3 &= 1 \\ x^5 + y^4 &= 10 \end{cases}$$

Vamos calcular as derivadas de y e de z com relação a x.

```
> eq1 := x^3 + y^3 + z^3 = 1;
```

$$eq1 := x^3 + y^3 + z^3 = 1$$

```
> eq2 := x^5 + y^4 = 10;
```

$$eq2 := x^5 + y^4 = 10$$

```
> `y'(x)` = implicitdiff({eq1, eq2}, {y, z}, y, x);
```

$$y'(x) = -\frac{5}{4}\frac{x^4}{y^3}$$

```
> `z'(x)` = implicitdiff({eq1, eq2}, {y, z}, z, x);
```

$$z'(x) = -\frac{1}{4}\frac{x^2(4y - 5x^2)}{z^2 y}$$

Exemplo 5.40 *Sejam $x = x(u, v)$ e $y = y(u, v)$ definidas implicitamente pelo sistema*

$$\begin{cases} u &= x^2 + y^2 \\ v &= xy \end{cases}$$

Vamos calcular as derivadas $\dfrac{\partial x}{\partial u}$ e $\dfrac{\partial y}{\partial v}$.

```
> s1 := u = x^2 + y^2;
```

$$s1 := u = x^2 + y^2$$

```
> s2 := v = x*y;
```

$$s2 := v = xy$$

```
> Diff(x, u) = implicitdiff({s1, s2}, {x, y}, x, u);
```

$$\frac{\partial}{\partial u}x = \frac{1}{2}\frac{x}{-y^2 + x^2}$$

```
> Diff(y, v) = implicitdiff({s1, s2}, {x, y}, y, v);
```

$$\frac{\partial}{\partial v}y = \frac{x}{-y^2 + x^2}$$

5.4 Integrais

O Maple possui amplos recursos para o cálculo de integrais definidas, indefinidas ou impróprias. Ele conhece as regras de integração usuais e os valores de muitas integrais em casos particulares.

5.4.1 Integrais de funções de uma variável

A integral (primitiva) de uma função definida por uma expressão algébrica $f(x)$ na variável x é calculada com um comando `int(`*f(x), x*`)`.

Esse comando também possui uma forma inercial: `Int(`*f(x), x*`)`. A forma inercial não efetua cálculos, apenas mostra a integral no formato usual e é útil na apresentação de fórmulas. Seu valor pode ser calculado aplicando-lhe um comando `value`.

Exemplo 5.41 *Vamos calcular* $\int(x^4 + 5x^3 + 9x - 1)dx$ *e derivar a resposta obtida para ver se coincide com o integrando dado. Depois, calculamos* $\int \sec(\alpha)d\alpha$ *e derivamos a resposta obtida. Muitas vezes é necessário uma simplificação na derivada para podermos observar que ela está relacionada com o integrando dado.*

```
> F := int(x^4 + 5*x^3 + 9*x - 1, x);
```

$$F := \frac{1}{5}x^5 + \frac{5}{4}x^4 + \frac{9}{2}x^2 - x$$

```
> diff(F, x);
```

$$x^4 + 5x^3 + 9x - 1$$

```
> G := int(sec(alpha), alpha);
```

$$G := \ln(\sec(\alpha) + \tan(\alpha))$$

```
> diff(G, alpha);
```

$$\frac{\sec(\alpha)\tan(\alpha) + 1 + \tan(\alpha)^2}{\sec(\alpha) + \tan(\alpha)}$$

```
> simplify(%);
```

$$\frac{1}{\cos(\alpha)}$$

Exemplo 5.42 *As regras básicas de integração como substituição, frações parciais e integração por partes são bem conhecidas pelo programa.*

```
> Int(x/(x^2 + 1), x) = int(x/(x^2 + 1), x);
```

$$\int \frac{x}{x^2+1}dx \;=\; \frac{1}{2}\ln(x^2+1)$$

```
> Int(x^3*exp(x^2), x) = int(x^3*exp(x^2), x);
```

$$\int x^3 e^{x^2}dx \;=\; \frac{1}{2}x^2e^{x^2} - \frac{1}{2}e^{x^2}$$

```
> Int(1/(x^4 + 1), x) = int(1/(x^4 + 1), x);
```

$$\int \frac{1}{x^4+1}dx \;=\; \frac{1}{8}\sqrt{2}\ln\left(\frac{x^2+x\sqrt{2}+1}{x^2-x\sqrt{2}+1}\right) + \frac{1}{4}\sqrt{2}\arctan(x\sqrt{2}+1) + \frac{1}{4}\sqrt{2}\arctan(x\sqrt{2}-1)$$

Exemplo 5.43 *O Maple supõe que as constantes das integrais satisfazem determinadas condições de modo a simplificar a resposta. Por exemplo, nos exemplos a seguir ele supõe* $n \neq -1$, $q \neq 0$, $r \neq s$ *e* $r \neq -s$.

```
> Int(x^n, x): % = value(%) + C;
```

$$\int x^n dx \;=\; \frac{x^{(n+1)}}{n+1} + C$$

```
> Int(1/(x*(x^q+1)), x): % = value(%) + C;
```

$$\int \frac{1}{x(x^q+1)}dx \;=\; -\frac{\ln(x^q+1)}{q} + \frac{\ln(x^q)}{q} + C$$

```
> Int(sin(r*t)*sin(s*t), t): % = value(%) + C;
```

$$\int \sin(rt)\sin(st)dt \;=\; \frac{1}{2}\frac{\sin((-r+s)t)}{-r+s} - \frac{1}{2}\frac{\sin((r+s)t)}{r+s} + C$$

5.4.2 Integrais definidas e impróprias

Uma integral definida em um intervalo $[a, b]$ é calculada com um comando do tipo `int(`*`f(x), x=a..b`*`)`. A função pode ser descontínua no intervalo dado.

Integrais impróprias são fornecidas como integrais definidas. Nesses casos podemos ter a ou b iguais a $+\infty$ ou $-\infty$.

Exemplo 5.44 *Calculemos a integral da função* $\cos(5t)$ *no intervalo* $[a, b]$ *e de* $\frac{1}{x^2+4}$ *no intervalo* $[1, 2]$.

```
> Int(cos(5*t), t = a..b) = int(cos(5*t), t = a..b);
```

$$\int_a^b \cos(5t)dt \;=\; \frac{1}{5}\sin(5b) - \frac{1}{5}\sin(5a)$$

```
> Int(1/(x^2+4), x = 1..2) = int(1/(x^2 + 4), x = 1..2);
```

$$\int_1^2 \frac{1}{x^2+4}dx \;=\; \frac{1}{8}\pi - \frac{1}{2}\arctan(\frac{1}{2})$$

Exemplo 5.45 *Pode ser que o problema precise de condições adicionais para poder ser calculado. A função* $f(x) = 1/(x-p)^3$ *possui uma descontinuidade no ponto* p. *Assim, para calcular a integral dessa função em um intervalo, é essencial saber se* p *pertence ou não ao intervalo dado. O comando* `int` *aceita uma opção* `continuous` *para avisar que a função dada é contínua no intervalo dado.*

Sem definir se p *pertence a* $[1, 5]$ *a integral não pode ser calculada:*

```
> int(1/(x - p)^3, x = 1..5);
```

$$\int_1^5 \frac{1}{(x-p)^3} dx$$

Avisando que p não pertence ao intervalo dado, ou seja, definindo a função como contínua, então ela pode ser calculada facilmente:

```
> int(1/(x - p)^3, x = 1..5, continuous);
```

$$-4\frac{-3+p}{(-5+p)^2(-1+p)^2}$$

Exemplo 5.46 *Aqui calculamos uma integral imprópria bem simples.*

```
> 1/(x^2 + 1), x=1/sqrt(3)..infinity: Int(%) = int(%);
```

$$\int_{\frac{1}{\sqrt{3}}}^{\infty} \frac{1}{x^2+1} dx = \frac{1}{3}\pi$$

Exemplo 5.47 *Às vezes o cálculo da integral imprópria só pode ser feito com condições adicionais. Neste exemplo, tentamos calcular a integral imprópria* $\int_{-3}^{4} \frac{1}{x} dx$. *Mas, o cálculo só é possível se considerarmos o Valor Principal de Cauchy da integral. Para isso, basta acrescentar no comando* `int` *uma opção* `CauchyPrincipalValue`.

```
> # Sem hipóteses adicionais, o seguinte problema não
> # tem solução (é indefinido)
> Int(1/x, x=-3..4) = int(1/x, x=-3..4);
```

$$\int_{-3}^{4} \frac{1}{x} dx = undefined$$

```
> # Acrescentando-se a hipótese do "Valor Principal de
> # Cauchy", o problema pode ser facilmente resolvido
> int(1/x, x=-3..4, CauchyPrincipalValue);
```

$$2\ln(2) - \ln(3)$$

Exemplo 5.48 *O Maple tem conhecimento de integrais impróprias famosas cujos cálculos normalmente não são acessíveis a quem estiver iniciando o estudo do Cálculo. Mostramos a seguir dois representantes ilustres dessa família de integrais.*

```
> exp(-x^2), x=-infinity..infinity: Int(%)=int(%);
```

$$\int_{-\infty}^{\infty} e^{-x^2} dx = \sqrt{\pi}$$

```
> sin(x)/x, x=0..infinity: Int(%)=int(%);
```

$$\int_0^{\infty} \frac{\sin(x)}{x} dx = \frac{1}{2}\pi$$

Exemplo 5.49 *É muito comum encontrarmos uma integral definida que não pode ser calculada de forma exata. Nesses casos, ela pode ser calculada numericamente de forma aproximada. Para isso, basta aplicar à integral um comando* `evalf`.

```
> int( root[3](x^5 + 1), x=2..7); # não pode ser calculada de forma exata
```

$$\int_2^7 (x^5+1)^{\frac{1}{3}} dx$$

```
> evalf(%);   # obtém uma aproximação numérica para a integral dada
```

$$64.88537023$$

Exemplo 5.50 *O Maple sabe como derivar funções definidas por integrais, mesmo que as integrais não possam ser calculadas de forma exata, como as que são mostradas neste exemplo.*

```
> F := x -> Int((1 + 1/t)^t, t=1..x);
```

$$F := x \to \int_1^x (1+\frac{1}{t})^t dt$$

```
> diff(F(x), x);
```

$$(1+\frac{1}{x})^x$$

```
> F(2); # Não consegue calcular exatamente ...
```

$$\int_1^2 (1+\frac{1}{t})^t dt$$

```
> evalf(F(2)); # ... mas calcula aproximadamente
```

$$2.142945126$$

```
> G := x -> Int(sin(t*cos(t)), t = -x^2..exp(-3*x));
```

$$G := x \to \int_{-x^2}^{e^{(-3x)}} \sin(t\cos(t))dt$$

```
> D(G)(x);
```

$$-3e^{(-3x)}\sin(e^{(-3x)}\cos(e^{(-3x)})) - 2x\sin(x^2\cos(x^2))$$

5.4.3 Integrais duplas e triplas

O pacote `student` possui dois comandos na forma inercial que são úteis nos cálculos de integrais múltiplas iteradas. O `value` aplicado a essas formas inerciais permite calcular seus valores.

A forma inercial da integral dupla de uma função de duas variáveis definida por uma expressão algébrica $f(x,y)$ nas variáveis x, y é:

`Doubleint(`*f(x, y), x=a..b, y=c..d*`)`,

onde *a..b* e *c..d* denotam a variação do x e do y, respectivamente. Essa forma inercial é equivalente a `Int(Int(`*f(x, y), x=a..b*`), `*y=c..d*`)`.

Os intervalos de integração podem depender das variáveis e a ordem da integração pode ser permutada em alguns casos.

Exemplo 5.51 *Calculemos a integral dupla de $x+y$ com $y \le x \le 3y$ e $1 \le y \le 2$.*

```
> with(student):
> Doubleint(x + y, x=y..3*y, y=1..2): %=value(%);
```

$$\int_1^2 \int_y^{3y} x + y \, dx \, dy \; = \; 14$$

O `Doubleint` *anterior é equivalente a:*

```
> Int(Int(x + y, x=y..3*y), y=1..2): %=value(%);
```

$$\int_1^2 \int_y^{3y} x + y \, dx \, dy \; = \; 14$$

Exemplo 5.52 *Calculemos a integral dupla de* $f(x, y, z) = x + 2y + 3$ *com* $0 \leq y \leq 4 - x^2$ *e* $-2 \leq x \leq 2$.

```
> Doubleint(x + 2*y + 3, y=0..4-x^2, x=-2..2): %=value(%);
```

$$\int_{-2}^2 \int_0^{4-x^2} x + 2y + 3 \, dy \, dx \; = \; \frac{992}{15}$$

Exemplo 5.53 *Integrais duplas em coordenadas polares também podem ser calculadas com o* `Doubleint`. *Neste exemplo calculamos a integral dupla de* $\rho \sin(\theta)$ *com* $0 \leq \rho \leq \cos(\theta)$ *e* $0 \leq \theta \leq \pi$.

```
> Doubleint(rho*sin(theta), rho=0..cos(theta), theta=0..Pi):
> % = value(%);
```

$$\int_0^{\pi} \int_0^{\cos(\theta)} \rho \, d\rho \, d\theta \; = \; \frac{1}{3}$$

A forma inercial da integral tripla de uma função de três variáveis definida por uma expressão algébrica $f(x, y, z)$ nas variáveis x, y, z é dada por:

`Tripleint(`*f(x, y, z), x=a..b, y=c..d, z=e..f*`)`,

onde *a..b*, *c..d* e *e..f* denotam a variação do x, y e do z, respectivamente. O Tripleint é equivalente a três comandos `Int` encaixados:

`Int(Int(Int(`*f(x, y, z), x=a..b*`), `*y=c..d*`), `*z=e..f*`)`.

Exemplo 5.54 *Calculemos a integral tripla de* $f(x, y, z) = z$ *com* $0 \leq z \leq \sqrt{4 - y^2}$, $2 - y \leq x \leq 6 - 2y$ *e* $0 \leq y \leq 2$.

```
> with(student):
> Tripleint(z, z=0..sqrt(4-y^2), x=2-y..6-2*y, y=0..2): %=value(%);
```

$$\int_0^2 \int_{2-y}^{6-2y} \int_0^{\sqrt{4-y^2}} z \, dz \, dx \, dy \; = \; \frac{26}{3}$$

O `Tripleint` *anterior é equivalente a:*

```
> Int(Int(Int(z, z=0..sqrt(4-y^2)), x=2-y..6-2*y), y=0..2): %=value(%);
```

$$\int_0^2 \int_{2-y}^{6-2y} \int_0^{\sqrt{4-y^2}} z \, dz \, dx \, dy \; = \; \frac{26}{3}$$

Exemplo 5.55 *Integrais triplas em coordenadas esféricas ou cilíndricas podem ser calculadas sem maiores dificuldades. Aqui, calculamos a integral tripla de* ρ *com* $0 \leq z \leq \rho^2$, $0 \leq \rho \leq 2\cos(\theta)$ *e* $0 \leq \theta \leq \pi/2$.

```
> Tripleint(rho, z=0..rho^2, rho=0..2*cos(theta), theta=0..Pi/2):
> % = value(%);
```

$$\int_0^{\frac{1}{2}\pi} \int_0^{2\cos(\theta)} \int_0^{\rho^2} \rho \, dz \, d\rho \, d\theta \; = \; \frac{3}{4}\pi$$

5.5 Gradiente, divergente, rotacional e laplaciano

O gradiente, o divergente, o rotacional e o laplaciano podem ser calculados com os comandos `grad`, `diverge`, `curl` e `laplacian` do pacote `linalg`.

A sintaxe deles são parecidas entre si: nome do comando seguido da função e da lista de variáveis. No caso do divergente e do rotacional a função deve ser uma função vetorial.

Exemplo 5.56 *Neste exemplo as definições desses operadores podem ser recordadas.*

```
> with(linalg):     # Ignore a advertência bobinha a seguir
```

Warning, the protected names norm and trace have been redefined and unprotected

```
> F := [f(x,y,z), g(x,y,z), h(x,y,z)]:
> v := [x, y, z]:  # lista de variáveis
>
> curl(F, v); # rotacional de F
```

$$\left[\left(\frac{\partial}{\partial y}\,f(x,y,z)\right)-\left(\frac{\partial}{\partial z}\,f(x,y,z)\right),\left(\frac{\partial}{\partial z}\,f(x,y,z)\right)-\left(\frac{\partial}{\partial x}\,f(x,y,z)\right),\left(\frac{\partial}{\partial x}\,f(x,y,z)\right)-\left(\frac{\partial}{\partial y}\,f(x,y,z)\right)\right]$$

```
> diverge(F, v); # divergente de F
```

$$\left(\frac{\partial}{\partial x}\,f(x,y,z)\right)+\left(\frac{\partial}{\partial y}\,g(x,y,z)\right)+\left(\frac{\partial}{\partial z}\,h(x,y,z)\right)$$

```
> grad(f(x,y,z), v); # gradiente de f
```

$$\left[\frac{\partial}{\partial x}\,f(x,y,z),\ \frac{\partial}{\partial y}\,f(x,y,z),\ \frac{\partial}{\partial z}\,f(x,y,z)\right]$$

```
> laplacian(f(x,y,z), v); # laplaciano de f
```

$$\left(\frac{\partial^2}{\partial x^2}f(x,y,z)\right)+\left(\frac{\partial^2}{\partial y^2}f(x,y,z)\right)+\left(\frac{\partial^2}{\partial z^2}f(x,y,z)\right)$$

Exemplo 5.57 *Neste exemplo é calculado o gradiente de $x^3+y^4-9z^2$ nas variáveis x,y,z e o gradiente de rs^2e^t nas variáveis r,s,t.*

```
> with(linalg):
> grad(x^3 + y^4 - 9*z^2, [x,y,z]);
```

$$\left[3x^2,\ 4y^3,\ -18z\right]$$

```
> F := r*s^2*exp(t): var := [r, s, t]:
> grad(F, var);
```

$$\left[s^2e^t,\ 2rse^t,\ rs^2e^t\right]$$

Exemplo 5.58 *Um exemplo simples de cálculo de laplaciano*

```
> with(linalg):
> laplacian(x^4 + y^4 + z^4, [x,y,z]);
```

$$12x^2+12y^2+12z^2$$

Exemplo 5.59 *Consideremos a função vetorial* $f(u, v, w) = (e^u \cos(v),\, e^u \sin(v),\, \ln(w))$ *e calculemos seus rotacional e divergente.*

```
> f := [exp(u)*cos(v), exp(u)*sin(v), ln(w)]:
> X := [u, v, w]: # lista de variáveis
> with(linalg):
> curl(f, X); # rotacional de f
```

$$[0,\, 0,\, 2e^u \sin(v)]$$

```
> diverge(f, X); # divergente de f
```

$$2e^u \cos(v) + \frac{1}{w}$$

Exemplo 5.60 *Neste exemplo calculamos o rotacional e o divergente da função vetorial* $F(x, y, z) = (y, x^3 \cos(z),\, xyz)$.

```
> F := [y, x^3*cos(z), x*y*z]:  var := [x, y, z]:
> with(linalg):
> RF := curl(F, var); # rotacional de F
```

$$RF := \left[xz + x^3 \sin(z),\, -yz,\, 3x^2 \cos(z) - 1\right]$$

```
> diverge(F, var);   # divergente de F
```

$$xy$$

```
> diverge(RF, var);  # divergente do rotacional de F
```

$$0$$

Exemplo 5.61 *Agora vamos ver como fica o cálculo desses operadores nos sistemas de coordenadas cilíndricas e esféricas. Os comandos são semelhantes ao do caso do sistema de coordenadas cartesianas, com a única diferença de acrescentar um terceiro parâmetro do tipo* `coords = cylindrical` *ou* `coords = esferical`.

```
> with(linalg):
>
> # gradiente em coordenadas cilíndricas
> f := r^2*cos(theta)*z^3:  v := [r, theta, z]:
> grad(f, v, coords=cylindrical);
```

$$\left[2r \cos(\theta) z^3,\, -r \sin(\theta) z^3,\, 3r^2 \cos(\theta) z^2\right]$$

```
>
> # laplaciano em coordenadas cilíndricas
> f := z*cos(theta):  v := [r, theta, z]:
> laplacian(f, v, coords=cylindrical);
```

$$-\frac{z \cos(\theta)}{r^2}$$

```
>
> # divergente em coordenadas esféricas
> G := vector([r, sin(theta), cos(phi)]):  v := vector([r, theta, phi]):
> diverge(G, v, coords=spherical);
```

$$\frac{3r \sin(\theta) + 2 \sin(\theta) \cos(\theta) - \sin(\phi)}{r \sin(\theta)}$$

5.6 Somatórios, produtórios, séries de potências

5.6.1 Somatórios

Um somatório pode ser calculado pelo Maple com um comando `sum(`*f(n), n=a..b*`)` que admite dois parâmetros: um com o termo geral $f(n)$ do somatório dependendo de um inteiro n e outro parâmetro com o intervalo de variação da variável no formato $n = a..b$ significando que $a \leq n \leq b$. A forma inercial `Sum(`*f(n), n=a..b*`)` possui a mesma sintaxe e pode ser calculada aplicando-lhe um comando `value`.

Exemplo 5.62 *A soma da progressão geométrica cujo termo geral é $a_n = 2^n$, com n variando de 1 a k é calculada com o comando:*

```
> sum(2^n, n = 1..k);
```

$$2^{(k+1)} - 2$$

De modo semelhante, a soma da progressão aritmética cujo termo geral é $a_n = 4n + 7$ com n variando de 1 a k é calculada por:

```
> sum(4*n + 7, n = 1..k);
```

$$2(n+1)^2 + 5n - 2$$

Podemos ter uma soma de uma quantidade infinita de termos, como mostrado a seguir onde calculamos a soma de uma progressão geométrica infinita cujo termo geral é $(3/5)^n$:

```
> sum((3/5)^n, n=0..infinity);
```

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

Exemplo 5.63 *Com o uso da forma inercial* `Sum` *podemos escrever fórmulas ou sentenças completas, que dispensam enunciados.*

```
> Sum(a[n], n=1..10) = sum(a[n], n=1..10);
```

$$\sum_{n=1}^{10} a_n = a_1 + a_2 + a_3 + a_4 + a_5 + a_6 + a_7 + a_8 + a_9 + a_{10}$$

```
> Sum('x^n/n!', 'n'=0..6) = sum('x^n/n!', 'n'=0..6);
```

$$\sum_{n=0}^{6} \frac{x^n}{n!} = 1 + x + \frac{1}{2}x^2 + \frac{1}{6}x^3 + \frac{1}{24}x^4 + \frac{1}{120}x^5 + \frac{1}{720}x^6$$

```
> Limit(Sum('x^n/n!', 'n'=0..k), k=infinity) = exp(x);
```

$$\lim_{k\to\infty} \sum_{n=0}^{k} \frac{x^n}{n!} = e^x$$

Em alguns casos o uso de apóstrofos é recomendado para evitar que sejam feitas substituições indesejáveis, por exemplo, quando variáveis com mesmos nomes tiverem sido usadas anteriormente.

Exemplo 5.64 *O Maple conhece o valor da soma de várias séries infinitas, como por exemplo as somas das p-séries (cujo termo geral é $a_n = 1/n^p$) com p par e $p \leq 48$. Neste exemplo, calculamos a soma $1 + \frac{1}{16} + \frac{1}{81} + \cdots + \frac{1}{n^4} + \cdots$ que, surpreendentemente, é uma expressão onde aparece π^4.*

```
> Sum(1/k^4, k = 1..infinity): %=value(%);
```

$$\sum_{k=1}^{\infty} \frac{1}{k^4} = \frac{1}{90}\pi^4$$

Calculando a soma parcial P dos 20 primeiros termos da série:

```
> P := sum(1/k^4, k = 1..20);
```

$$P := \frac{3178468114001972330616643665140369}{2936813615588238080381960232960000}$$

```
> P := evalf(P); # substitui P pelo seu valor decimal
```

$$P := 1.082284588$$

Note que P é uma aproximação para a soma da série, cujo valor na notação decimal é mostrado a seguir:

```
> Pi^4/90: % = evalf(%);
```

$$\frac{1}{90}\pi^4 = 1.082323234$$

Sendo P uma aproximação para a soma da série, a raiz quarta de $90P$ é uma aproximação para o o valor de π:

```
> root[4](90*P);
```

$$3.141564610$$

Exemplo 5.65 *Segue diversos cálculos de somas de séries, alguns são bem conhecidos.*

```
> Sum(1/n!, n=0..infinity): %=value(%);
```

$$\sum_{n=0}^{\infty} \frac{1}{n!} = e$$

```
> Sum(k^2, k=1..n): %=value(%);
```

$$\sum_{k=1}^{n} k^2 = \frac{1}{3}(n+1)^3 - \frac{1}{2}(n+1)^2 + \frac{1}{6}n + \frac{1}{6}$$

```
> Sum(k^5, k = 1..n): %=value(%);
```

$$\sum_{k=1}^{n} k^5 = \frac{1}{6}(n+1)^6 - \frac{1}{2}(n+1)^5 + \frac{5}{12}(n+1)^4 - \frac{1}{12}(n+1)^2$$

```
> Sum( 1/n, n=1..infinity): %=value(%); # série harmônica
```

$$\sum_{n=1}^{\infty} \frac{1}{n} = \infty$$

```
> Sum (1/(3*n+7), n=2..infinity): %=value(%);
```

$$\sum_{n=2}^{\infty} \frac{1}{3n+7} = \infty$$

```
> Sum ((-1)^n/(3*n + 7), n=2..infinity): %=value(%);
```

$$\sum_{n=2}^{\infty} \frac{(-1)^n}{3n+7} = -\frac{111}{140} + \frac{1}{9}\pi\sqrt{3} + \frac{1}{3}\ln(2)$$

5.6.2 Produtórios

O cálculo de produtórios é feito com o comando `product`, cujo sintaxe é semelhante à do comando `sum`. Esse comando também possui uma forma inercial chamada `Product`.

Exemplo 5.66 *Utilizamos a forma inercial combinada com o* `value` *para calcular três produtos.*

```
> Product( a[k], k=1..8): %=value(%);
```

$$\prod_{k=1}^{8} a_k \;=\; a_1a_2a_3a_4a_5a_6a_7a_8$$

```
> Product( 2^n, n=0..100):  % = ifactor(value(%));
```

$$\prod_{n=0}^{100} 2^n \;=\; (2)^{5050}$$

```
> Product( (1+1/n^2), n=1..infinity): %=value(%);
```

$$\prod_{n=1}^{\infty}\left(1+\frac{1}{n^2}\right) \;=\; \frac{\sinh(\pi)}{\pi}$$

5.6.3 Séries de potências

O Maple possui muitos comandos para serem utilizados com séries de potências.

Vamos inicialmente citar um dos mais simples: o que obtém desenvolvimento em série de Taylor de uma função.

A sintaxe desse comando é `taylor(`*f(x), x=a, n*`)` ou `series(`*f(x), x=a, n*`)` onde $f(x)$ é uma expressão algébrica na variável x, a é o ponto em torno do qual será feito o desenvolvimento em série (ou seja, os termos da série são potências de $(x-a)$) e n corresponde à ordem.

Exemplo 5.67 *Neste exemplo, obtemos desenvolvimentos de Taylor da função* $\operatorname{sen}(x)$ *em série de potências de* x *e de* $(x-1)$*. Depois, convertemos as séries obtidas para polinômios com um comando* `convert(`*série*`, polynom)`*.*

```
> S1 := taylor(sin(x), x=0, 10);
```

$$S1 := x - \frac{1}{6}x^3 + \frac{1}{120}x^5 - \frac{1}{5040}x^7 + \frac{1}{362880}x^9 + O(x^{10})$$

```
> S2 := taylor(sin(x), x = 1, 4);
```

$$S2 := \sin(1) + \cos(1)(x-1) - \frac{1}{2}\sin(1)(x-1)^2 - \frac{1}{6}\cos(1)(x-1)^3 + O((x-1)^4)$$

A notação $O(f(x-a))$ *representa uma função que tende a 0 mais rapidamente do que* $f(x-a)$ *quando* $x \to a$*, ou seja, significa que* $\lim\limits_{x\to a}\dfrac{O(f(x-a))}{f(x-a)} = 0$*. Por exemplo, o* $O(x^{10})$ *utilizado anteriormente significa que* $\lim\limits_{x\to 0}\dfrac{O(x^{10})}{x^{10}} = 0$*.*

```
> P1 := convert(S1, polynom); # converte S1 para polinômio
```

$$P1 := x - \frac{1}{6}x^3 + \frac{1}{120}x^5 - \frac{1}{5040}x^7 + \frac{1}{362880}x^9$$

```
> P2 := convert(S2, polynom); # converte S2 para polinômio
```

$$P2 := \sin(1) + \cos(1)(x-1) - \frac{1}{2}\sin(1)(x-1)^2 - \frac{1}{6}\cos(1)(x-1)^3$$

Depois da conversão para polinômio, diversas operações podem ser feitas como cálculo de derivadas, integrais, construção de gráficos, etc. Aqui, transformamos $P1$ em uma função f e calculamos $f(\pi/6)$ que é uma aproximação para $\operatorname{sen}(\pi/6)$.

```
> f := unapply(P1, x);
```

$$f := x \to x - \frac{1}{6}x^3 + \frac{1}{120}x^5 - \frac{1}{5040}x^7 + \frac{1}{362880}x^9$$

```
> evalf( f(Pi/6) );
```

$$.5000000003$$

O pacote `powseries` possui vários comandos para manipulação de séries de potências.

```
> with(powseries);
```

[*compose, evalpow, inverse, multconst, multiply, negative, powadd, powcos, powcreate, powdiff, powexp, powint, powlog, powpoly, powsin, powsolve, powsqrt, quotient, reversion, subtract, template, tpsform*]

Normalmente, as séries obtidas com esses comandos não são mostradas automaticamente. Para ver uma expressão truncada da série, devemos usar um comando `tpsform`(*f, variável, ordem*) onde f está relacionado com os coeficientes da série.

Nesse pacote, uma série cujos coeficientes do termo geral são definidos por igualdade(s) do tipo $f(n) = expr$ é definida com um comando `powcreate`(*f(n) = expr*). Depois disso, o f pode ser passado como parâmetro para comandos que executam operações com as séries.

Exemplo 5.68 *Aqui criamos uma série S_1 cujo termo geral é $a_n = 1/n!$. Depois, usamos S_1 para construir uma série S_2 cujos valores nos pontos do seu domínio correspondem aos inversos multiplicativo dos de S_1.*

```
> with(powseries):
> powcreate(a(n) = 1/n!):
> tpsform(a, x, 6);
```

$$1 + x + \frac{1}{2}x^2 + \frac{1}{6}x^3 + \frac{1}{24}x^4 + \frac{1}{120}x^5 + O(x^6)$$

```
> b := inverse(a):
> tpsform(b, x, 6);
```

$$1 - x + \frac{1}{2}x^2 - \frac{1}{6}x^3 + \frac{1}{24}x^4 - \frac{1}{120}x^5 + O(x^6)$$

Agora, vamos multiplicar as séries com coeficientes definidos por a e b. Note que o resultado é a série constante igual a 1.

```
> c := multiply(a, b):
> tpsform(c, x, 6);
```

$$1 + O(x^6)$$

Exemplo 5.69 *Neste exemplo definimos duas séries cujos coeficientes dos termos gerais são a_n e b_n, duas seqüências definidas recursivamente por $a_n = a_{n-1} + a_{n-2}$, $a_0 = a_1 = 1$ e $b_n = 1 + \frac{1}{b_{n-1}}$, $b_0 = 1$. Depois, somamos as séries obtidas, integramos a primeira e derivamos a segunda termo a termo.*

```
> with(powseries):
>
> # Definição da série 1 cujo termo geral é a(n)
> powcreate(a(n) = a(n-2) + a(n-1), a(0)=1, a(1)=1):
>
> # Definição da série 2 cujo termo geral é b(n)
> powcreate(b(n)=1 + 1/b(n-1), b(0)=1):
>
> tpsform(a, x, 7); # mostra termos da série 1
```

$$1 + x + 2x^2 + 3x^3 + 5x^4 + 8x^5 + 13x^6 + O(x^7)$$

```
> tpsform(b, x, 7); # mostra termos da série 2
```

$$1 + 2x + \frac{3}{2}x^2 + \frac{5}{3}x^3 + \frac{8}{5}x^4 + \frac{13}{8}x^5 + \frac{21}{13}x^6 + O(x^7)$$

```
> s := powadd(a, b): # soma das séries 1 e 2
> tpsform(s, x, 7);
```

$$2 + 3x + \frac{7}{2}x^2 + \frac{14}{3}x^3 + \frac{33}{5}x^4 + \frac{77}{8}x^5 + \frac{190}{13}x^6 + O(x^7)$$

```
> d := powdiff(b); # derivada da série 2
```

$$d \;:=\; proc(powparm) \;\ldots\; end\;\; proc$$

```
> tpsform(d, x, 7);
```

$$2 + 3x + 5x^2 + \frac{32}{5}x^3 + \frac{65}{8}x^4 + \frac{126}{13}x^5 + \frac{34}{3}x^6 + O(x^7)$$

```
> i := powint(a); # integral da série 1
```

$$i \;:=\; proc(powparm) \;\ldots\; end\;\; proc$$

```
> tpsform(i, x, 7);
```

$$x + \frac{1}{2}x^2 + \frac{2}{3}x^3 + \frac{3}{4}x^4 + x^5 + \frac{4}{3}x^6 + O(x^7)$$

Exemplo 5.70 *O pacote* `powseries` *possui comandos para a construção de séries de potências de várias funções básicas como logaritmo, raiz quadrada, seno, cosseno, etc. Aqui, construímos as séries do seno e da exponencial (de base e) e efetuamos algumas operações aritméticas com essas séries.*

```
> with(powseries):
> SENO := powsin(x): # série para sen(x)
> tpsform(SENO, x, 8);
```

$$x - \frac{1}{6}x^3 + \frac{1}{120}x^5 - \frac{1}{5040}x^7 + O(x^8)$$

```
> EXPO := powexp(x): # série para exp(x)
> tpsform(EXPO, x, 8);
```

$$1 + x + \frac{1}{2}x^2 + \frac{1}{6}x^3 + \frac{1}{24}x^4 + \frac{1}{120}x^5 + \frac{1}{720}x^6 + \frac{1}{5040}x^7 + O(x^8)$$

```
> COMP := compose(EXPO, SENO): # série para a composta exp(sen(x))
> tpsform(COMP, x, 8);
```

$$1 + x + \frac{1}{2}x^2 - \frac{1}{8}x^4 - \frac{1}{15}x^5 - \frac{1}{240}x^6 + \frac{1}{90}x^7 + O(x^8)$$

```
> MULT := multiply(EXPO, SENO): # série para sen(x) exp(x)
> tpsform(MULT, x, 8);
```

$$x + x^2 + \frac{1}{3}x^3 - \frac{1}{30}x^5 - \frac{1}{90}x^6 - \frac{1}{630}x^7 + O(x^8)$$

```
> QUOC := quotient(SENO, EXPO): # série para sen(x)/exp(x)
> tpsform(QUOC, x, 8);
```

$$x - x^2 + \frac{1}{3}x^3 - \frac{1}{30}x^5 + \frac{1}{90}x^6 - \frac{1}{630}x^7 + O(x^8)$$

5.7 Exercícios

1) Calcule os limites:

a) $\lim\limits_{x\to\infty} \dfrac{\sqrt{x}}{\sqrt{x+\sqrt{x+\sqrt{x}}}}$;

b) $\lim\limits_{x\to\infty} \left(\sqrt[3]{(x+a)(x+b)(x+c)} - x\right)$;

c) $\lim\limits_{x\to\infty} \left(\sqrt[4]{(x+a)(x+b)(x+c)(x+d)} - x\right)$.

2) Sabendo que a série

$$\sum_{n=0}^{\infty} \frac{1}{4^n}\left(\frac{4}{8n+8} + \frac{1}{4n+2}\right)$$

converge para $\ln 2$, adicionando os 50 primeiros termos dessa série obtenha o valor de $\ln 2$ com 32 casas decimais exatas.

3) A série

$$\sum_{n=0}^{\infty} \frac{1}{16^n}\left(\frac{4}{8n+1} - \frac{2}{8n+4} - \frac{1}{8n+5} - \frac{1}{8n+6}\right)$$

converge para π. Adicione os 11 primeiros termos dessa série e verifique que a soma obtida corresponde ao valor de π com 15 casas decimais exatas.

4) A série

$$\frac{\sqrt{8}}{9801}\sum_{n=0}^{\infty}\left[\frac{(4n)!(1103+26390n)}{(n!)^4 396^{4n}}\right]$$

converge para $1/\pi$. Verifique que adicionando-se os 11 primeiros termos dessa série e invertendo-se a soma obtida, obtemos o valor de π com 39 casas decimais exatas.

5) Mostre que se $0 < x < 1$, então

$$f(x) \;=\; 2\,\mathrm{arctg}\left(\frac{1+x}{1-x}\right) + \;\mathrm{arcsen}\left(\frac{1-x^2}{1+x^2}\right)$$

é uma função constante. Para isso, calcule e simplifique sua derivada.

6) Os polinômios de Legendre podem ser definidos por

$$P_n(x) = \frac{1}{2^n n!}\frac{d^n}{dx^n}(x^2-1)^n.$$

Escreva $P_4(x)$ e $P_5(x)$ e calcule suas raízes.

7) A regra de Gauss com n pontos para o cálculo aproximado de uma integral de uma função contínua $f(x)$ no intervalo $[-1,1]$ consiste em aproximar $\int_{-1}^{1} f(x)dx$ pelo somatório $\sum_{i=1}^{n} a_i f(t_i)$ onde t_i é a i-ésima raiz do polinômio de Legendre $P_n(x)$ (veja exercício anterior) e $a_i = \dfrac{2(1-t_i^2)}{[nP_{n-1}(t_i)]^2}$. Quanto maior o valor de n, melhor a aproximação.

a) Verifique que para $n = 5$ a regra de Gauss pode ser escrita como

$$\int_{-1}^{1} f(x)dx \approx 0,236927 f(-0,906179) + 0,478629 f(-0,538469) + 0,568889 f(0) + 0,478629 f(0,538469) + 0,236927 f(0,906179).$$

b) Calcule $\displaystyle\int_{-1}^{1} \frac{1}{1+x^2+x^4}dx$ usando a regra do item (a) e compare com o valor exato da integral obtida com os comandos `int` e `evalf`.

8) Considere a seqüência de funções

$$f_n(x) = \sum_{k=1}^{n} a^k \cos(\lambda^k x)$$

onde $0 < a < 1$, λ é um inteiro ímpar maior ou igual a 3 e $a\lambda > 1$. Escolhamos $a = 2/3$ e $\lambda = 9$.

a) Construa o gráfico de $f_n(x)$ quando $n = 10$ e $-0,01 \leq x \leq 0,01$;

b) Calcule $f_n'(1/2)$ quando $n \in \{1, 5, 10, 15, 20\}$.

Capítulo 6

Problemas de Cálculo e de Geometria Analítica

Inúmeros problemas de Cálculo Diferencial e Integral e de Geometria Analítica podem ser resolvidos com o auxílio do Maple. Praticamente todo tipo de problema pode ter uma participação maior ou menor desse tipo de programa na sua resolução. O que apresentamos neste capítulo é apenas uma pequena amostra dessa variedade de problemas.

O pacote `student` possui comandos que podem ser úteis na resolução de problemas passo a passo. A título de ilustração, são abordados vários exemplos de utilização de comandos desse pacote.

```
> with(student);
```

[D, Diff, Doubleint, Int, Limit, Lineint, Product, Sum, Tripleint, changevar, completesquare, distance, equate, integrand, intercept, intparts, leftbox, leftsum, makeproc, middlebox, middlesum, midpoint, powsubs, rightbox, rightsum, showtangent, simpson, slope, summand, trapezoid]

6.1 Operações elementares com gráficos de funções

Conhecido o gráfico de uma função $f(x)$, podemos construir a partir dele vários outros gráficos através de operações de translação, reflexão, ampliação e redução. O Maple é útil para observar os efeitos desse tipo de operação ao construir o gráfico de $f(x)$ e o gráfico transformado em um mesmo sistema de eixos, conforme exemplificamos a seguir com a função polinomial $f(x) = x^3 - x$.

```
> f := x -> x^3 - x;
```

$$f := x \to x^3 - x$$

Exemplo 6.1 *O gráfico de $-f(x)$ é o gráfico de $f(x)$ "virado de cabeça para baixo", ou seja, refletido com relação ao eixo Ox (Figura 6.1).*

```
> plot([f(x), -f(x)], x=-2..2, y=-2..2, thickness=[0, 3],
>                                       color=[red, blue]);
```

Exemplo 6.2 *Ao somar uma constante positiva k à variável x, o gráfico desloca-se k unidades para a esquerda (Figura 6.2).*

Figura 6.1:

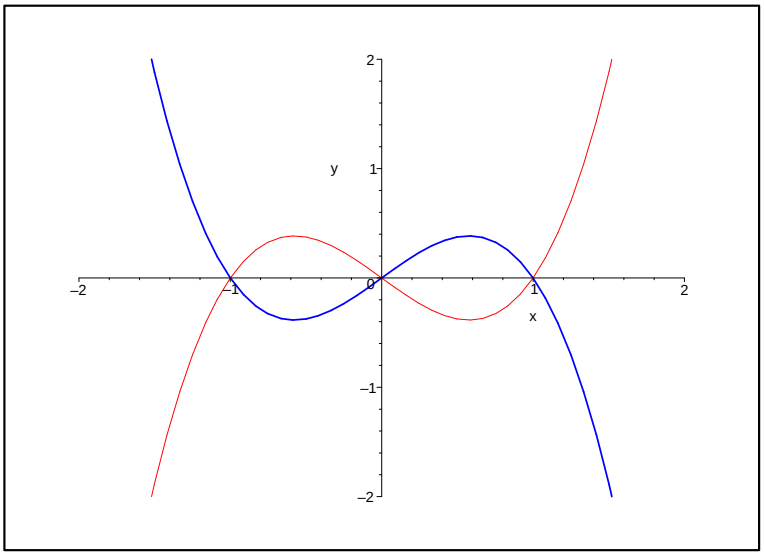
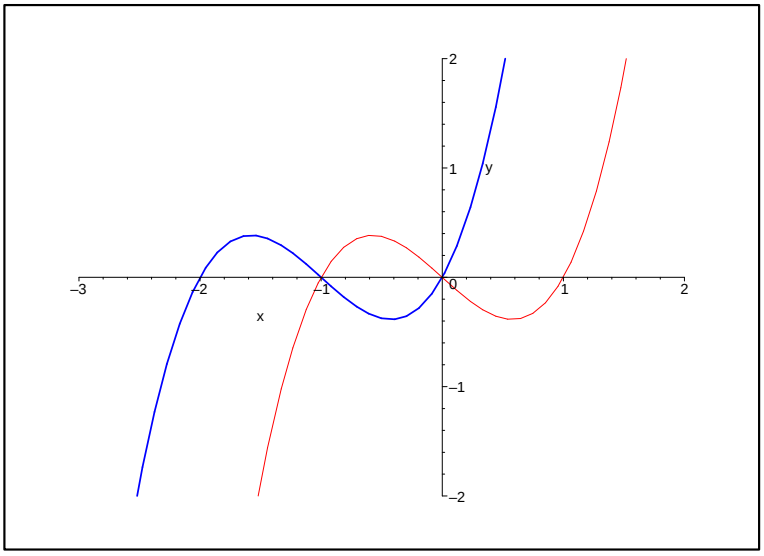


Figura 6.2:



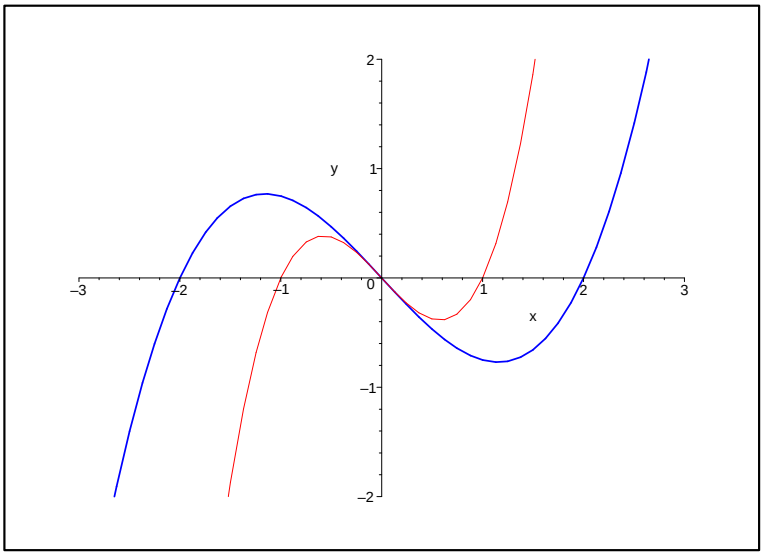

```
> k := 1;
> plot([f(x), f(x + k)], x=-3..2, y=-2..2, thickness=[0, 3],
>                                          color=[red, blue]);
```

Exemplo 6.3 *Uma operação do tipo $kf(x/k)$ causa uma ampliação ou redução no gráfico de $f(x)$ (Figura 6.3).*

```
> k := 2;
> plot([f(x), k*f(x/k)], x=-3..3, y=-2..2, thickness=[0, 3],
>                                          color=[red, blue]);
```

Outros gráficos que podem ser construídos são os de $|f(x)|$, $f(|x|)$, $f(x)+k$, $f(x)-k$, $f(x-k)$, $f(-x)$, $kf(x)$ e $f(kx)$ onde $k > 0$.

Figura 6.3:



6.2 Teorema do Valor Médio

Um dos mais importantes teoremas do Cálculo é o Teorema do Valor Médio que afirma que se uma função f é contínua em um intervalo fechado $[a,\, b]$ e derivável no seu interior, então existe um ponto c do seu interior tal que

$$f'(c) \;=\; \frac{f(b) - f(a)}{b - a}.$$

Geometricamente, isso significa que é possível construir uma reta tangente ao gráfico de f em $(c,\, f(c))$ que seja paralela à reta que passa pelos pontos $(a,\, f(a))$ e $(b,\, f(b))$.

Exemplo 6.4 *Consideremos a função* $f(x) = x + \operatorname{sen} x$ *no intervalo* $[1,\, 3]$. *Vamos determinar inicialmente um* c *a que se refere o Teorema do Valor Médio. O cálculo desse* c *em geral só pode ser feito de forma aproximada.*

```
> a := 1.0: b := 3.0:
> f := x -> x + sin(x):
> m := (f(b) - f(a))/(b - a):
> eq := D(f)(c) = m;
```

$$eq := 1 + \cos(c) = .6498245115$$

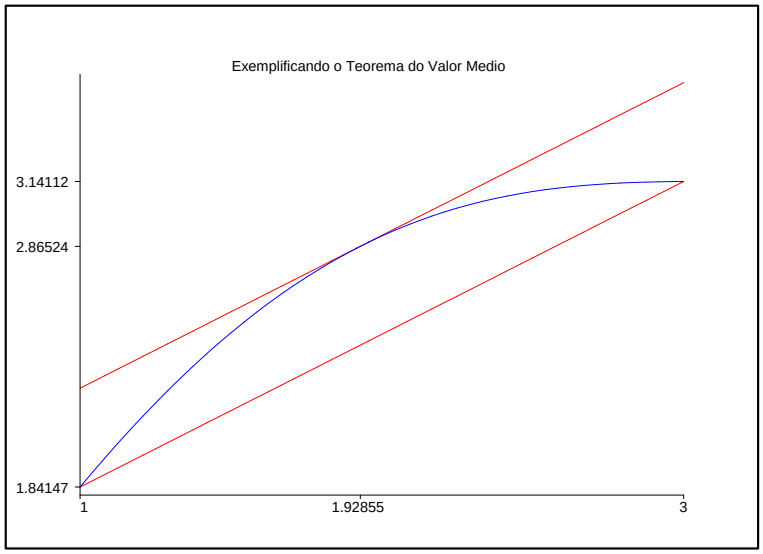
```
> c := fsolve(eq);
```

$$c := 1.928554775$$

Agora, construímos o gráfico de f, *da reta* $y = f(a) + m(x - a)$ *que passa por* $(a, f(a))$ *e por* $(b, f(b))$ *e da sua paralela* $y = f(c) + m(x - c)$ *que passa por* $(c, f(c))$.

```
> fa := f(a): fb := f(b): fc := f(c):
> opcoes := x=a..b, xtickmarks=[a, b, c],
>             ytickmarks=[fa,fb,fc], labels=[‘ ‘, ‘ ‘]:
> graf1 := plot(f(x), opcoes, thickness=2, color=blue,
>           title = "Exemplificando o Teorema do Valor Médio"):
> graf2 := plot(f(a) + (x - a)*m, opcoes):
> graf3 := plot(f(c) + (x - c)*m, opcoes):
> plots[display](graf1, graf2, graf3);
```

Figura 6.4:



6.3 Problemas de máximos e mínimos

Os problemas que consistem no cálculo de máximos ou mínimos de funções são muito freqüentes e podem ser associados a muitas situações de natureza prática. Nesta seção resolvemos alguns problemas desse tipo usando comandos do pacote `student`.

Os comandos `maximize(`*f(x), x=a..b*`)` e `minimize(`*f(x), x=a..b*`)` do pacote `student` calculam o valor máximo e o valor mínimo de $f(x)$ no intervalo $[a, b]$, respectivamente. Se for acrescentado um terceiro parâmetro igual a `location`, então, além do valor máximo ou mínimo, são mostrados também os pontos do domínio onde eles ocorrem.

Exemplo 6.5 *Calcular os máximos e mínimos locais da função* $y = x^3 - 24x + 40$ *no intervalo* $[-3, 3]$.

```
> with(student):
> y := x^3 - 24*x + 40;
```

$$y := x^3 - 24x + 40$$

```
> maximize(y, x=-3..3, location);
```

$$40 + 32\sqrt{2},\ \{\left[\{x = -2\sqrt{2}\},\ 40 + 32\sqrt{2}\right]\}$$

```
> minimize(y, x=-3..3, location);
```

$$40 - 32\sqrt{2},\ \{\left[\{x = 2\sqrt{2}\},\ 40 - 32\sqrt{2}\right]\}$$

Exemplo 6.6 *Calcular os máximos locais da função seno no intervalo* $[-10, 10]$ *com suas respectivas localizações:*

```
> with(student): maximize(sin(x), x=-10..10, location);
```

$$1, \{\left[\{x = -\frac{3}{2}\pi\}, 1\right], \left[\{x = \frac{5}{2}\pi\}, 1\right], \left[\{x = \frac{1}{2}\pi\}, 1\right]\}$$

Agora calculando os mínimos locais da função $\cos(x) + \dfrac{\cos(2x)}{2}$ *no intervalo* $[-6, 6]$*:*

```
> minimize( cos(x) + cos(2*x)/2, x=-6..6, location);
```

$$-\frac{3}{4}, \{\left[\{x = -\frac{4}{3}\pi\}, -\frac{3}{4}\right], \left[\{x = -\frac{2}{3}\pi\}, -\frac{3}{4}\right], \left[\{x = \frac{4}{3}\pi\}, -\frac{3}{4}\right], \left[\{x = \frac{2}{3}\pi\}, -\frac{3}{4}\right]\}$$

Observe agora que se não for fornecido o parâmetro opcional `location` *ao comando* `minimize`*, então é mostrado apenas o valor mínimo, sem a localização do mesmo no domínio da função.*

```
> minimize(cos(x) + cos(2*x)/2, x=-6..6);
```

$$-\frac{3}{4}$$

Exemplo 6.7 *Seja* $f(x) = \dfrac{2x}{1+x^2}$*. Vamos determinar passo a passo máximos ou mínimos locais de* f*.*

```
> f := x -> 2*x/(1 + x^2);
```

$$f := x \to 2\frac{x}{1+x^2}$$

Inicialmente determinamos os pontos críticos de f*. Para isso, resolvemos a equação* $f'(x) = 0$*. Denominamos* a *e* b *as raízes encontradas e escolhemos* $a < b$*.*

```
> s := solve(D(f)(x) = 0);   # poderia ser simplesmente solve(D(f)(x))
```

$$s := -1,\ 1$$

```
> a := s[1]; b := s[2];
```

$$a := -1$$

$$b := 1$$

Agora, calculamos $f''(a)$*,* $f(a)$*,* $f''(b)$ *e* $f(b)$*.*

```
> `f''(a)` = (D@@2)(f)(a), `f(a)` = f(a);
```

$$f''(a) = 1, \quad f(a) = -1$$

```
> `f''(b)` = (D@@2)(f)(b), `f(b)` = f(b);
```

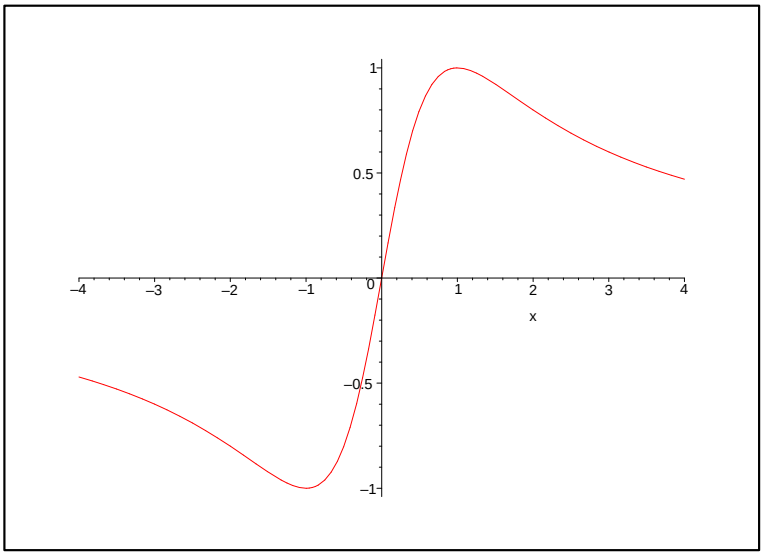
$$f''(b) = -1, \quad f(b) = 1$$

Como $f''(a) > 0$ *chegamos à conclusão de que* $a = -1$ *é um ponto de mínimo local. Como* $f''(b) < 0$*, temos que* $b = 1$ *é um máximo local.*

Para encerrar este exemplo, construímos o gráfico de f *(Figura 6.5) para visualizar os extremos locais da função.*

```
> plot(f(x), x = -4..4);
```

Figura 6.5:



6.4 Somas de Riemann

Consideremos uma função $f : [a,b] \longrightarrow \mathbb{R}$ e uma subdivisão do intervalo $[a, b]$ em n partes $[x_{i-1}, x_i]$, $i = 1, 2, \cdots, n$, tais que $a = x_0 < x_1 < x_2 < \cdots < x_n = b$. Sejam $\Delta x_i = x_i - x_{i-1}$ o comprimento e c_i um ponto qualquer do i-ésimo subintervalo. A soma $f(c_1)\Delta x_1 + f(c_2)\Delta x_2 + \cdots f(c_n)\Delta x_n = \sum_{i=1}^{n} f(c_i)\Delta x_i$ é chamada *soma de Riemann*. Quando f é positiva em $[a,b]$, $n \to \infty$ e os comprimentos $\Delta x_i \to 0$, para todo i, então a soma de Riemann tende à área da região delimitada pelo gráfico de $y = f(x)$, pelas retas $x = a$, $x = b$ e pelo eixo dos x. Essa área é numericamente igual à integral de $f(x)$ no intervalo $[a,b]$.

O pacote `student` possui 6 funções relacionadas com as somas de Riemann de uma função $f(x)$ em um intervalo $[a,b]$:

leftsum(f(x), x=a..b, n) Forma inercial da soma de Riemann com n subintervalos de comprimentos iguais e escolhe cada c_i como sendo a extremidade esquerda de cada subintervalo.

leftbox(f(x), x=a..b, n) Constrói um gráfico relacionado com o `leftsum`.

middlesum(f(x), x=a..b, n) Forma inercial da soma de Riemann com n subintervalos de comprimentos iguais e escolhe cada c_i como sendo o ponto médio de cada subintervalo.

middlebox(f(x), x=a..b, n) Constrói um gráfico relacionado com o `middlesum`.

rightsum(f(x), x=a..b, n) Forma inercial da soma de Riemann com n subintervalos de comprimentos iguais e escolhe cada c_i como sendo a extremidade direita de cada subintervalo.

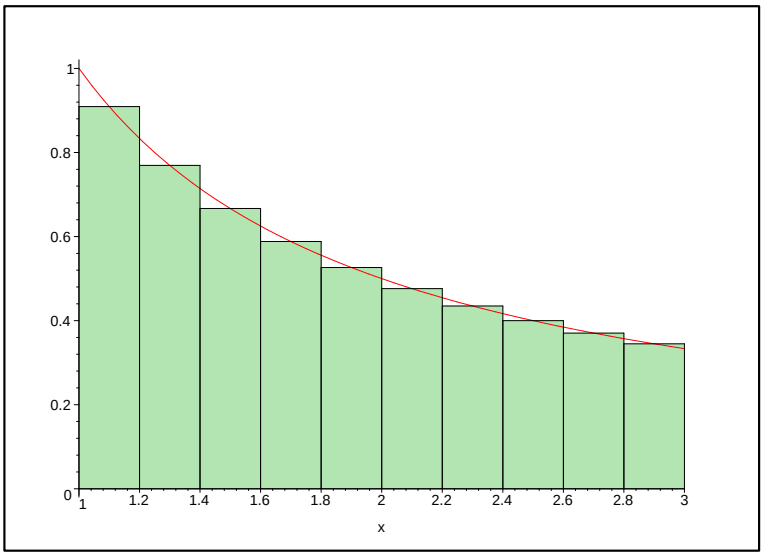
rightbox(f(x), x=a..b, n) Constrói um gráfico relacionado com o `rightsum`.

Exemplo 6.8 *Seja $f : [1,3] \longrightarrow \mathbb{R}$ definida por $f(x) = 1/x$. Inicialmente, construímos o gráfico e calculamos o valor da soma de Riemann de f com 10 subintervalos, escolhendo*

o ponto médio em cada subintervalo. Observe que o resultado obtido é próximo de $\ln(3)$ *que é o valor exato dessa área.*

```
> with(student):
> middlebox(1/x, x=1..3, 10);
```

Figura 6.6:




```
> s := middlesum(1/x, x=1..3, 10);
```

$$s := \frac{1}{5}\left(\sum_{i=0}^{9} \frac{1}{\frac{11}{10}+\frac{1}{5}i}\right)$$

```
> value(s);
```

$$\frac{159708887504}{145568097675}$$

```
> evalf(s);
```

$$1.097142094$$

```
> `ln(3) ` = ln(3.0);
```

$$\ln(3) = 1.098612289$$

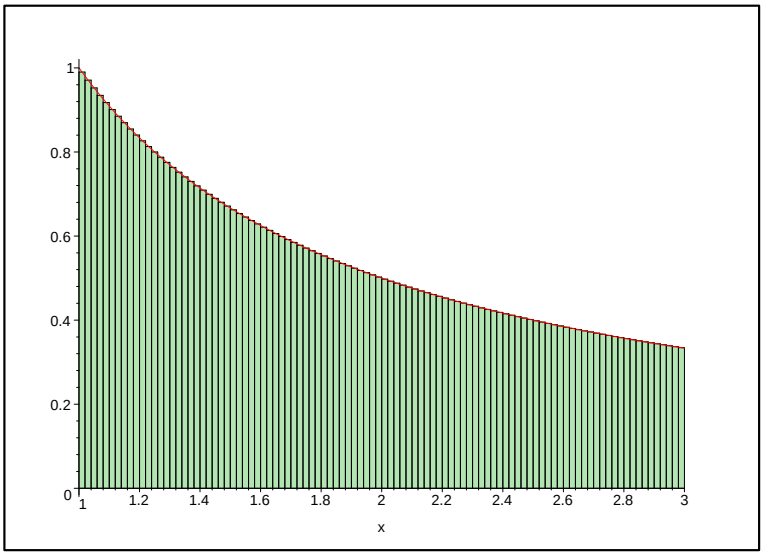
Se aumentarmos a quantidade de subintervalos para 100, a soma obtida fica ainda mais próxima de $\ln(3)$.

```
> s2 := middlesum(1/x, x=1..3, 100):
> evalf(s2);
```

$$1.098597475$$

```
> middlebox(1/x, x=1..3, 100);
```

Figura 6.7:



Quando $n \to \infty$ o limite da soma de Riemann com n subintervalos é igual a $\ln(3)$, *ou seja, é igual à integral de* $1/x$ *no intervalo* $[1, 3]$:

```
> Limit(middlesum(1/x, x=1..3, n), n=infinity);
```

$$\lim_{n\to\infty} 2\frac{\displaystyle\sum_{i=0}^{n-1} \frac{1}{1+\frac{2(i+1/2)}{n}}}{n}$$

```
> S := value(%);
```

$$S := \ln(3)$$

Exemplo 6.9 *Ainda com relação à função* $f(x) = 1/x$ *no intervalo* $[1, 3]$, *definimos e construímos o gráfico de uma soma de Riemann dessa função com 10 subintervalos, escolhendo a extremidade direita de cada subintervalo. O gráfico é o da Figura 6.8.*

```
> rightbox(1/x, x=1..3, 10);
```

```
> rightsum(1/x, x=1..3, 10);
```

$$\frac{1}{5}\left(\sum_{i=1}^{10} \frac{1}{1+\frac{1}{5}i}\right)$$

```
> evalf(%);
```

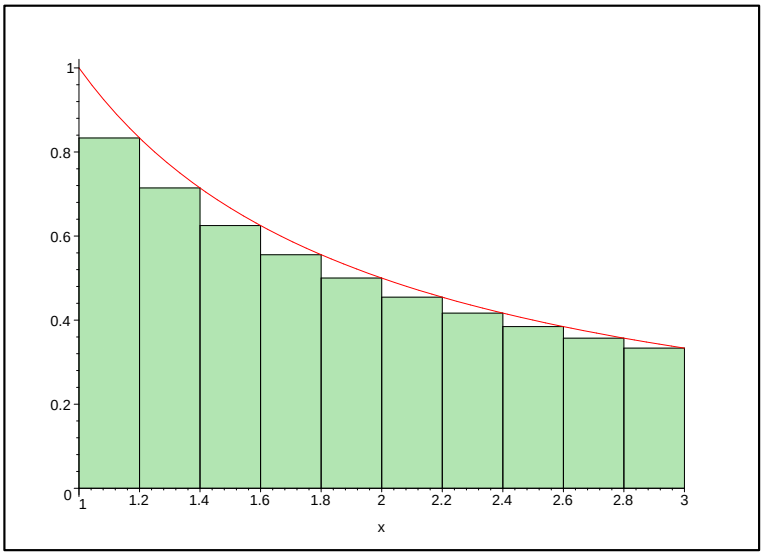
$$1.034895660$$

6.5 Regras de integração

As regras de integração costumam fazer parte dos cursos introdutórios de Cálculo. É comum encontrarmos problemas trabalhosos, com muitos cálculos, envolvendo tais regras.

Nesta seção resolvemos passo a passo algumas integrais usando regras básicas como integração por substituição, integração por partes e integração de funções racionais através da separação em frações parciais.

Figura 6.8:



6.5.1 Integração por substituição

O pacote `student` traz o comando `changevar(`*f(x) = u, I, u*`)` que faz a mudança de variável x para u em uma integral I. É recomendável que a integral esteja na sua forma inercial, para não ser calculada de imediato.

Exemplo 6.10 *Seja* $G = \int (3x^2 - 2)e^{x^3-2x+1}dx$. *Vamos calculá-la usando a mudança de variável* $u = x^3 - 2x + 1$. *Neste caso, o cálculo* $du = f'(x)dx$ *é feito automaticamente.*

```
> with(student):
> G := Int((3*x^2 - 2)*exp(x^3 - 2*x + 1), x);
```

$$G := \int (3x^2 - 2)e^{(x^3-2x+1)}dx$$

```
> G1 := changevar(x^3 - 2*x + 1 = u, G, u);
```

$$G1 := \int e^u du$$

```
> G2 := value(G1) + C;
```

$$G2 := e^u + C$$

A integral está calculada na variável u *e agora basta usar o comando* `subs` *para substituir* u *por* x.

```
> G = subs(u = x^3 - 2*x + 1, G2);
```

$$\int (3x^2 - 2)e^{(x^3-2x+1)}dx = e^{(x^3-2x+1)} + C$$

Fica calculada assim, passo a passo, a integral G *dada. É claro que o* C *que foi acrescentado no final dos cálculos representa uma constante real qualquer.*

Exemplo 6.11 *Neste exemplo, calculamos* $\int x(x+1)^{2002}dx$. *Essa integral é interessante porque o Maple não consegue calculá-la "sozinho", sem ajuda do usuário. Ele tenta desenvolver o binômio e, com isso, termina encontrando algum problema de falta de memória. Mas, ao solicitar que seja feita a substituição* $x + 1 = u$, *ele conclui o cálculo imediatamente.*

```
> H := Int( x*(x + 1)^2002, x);
```

$$H := \int x(x+1)^{2002}dx$$

```
> H1 := changevar(x + 1 = u, H, u);
```

$$H1 := \int (-1+u)u^{2002}du$$

```
> H2 := value(H1) + C;
```

$$H2 := \frac{1}{2004}u^{2004} - \frac{1}{2003}u^{2003} + C$$

```
> H = subs(u = x + 1, H2);
```

$$\int x(x+1)^{2002}dx = \frac{1}{2004}(x+1)^{2004} - \frac{1}{2003}(x+1)^{2003} + C$$

onde C é uma constante qualquer.

Exemplo 6.12 *O sucesso da integração por substituição depende da escolha correta da mudança de variável. Uma boa mudança de variável deve simplificar a integral. Neste exemplo calculamos a integral* $\int \frac{\cos(x)}{\sqrt{1+\operatorname{sen}(x)}}dx$. *Inicialmente, fazemos uma mudança inadequada* $\cos(x) = u$ *para ver o que acontece.*

```
> J := Int(cos(x)/sqrt(1 + sin(x)), x);
```

$$J := \int \frac{\cos(x)}{\sqrt{1+\sin(x)}}dx$$

```
> changevar(cos(x) = u, J, u);
```

$$\int -\frac{u}{\sqrt{1+\sqrt{1-u^2}}\sqrt{1-u^2}}du$$

Observando o resultado mostrado vemos que a integral obtida fica mais complicada do que a integral dada. Por isso, desistimos dessa mudança de variável e tentamos usar outra: $sen(x) = u$.

```
> changevar(sin(x) = u, J, u);
```

$$\int \frac{1}{\sqrt{1+u}}du$$

Agora, sim, a integral ficou simplificada ao ponto de poder ser calculada diretamente (com um comando `value`*).*

```
>  value(%);
```

$$2\sqrt{1+u}$$

Usamos o comando `subs` *para voltar à variável x inicial:*

```
> J = subs(u = sin(x), %);
```

$$\int \frac{\cos(x)}{\sqrt{1+\sin(x)}}dx = 2\sqrt{1+\sin(x)}$$

Agora, derivando ambos os membros da igualdade anterior para conferir se os cálculos foram feitos corretamente.

```
> diff(%, x);
```

$$\frac{\cos(x)}{\sqrt{1+\sin(x)}} = \frac{\cos(x)}{\sqrt{1+\sin(x)}}$$

Exemplo 6.13 *Vamos calcular a integral da função racional* $x/(x^2+7x+13)$. *Como o polinômio do* 2^o *grau que aparece no denominador não tem raízes reais, iniciamos fazendo um completamento de quadrados dele. Fazemos isso apelando para o comando* `completesquare` *do pacote* `student`.

```
> with(student):
> K := Int(x/(x^2 + 7*x + 13), x);
```

$$K := \int \frac{x}{x^2+7x+13}dx$$

Substituímos o denominador pelo resultado do completamento:

```
> subs(x^2 + 7*x + 13 = completesquare(x^2 + 7*x + 13), K);
```

$$\int \frac{x}{(x+\frac{7}{2})^2+\frac{3}{4}}dx$$

Observando o quadrado completado no denominador ficamos sabendo qual a substituição de variável a ser utilizada: $x+\frac{7}{2} = t\sqrt{\frac{3}{4}}$

```
> changevar(x + 7/2 = t*sqrt(3/4), %, t);
```

$$\int \frac{1}{2}\frac{(-\frac{7}{2}+\frac{1}{2}\sqrt{3}t)\sqrt{3}}{\frac{3}{4}t^2+\frac{3}{4}}dt$$

Simplificamos e expandimos o último resultado mostrado, até obtermos integrais que sejam suficientemente simples para poderem ser calculadas diretamente.

```
> simplify(%);
```

$$\frac{1}{3}\sqrt{3}\int \frac{-7+\sqrt{3}t}{t^2+1}dt$$

```
> expand(%);
```

$$-\frac{7}{3}\sqrt{3}\int \frac{1}{t^2+1}dt + \int \frac{t}{t^2+1}dt$$

```
> value(%);
```

$$-\frac{7}{3}\sqrt{3}\arctan(t) + \frac{1}{2}\ln(t^2+1)$$

Finalmente, substituímos $t = \sqrt{\frac{4}{3}}\left(x+\frac{7}{2}\right)$ *para voltarmos à variável original.*

```
> subs(t = sqrt(4/3)*(x + 7/2), %);
```

$$-\frac{7}{3}\sqrt{3}\arctan(\frac{2}{3}\sqrt{3}(x+\frac{7}{2})) + \frac{1}{2}\ln(\frac{4}{3}(x+\frac{7}{2})^2+1)$$

```
> `K` = normal(%);
```

$$K = -\frac{7}{3}\sqrt{3}\arctan(\frac{1}{3}\sqrt{3}(2x+7)) + \frac{1}{2}\ln(\frac{4}{3}x^2+\frac{28}{3}x+\frac{52}{3})$$

6.5.2 Integração por partes

O pacote `student` possui o comando `intparts(I, u)` que calcula a integral I por partes, considerando o segundo parâmetro como sendo a função u da fórmula da integração por partes $\int udv = uv - \int vdu$.

O critério da escolha da função u deve ser o de que a integral obtida após a aplicação da fórmula seja mais simples de calcular do que a integral original. Uma má escolha dessa função leva a resultados desastrosos.

Exemplo 6.14 *Neste exemplo, o Maple mostra uma variação da fórmula da integração por partes.*

```
> with(student):
> Int(U(x)*Diff(V(x), x), x) = intparts(Int(U(x)*Diff(V(x), x), x), U(x));
```

$$\int U(x)\frac{\partial}{\partial x}V(x)dx = U(x)V(x) - \int \left(\frac{\partial}{\partial x}U(x)\right)V(x)dx$$

Exemplo 6.15 *Seja $F = \int x^3 \ln x dx$. Se quisermos usar integração por partes, temos pelo menos duas opções para a escolha da função u: $u = x^3$ ou $u = \ln x$. Testamos inicialmente $u = \ln x$.*

```
> F := Int(x^3*ln(x), x);
```

$$F := \int x^3 \ln(x)dx$$

```
> intparts(F, ln(x));
```

$$\frac{1}{4}\ln(x)x^4 - \int \frac{1}{4}x^3 dx$$

A nova integral obtida é mais simples do que a integral dada originalmente. Logo, $u = \ln x$ foi uma boa escolha. Calculamos agora a integral diretamente:

```
> value(%);
```

$$\frac{1}{4}\ln(x)x^4 - \frac{1}{16}x^4$$

Veja agora o que aconteceria se no cálculo de F tivéssemos escolhido $u = x^3$:

```
> intparts(F, x^3);
```

$$x^3(x\ln(x) - x) - \int 3x^2(x\ln(x) - x)dx$$

Observe que a integral obtida é mais complicada do que a original.

Exemplo 6.16 *Agora vamos calcular $\int x^2 \cos x dx$. Podemos escolher $u = x^2$ ou $u = \cos x$. Inicialmente, vamos escolher $u = \cos x$:*

```
> G := Int(x^2*cos(x), x);
```

$$G := \int x^2 \cos(x)dx$$

```
> intparts(G, cos(x));
```

$$\frac{1}{3}\cos(x)x^3 - \int -\frac{1}{3}\sin(x)x^3 dx$$

Vemos que a integral resultante é mais complicada. Logo, $u = \cos x$ é uma má escolha.

Agora, escolhamos $u = x^2$:

```
> G0 := intparts(G, x^2);
```

$$G0 := x^2\sin(x) - \int 2x\sin(x)dx$$

A integral obtida é mais simples do que a original, portanto foi feita uma boa escolha da função u. No entanto, ainda não é uma integral tão simples, devendo ser usada integração por partes novamente para ela poder ser calculada.

```
> G1 := op(1, G0); G2 := op(2, G0);
```

$$G1 := x^2\sin(x)$$

$$G2 := -\int 2x\sin(x)dx$$

```
> G2 := intparts(G2, 2*x);
```

$$G2 := 2x\cos(x) + \int -2\cos(x)dx$$

A integral assim obtida pode ser calculada diretamente:

```
> G2 := value(G2);
```

$$G2 := 2x\cos(x) - 2\sin(x)$$

```
> G = G1 + G2;
```

$$\int x^2\cos(x)dx = x^2\sin(x) + 2x\cos(x) - 2\sin(x)$$

6.5.3 Frações parciais

Com a opção `parfrac`, o comando `convert(`*R(x),* `parfrac`*, x)* pode ser usado para escrever a função racional $R(x)$ como soma de frações parciais na variável x, fornecida como terceiro parâmetro.

Exemplo 6.17 *Vamos escrever as funções racionais* $\dfrac{x+3}{x^3+8x^2+7x}$ *e* $\dfrac{1}{x^2-5x+6}$ *como somas de frações parciais.*

```
> F1 := (x+3)/(x^3 + 8*x^2 + 7*x);
```

$$F1 := \frac{x+3}{x^3+8x^2+7x}$$

```
> convert(F1, parfrac, x);
```

$$\frac{3}{7}\frac{1}{x} - \frac{2}{21}\frac{1}{x+7} - \frac{1}{3}\frac{1}{x+1}$$

```
> 1/(x^2 - 5*x + 6) = convert( 1/(x^2 - 5*x + 6), parfrac, x);
```

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

Exemplo 6.18 *Neste exemplo calculamos a integral*

$$P = \int \frac{10x^5 + 43x^4 - 9x^3 + 138x^2 - 331x - 189}{(x^2+9)(x^2+3x-10)^2} dx$$

usando o método da separação em frações parciais.

```
> P := Int((10*x^5 + 43*x^4 - 9*x^3 + 138*x^2 - 331*x - 189)/
>           ((x^2 + 9)*(x^2 + 3*x - 10)^2), x);
```

$$P := \int \frac{10x^5 + 43x^4 - 9x^3 + 138x^2 - 331x - 189}{(x^2+9)(x^2+3x-10)^2} dx$$

```
> with(student):
> R := integrand(P);
```

$$R := \frac{10x^5 + 43x^4 - 9x^3 + 138x^2 - 331x - 189}{(x^2+9)(x^2+3x-10)^2}$$

```
> R := convert(R, parfrac, x);
```

$$R := \frac{1}{(x+5)^2} + \frac{5}{x+5} + \frac{1}{(x-2)^2} + \frac{3}{x-2} + \frac{2x}{x^2+9}$$

Concluída a separação em frações parciais, calculamos as integrais de cada fração assim obtida.

```
> intR := Int(R, x);
```

$$R := \int \frac{1}{(x+5)^2} + \frac{5}{x+5} + \frac{1}{(x-2)^2} + \frac{3}{x-2} + \frac{2x}{x^2+9} dx$$

```
> expand(intR);
```

$$\int \frac{1}{(x+5)^2} dx + 5\int \frac{1}{x+5} dx + \int \frac{1}{(x-2)^2} dx + \int \frac{1}{x-2} dx + \int \frac{3}{x-2} dx + 2\int \frac{x}{x^2+9} dx$$

```
> `P` = value(%);
```

$$P = -\frac{1}{x+5} + 5\ln(x+5) - \frac{1}{x-2} + 3\ln(x-2) + \ln(x^2+9)$$

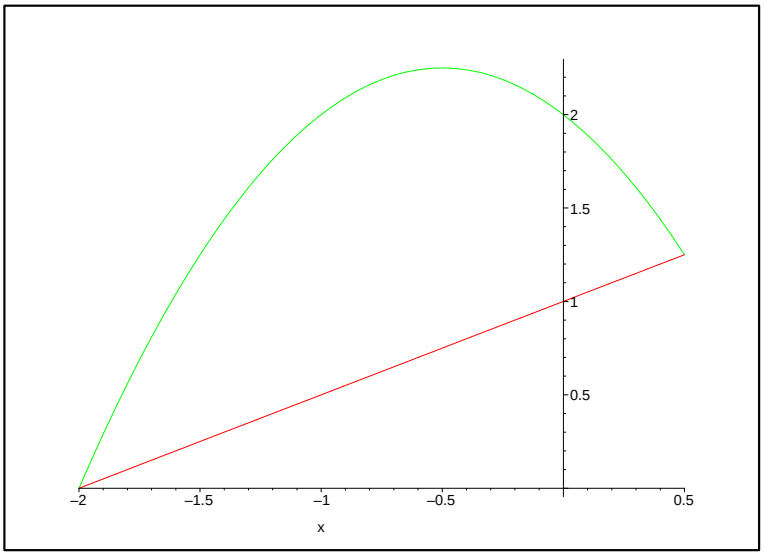
6.6 Cálculo de áreas

Uma das aplicações da integral definida é o cálculo da área de regiões do plano delimitadas por gráficos de funções. A resolução desse tipo de problema requer que sejam determinados os pontos de interseção dos gráficos (um caso para usar um comando `solve` ou `fsolve`) e que sejam conhecidas as posições relativas das curvas envolvidas, ou seja, qual a curva que aparece em cima e qual a que aparece na parte de baixo da região. Isso pode ser decidido facilmente observando-se os gráficos construídos em um mesmo sistema de eixos coordenados (com um comando `plot`).

Exemplo 6.19 *Queremos calcular a área delimitada pelos gráficos de* $f(x) = x/2 + 1$ *e por* $g(x) = (2+x)(1-x)$.

```
> f := x/2 + 1; g := (2 + x)*(1 - x);
```

Figura 6.9:



$$f := 1/2x + 1$$

$$g := (2 + x)(1 - x)$$

Determinamos os pontos de interseção dos gráficos:

```
> S := solve(f = g);
```

$$S := \frac{1}{2}, -2$$

```
> b := S[1]; a := S[2];
```

$$b := 1/2$$

$$a := -2$$

Temos assim que os gráficos interceptam-se nos pontos $a = -2$ *e* $b = 1/2$ *do domínio comum a essas funções.*

Construímos os gráficos de f *e de* g *no intervalo* $[a, b]$*. O resultado é mostrado na Figura 6.9.*

```
> plot([f, g], x=a..b, thickness=2);
```

Na Figura 6.9, vemos que o gráfico da função g *aparece acima do gráfico de* f*. Logo a integral a ser calculada é a da diferença* $g - f$ *no intervalo* $[a, b]$*, que corresponde ao valor numérico* A *da área desejada.*

```
> A = int(g - f, x = a..b);
```

$$A = \frac{125}{48}$$

6.7 Geometria Analítica plana

O Maple possui um interessante pacote chamado `geometry` com mais de 100 comandos relacionados com itens de geometria euclidiana plana.

```
> with(geometry);
```

[*Apollonius, Appolonius, AreCollinear, AreConcurrent, AreConcyclic, AreConjugate, AreHarmonic, AreOrthogonal, AreParallel, ArePerpendicular, AreSimilar, AreTangent, CircleOfSimilitude, CrossProduct, CrossRatio, DefinedAs, Equation, EulerCircle, EulerLine, ExteriorAngle, ExternalBisector, FindAngle, GergonnePoint, GlideReflection, HorizontalCoord, HorizontalName, InteriorAngle, IsEquilateral, IsOnCircle, IsOnLine, IsRightTriangle, MajorAxis, MakeSquare, MinorAxis, NagelPoint, OnSegment, ParallelLine, PedalTriangle, PerpenBisector, PerpendicularLine, Polar, Pole, RadicalAxis, RadicalCenter, RegularPolygon, RegularStarPolygon, SensedMagnitude, SimsonLine, SpiralRotation, StretchReflection, StretchRotation, TangentLine, VerticalCoord, VerticalName, altitude, apothem, area, asymptotes, bisector, center, centroid, circle, circumcircle, conic, convexhull, coordinates, detail, diagonal, diameter, dilatation, directrix, distance, draw, dsegment, ellipse, excircle, expansion, foci, focus, form, homology, homothety, hyperbola, incircle, inradius, intersection, inversion, line, medial, median, method, midpoint, orthocenter, parabola, perimeter, point, powerpc, projection, radius, randpoint, reciprocation, reflection, rotation, segment, sides, similitude, slope, square, stretch, tangentpc, translation, triangle, vertex, vertices*]

6.7.1 Pontos e retas

No pacote `geometry` um ponto $P = (a, b)$ é definido na forma `point(P, a, b)`. Depois de usado um comando `point`, o ponto assim definido pode ser fornecido como parâmetro em outros comandos.

Uma reta L pode ser definida de duas maneiras com um comando `line`:

- Com sua equação e a lista de variáveis: `line(`*L, equação*`, [`*variáveis*`])`
- Com dois de seus pontos *A*, *B* e as variáveis: `line(`*L*`, [`*A, B*`], [`*variáveis*`])`

A lista de variáveis é opcional. Se ela não for definida logo, o programa perguntará por ela quando for preciso.

Entre as muitas funções do pacote `geometry` que exigem os objetos `point` ou `line` como parâmetros, vamos citar algumas:

coordinates(P) Coordenadas do ponto **P**

Equation(L) Equação da reta **L**

slope(L) Declividade da reta **L**

detail(P) Informações detalhadas sobre o ponto **P**

detail(L) Informações detalhadas sobre a reta **L**

distance(P, Q) Distância entre os pontos **P** e **Q**

distance(P, L) Distância entre o ponto **P** e a reta **L**

PerpendicularLine(L2, P, L1) Define a reta L2 que passa pelo ponto **P** e é perpendicular à reta **L1**

ParallelLine(L4, Q, L3) Define a reta **L4** que passa pelo ponto **Q** e é paralela à reta **L3**

draw(O) Desenha o objeto **O**

draw([O1, O2, ...]) Desenha uma lista de objetos [**O1**, **O2**, ...]

Exemplo 6.20 *Vamos definir para o programa o ponto* $P = (5, -6)$ *e mostrar suas coordenadas com um comando* `coordinates`.

```
> with(geometry):
> point(P, 5, -6);
```

$$P$$

```
> coordinates(P);
```

$$[5, -6]$$

Agora, definimos a reta $r1$ *que passa pelos pontos* $A = (1, 1)$ *e* $B = (-3, 2)$*, calculamos sua equação e declividade.*

```
> line(r1, [point(A, 1, 1), point(B, -3, 2)], [x, y]);
```

$$r1$$

```
> Equation(r1);
```

$$5 - x - 4y = 0$$

```
> slope(r1);   # declividade de r1
```

$$-\frac{1}{4}$$

Definimos a reta r2 que passa por P e é perpendicular a r1, calculamos sua declividade e equação.

```
> PerpendicularLine(r2, P, r1);
```

$$r2$$

```
> slope(r2); # declividade de r2
```

$$4$$

```
> Equation(r2);
```

$$26 - 4x + y = 0$$

Definimos a reta r3 que passa por P e é paralela a r1 e calculamos sua equação.

```
> ParallelLine(r3, P, r1);
```

$$r3$$

```
> Equation(r3);
```

$$-19 - x - 4y = 0$$

Calculamos a distância do ponto P à reta r1.

```
> distance(P, r1);
```

$$\frac{24}{17}\sqrt{17}$$

6.7.2 Circunferências

Uma circunferência C pode ser definida usando-se o comando `circle` de uma das quatro maneiras mostradas a seguir:

- Com três pontos P, Q e R e lista de variáveis: `circle(`C`, [`P`, `Q`, `R`], [`*variáveis*`])`
- Com dois de seus pontos P e Q diametralmente opostos e a lista de variáveis: `circle(`C`, [`P`, `Q`], [`*variáveis*`])`
- Com sua equação e a lista de variáveis: `circle(`C`, `*equação*`, [`*variáveis*`])`
- Com o centro O, raio r e *variáveis*: `circle(`C`, [`O`, `r`], [`*variáveis*`])`

A lista de variáveis é opcional. Tem também um quarto parâmetro opcional com o nome do centro da circunferência que deve ser escrito na seguinte forma: `centername =` *NomeCentro*.

Citemos algumas das funções do pacote `geometry` relacionadas com o comando `circle`:

center(C) Centro da circunferência **C**

radius(C) Raio da circunferência **C**

area(C) Área do círculo **C**

Equation(C) Equação da circunferência **C**

detail(C) Informações detalhadas sobre a circunferência **C**

Exemplo 6.21 *Neste exemplo é definida uma circunferência* $C1$ *de centro* $O1 = (-1, 7)$ *e raio* 4 *sem a lista de variáveis. O Maple pergunta pela lista na hora de mostrar sua equação.*

```
> with(geometry):
> circle(C1, [point(O1, -1, 7), 4]);
```

$$C1$$

```
> Equation(C1);
  enter name of the horizontal axis> m;
  enter name of the vertical axis> n;
```

$$m^2 + 34 + n^2 + 2m - 14n = 0$$

Equivalentemente, essa mesma circunferência poderia ter sido definida na forma:

```
> point(O1, -1, 7);
> circle(C1, [O1, 4], [m, n]):
```

Depois de definida, podemos obter alguns dados sobre $C1$.

```
> radius(C1);
```

$$4$$

```
> center(C1);
```

$$O1$$

```
> coordinates(center(C1));
```

$$[-1,\ 7]$$

```
> area(C1);
```

$$16\pi$$

Se a lista de variáveis $[var1, var2]$ de um determinado objeto gráfico não tiver sido definida, ela poderá ser definida com os seguintes comandos: `__EnvHorizontalName =` *var1* e `__EnvHorizontalName =` *var2*.

Exemplo 6.22 *Neste exemplo definimos uma circunferência C2 de raio* $\sqrt{2}$ *e centro* $O2 = (3, -4/5)$ *e usamos algumas funções relacionadas.*

```
> _EnvHorizontalName := x: _EnvVerticalName := y:
> circle(C2, [point(O2, 3, -4/5), sqrt(2)]):
> eq := Equation(C2);
```

$$eq := x^2 + \frac{191}{25} + y^2 - 6x + \frac{8}{5}y = 0$$

```
> area(C2);
```

$$2\pi$$

```
> detail(C2);
```

name of the object: C2
form of the object: circle2d
name of the center: O2
coordinates of the center: [3, -4/5]
radius of the circle: 2^(1/2)
*equation of the circle: x^2+191/25+y^2-6*x+8/5*y = 0*

Agora usamos o comando `completesquare` *do pacote* `student` *para completar os quadrados referentes à variável* x *na equação da circunferência:*

```
> with(student): completesquare(eq, x);
```

$$(x-3)^2 - \frac{34}{25} + y^2 + \frac{8}{5}y = 0$$

E agora fazemos o mesmo com relação à variável y*:*

```
> completesquare(%, y);
```

$$(y + \frac{4}{5})^2 - 2 + (x-3)^2 = 0$$

Somamos 2 *aos dois membros da equação (equivale a passar o* 2 *para o segundo membro)*

```
> 2 + lhs(%) = 2 + rhs(%);
```

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

Obtivemos assim a equação da circunferência $C2$ *escrita em um formato usual.*

Exemplo 6.23 *Seja* $C3$ *a circunferência cuja equação é* $x^2+y^2-3x+7y-1 = 0$*. Calcule as coordenadas do seu centro e o seu raio.*

```
> with(geometry):
> circle(C3, x^2 + y^2 - 3*x + 7*y - 1 = 0, [x, y]);
```

$$C3$$

```
> coordinates(center(C3));
```

$$\left[\frac{3}{2}, -\frac{7}{2}\right]$$

```
> radius(C3);
```

$$\frac{1}{2}\sqrt{31}\sqrt{2}$$

```
> Equation(C3);
```

$$x^2 + y^2 - 3x + 7y - 1 = 0$$

Exemplo 6.24 *Determine a interseção da circunferência C4 que passa pelos pontos* $A(5,4)$, $B(6,1)$ *e* $C(-3,-2)$ *com a reta R1 que passa pelos pontos* $D(1,2)$ *e* $E(-4,1)$.

```
> with(geometry):
> circle(C4, [point(A,5,4), point(B,6,1), point(C,-3,-2)], [x,y]):
> line(R1, [point(D, 1, 2), point(E, -4, 1)], [x,y]):
> EqCirc := Equation(C4);
```

$$EqCirc := x^2 - 23 + y^2 - 2x - 2y = 0$$

```
> EqReta := Equation(R1);
```

$$EqReta := 9 + x - 5y = 0$$

```
> solve({EqCirc, EqReta});
```

$$\{y = 1,\, x = -4\},\, \{y = \frac{38}{13},\, x = \frac{73}{13}\}$$

Obtivemos então uma interseção formada pelo conjunto de pontos

$$\left\{(-4,\, 1),\, (\frac{73}{13},\, \frac{38}{13})\right\}$$

6.7.3 Cônicas

No pacote `geometry`, cada cônica pode ser definida de várias maneiras.

Parábola Uma parábola `P`, da qual sejam conhecidos cinco de seus pontos `A, B, C, D, E,` ou seu foco `F` ou seu vértice `V` ou sua diretriz `R` ou sua *equação*, pode ser definida de um dos seguintes modos:

- `parabola(P, [A, B, C, D, E], [`*variáveis*`])`
- `parabola(P, [focus = F, vertex = V], [`*variáveis*`])`
- `parabola(P, [directrix = R, focus = F], [`*variáveis*`])`
- `parabola(P,` *equação*`, [`*variáveis*`])`

Elipse Uma elipse `EL`, da qual sejam conhecidos cinco de seus pontos `A, B, C, D, E,` ou seus focos `F1, F2` ou sua diretriz `R` ou sua excentricidade `e` ou a soma `s` dos raios focais ou o comprimento do eixo maior `EMaior` ou o comprimento do eixo menor `EMenor` ou pontos `P1, P2` extremos do eixo maior ou os pontos `Q1, Q2` extremos do eixo menor, pode ser definida de um dos seguintes modos:

- `ellipse(EL, [A, B, C, D, E], [`*variáveis*`])`
- `ellipse(EL, [directrix=R, focus=F1, eccentricity=e], [`*variáveis*`])`
- `ellipse(EL, [foci = [F1, F2], MajorAxis = EMaior], [`*variáveis*`])`
- `ellipse(EL, [foci = [F1, F2], MinorAxis = EMenor], [`*variáveis*`])`
- `ellipse(EL, [foci = [F1, F2], distance= s], [`*variáveis*`])`
- `ellipse(EL, [MajorAxis=[P1, P2], MinorAxis=[Q1, Q2], [`*variáveis*`])`
- `ellipse(EL,` *equação*`, [`*variáveis*`])`

Hipérbole Uma hipérbole `H`, da qual sejam conhecidos cinco de seus pontos `A, B, C, D, E`, ou seus focos `F1, F2` ou sua diretriz `R` ou sua excentricidade `e` ou a distância focal `DistF` ou a distância entre os vértices `DistV` ou os vértices `V1, V2` ou sua *equação*, pode ser definida de um dos seguintes modos:

- `hyperbola(H, [A, B, C, D, E], [`*variáveis*`])`
- `hyperbola(H, [directrix=R, focus=F1, eccentricity=e], [`*variáveis*`])`
- `hyperbola(H, [foci = [F1, F2], vertices = [V1, V2]], [`*variáveis*`])`
- `hyperbola(H, [foci = [F1, F2], distancev = DistV], [`*variáveis*`])`
- `hyperbola(H, [vertices = [V1, V2], distancef = DistF], [`*variáveis*`])`
- `hyperbola(H,` *equação*`, [`*variáveis*`])`

Uma cônica `G`, da qual sejam conhecidos (1) cinco de seus pontos `A, B, C, D, E` ou (2) sua diretriz `R`, foco `F` e excentricidade `e` ou (3) sua *equação*, pode ser definida com um dos seguintes comandos:

- `conic(G, [A, B, C, D, E], [`*variáveis*`])`
- `conic(G, [R, F, e], [`*variáveis*`])`
- `conic(G,` *equação*`, [`*variáveis*`])`

O comando `conic` admite inclusive os casos degenerados de cônicas como par de retas, ponto, etc.

Na elipse devemos ter a excentricidade $e < 1$, na parábola $e = 1$, e na hipérbole $e > 1$.

Depois de definida, podemos obter informações a respeito de uma cônica `G` com os seguintes comandos:

form(G) Classificação de **G**

Equation(G) Equação de **G**

detail(G) Informações detalhadas sobre **G**

focus(G) Foco de **G** (parábola)

foci(G) Focos de **G**

vertex(G) Vértice de **G** (parábola)

vertices(G) Vértices de **G**

center(G) Centro de **G**

asymptotes(G) Assíntotas de **G** (hipérbole)

directrix(G) Diretriz de **G**

MajorAxis(G) Comprimento do eixo maior (elipse)

MinorAxis(G) Comprimento do eixo menor (elipse)

Exemplo 6.25 *Neste exemplo, definimos a parábola* $y = x^2 - 5x + 6$ *como sendo um objeto geométrico* `p1` *do pacote* `geometry` *e obtemos as coordenadas do vértice, as coordenadas do foco e a equação da reta diretriz.*

```
> with(geometry):
> parabola(p1, y = x^2 - 5*x + 6, [x,y]):
> coordinates(vertex(p1));
```

$$\left[\frac{5}{2}, -\frac{1}{4}\right]$$

```
> coordinates(focus(p1));
```

$$\left[\frac{5}{2}, 0\right]$$

```
> Equation(directrix(p1));
```

$$y + \frac{1}{2} = 0$$

Exemplo 6.26 *Consideremos a hipérbole* `h1` *definida por* $xy = 1$ *e a elipse* `e1` *definida por* $\frac{x^2}{36} + \frac{y^2}{100} = 1$. *Vamos obter todas as informações básicas sobre esses objetos geométricos.*

```
> hyperbola(h1, x*y = 1, [x, y]);
```

$$h1$$

```
> detail(h1);
```

name of the object: h1
form of the object: hyperbola2d
center: [0, 0]
foci: [[-2^(1/2), -2^(1/2)], [2^(1/2), 2^(1/2)]]
vertices: [[-1, -1], [1, 1]]
*the asymptotes: [y*2^(1/2) = 0, -x*2^(1/2) = 0]*
*equation of the hyperbola: x*y-1 = 0*

```
> ellipse(e1, x^2/36 + y^2/100 = 1, [x, y]);
```

$$e1$$

```
> detail(e1);
```

name of the object: e1
form of the object: ellipse2d
center: [0, 0]
foci: [[0, -8], [0, 8]]
length of the major axis: 20
length of the minor axis: 12
*equation of the ellipse: 1/36*x^2+1/100*y^2-1 = 0*

Exemplo 6.27 *Considere a parábola com vértice $V = (-3, 4)$ e foco $F = (2, 4)$. Determine sua equação e sua diretriz.*

```
> with(geometry):
> parabola(p2, [vertex = point(V,-3,4), focus = point(F,2,4)], [x,y]);
```

$$p2$$

```
> EqPar := Equation(p2);
```

$$EqPar := -1100 + 25y^2 - 500x - 200y = 0$$

```
> EqPar/25;
```

$$-44 + y^2 - 20x - 8y = 0$$

```
> Equation(directrix(p2));
```

$$x + 8 = 0$$

Exemplo 6.28 *Determine a equação da elipse* `e3` *de focos $F_1 = (2, -3)$, $F_2 = (12, -3)$ e eixo maior de comprimento 16.*

```
> with(geometry):
> point(F1, 2, -3), point(F2, 12, -3);
```

$$F1, F2$$

```
> ellipse(e3, [foci=[F1, F2], MajorAxis=16], [x, y]);
```

$$e4$$

```
> Equation(e3);
```

$$624x^2 - 8736x + 1024y^2 + 6144y - 144 = 0$$

```
> foci(e3);
```

$$[foci\_1\_e3,\ foci\_2\_e3]$$

```
> map(coordinates, foci(e3));
```

$$[[2, -3],\ [12, -3]]$$

```
> detail(e3);
```

name of the object: e3
form of the object: ellipse2d
center: [7, -3]
foci: [[2, -3], [12, -3]]
length of the major axis: 16
*length of the minor axis: 2*39^(1/2)*
*equation of the ellipse: 624*x^2-8736*x+1024*y^2+6144*y-144 = 0*

Exemplo 6.29 *Determine a equação da elipse* `e4` *de focos $F_1 = (4, 3)$, $F_2 = (4, -5)$ e soma dos raios focais igual a 12.*

```
> with(geometry):
> point(F1, 4, 3), point(F2, 4, -5);
```

$$F1, F2$$

```
> ellipse(e4, [foci=[F1, F2], distance=12], [x, y]);
```

$$e4$$

```
> eq4 := Equation(e4);
```

$$eq4 := 576x^2 - 4608x + 320y^2 + 640y - 1984 = 0$$

Podemos usar o comando `completesquare(`*equação*`,` *variável*`)` *do pacote* `student` *para completar os quadrados e escrever a equação em um formato usual.*

```
> with(student): eq4 := completesquare(eq4, x);
```

Warning, the names distance, midpoint and slope have been redefined

$$eq4 := 576(x-4)^2 - 11200 + 320y^2 + 640y = 0$$

```
> eq4 := completesquare(eq4, y);
```

$$eq4 := 320(y+1)^2 - 11520 + 576(x-4)^2 = 0$$

```
> eq4 := lhs(eq4) + 11520 = rhs(eq4) + 11520;
```

$$eq4 := 320(y+1)^2 + 576(x-4)^2 = 11520$$

```
> eq4 := eq4/11520;
```

$$eq4 := \frac{1}{36}(y+1)^2 + \frac{1}{20}(x-4)^2 = 1$$

Exemplo 6.30 *Determine o centro, os focos, os vértices, as assíntotas e construa o gráfico da hipérbole* `H` *de equação* $9y^2 - 4x^2 - 36y + 20x = 100$.

```
> with(geometry):
> hyperbola(H, 9*y^2 - 4*x^2 - 36*y + 20*x = 100, [x,y]):
> coordinates(center(H));
```

$$[\frac{5}{2}, 2]$$

```
> map(coordinates, foci(H));
```

$$[[\frac{5}{2},\ 2 - \frac{1}{6}\sqrt{1443}],\ [\frac{5}{2},\ 2 + \frac{1}{6}\sqrt{1443}]]$$

```
> map(coordinates, vertices(H));
```

$$[[\frac{5}{2},\ 2 - \frac{1}{3}\sqrt{111}],\ [\frac{5}{2},\ 2 + \frac{1}{3}\sqrt{111}]]$$

```
> map(Equation, asymptotes(H));
```

$$[y + \frac{2}{3}x - \frac{11}{3} = 0,\ y - \frac{2}{3}x - \frac{1}{3} = 0]$$

Finalmente, construímos o gráfico de H (Figura 6.10).

```
> draw(H);
```

Exemplo 6.31 *Identifique as cônicas definidas pelas equações* $9x^2 + 16y^2 - 24xy - 20x + 110y - 50 = 0$, $2x^2 + 3y^2 + 8x - 6y - 13 = 0$ *e* $x^2 + 3y^2 + 4xy + x - 7y - 20 = 0$.

Figura 6.10:

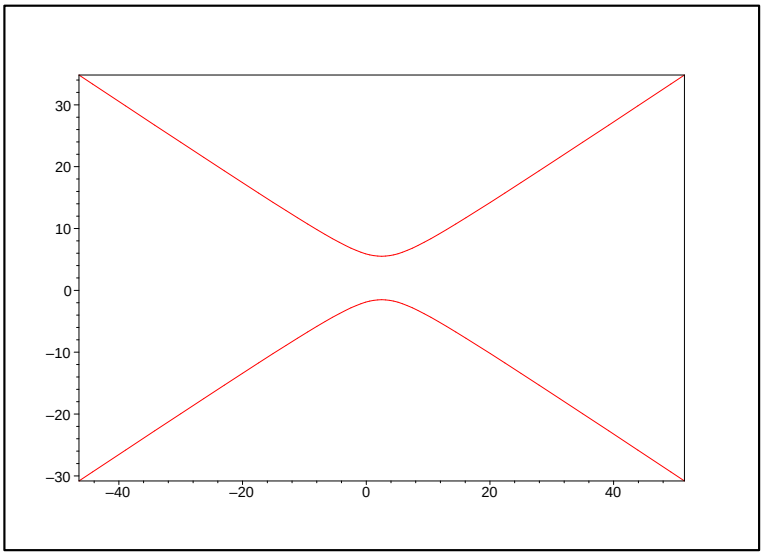
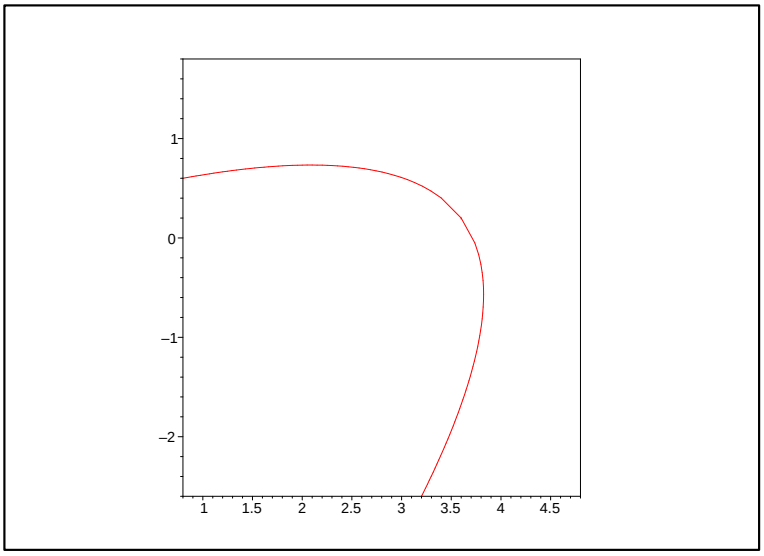


Figura 6.11:




```
> with(geometry):
> conic(c1, 9*x^2 + 16*y^2 - 24*x*y - 20*x + 110*y - 50 = 0, [x,y]);
```

$$c1$$

```
> form(c1);
```

$$parabola2d$$

Pode parecer estranho à primeira vista que c1 seja uma parábola, mas veja seu gráfico na Figura 6.11.

```
> draw(c1);
```

```
> detail(c1);
```

name of the object: c1

form of the object: parabola2d
vertex: [18/5, 1/5]
focus: [16/5, -1/10]
*directrix: 4/5*x+3/5*y-7/2 = 0*
*equation of the parabola: 9*x^2+16*y^2-24*x*y-20*x+110*y- 50 = 0*

```
> conic(c2, 2*x^2 + 3*y^2  + 8*x - 6*y - 13 = 0, [x, y]);
```

$$c2$$

```
> form(c2);
```

$$ellipse2d$$

```
> detail(c2);
```

name of the object: c2
form of the object: ellipse2d
center: $[-2, 1]$
foci: $[[-4, 1], [0, 1]]$
*length of the major axis: 4*3^(1/2)*
*length of the minor axis: 4*2^(1/2)*
*equation of the ellipse: 2*x^2+3*y^2+8*x-6*y-13 = 0*

```
> conic(c3, x^2 + 3*y^2 + 4*x*y + x - 7*y - 20 = 0, [x, y]);
```

geometry/conic/classify: "degenerate case: two intersecting lines"

$$[Line\_1\_c3,\ Line\_2\_c3]$$

A terceira equação corresponde a um par de retas concorrentes, considerado um caso degenerado de cônica.

Exemplo 6.32 *Uma cônica em posição geral pode ser definida se forem dados 5 de seus pontos, desde que não existam 3 colineares entre eles. Determine a equação e identifique a cônica que passa pelos pontos* $P_1 = (1, 1)$, $P_2 = (-3, 4)$, $P_3 = (0, -2)$, $P_4 = (2, -2)$, *e* $P_5 = (6, 7)$.

```
> point(P1, 1, 1), point(P2, -3, 4), point(P3, 0, -2),
>                  point(P4, 2, -2), point(P5, 6, 7);
```

$$P1, P2, P3, P4, P5$$

```
> conic(c4, [P1, P2, P3, P4, P5], [x, y]);
```

$$c4$$

```
> form(c4);
```

$$hyperbola2d$$

```
> eq := Equation(c4);
```

$$eq := 20196 - 20088x - 6858y + 2592xy + 12636x^2 - 8478y^2 = 0$$

```
> eq := eq/igcd(20196, 20088, 6858, 2592, 12636, 8478);
```

$$eq := 374 - 372x - 127y + 48xy + 234x^2 - 157y^2 = 0$$

Exemplo 6.33 *Obtenha a equação da cônica de excentricidade 1 que tem uma diretriz de equação* $x + 5 = 0$ *e um foco no ponto* $F = (-1, -2)$.

```
> line(R, x + 5 = 0, [x, y]): point(F, -1, -2):
> conic(c5, [R, F, 1], [x, y])
> form(c5);
```

$$parabola2d$$

```
> Equation(c5);
```

$$-20 + y^2 - 8x + 4y = 0$$

Com essa mesma diretriz e mesmo foco, aumentando a excentricidade para $4/3$*, vamos ver como fica a equação da cônica.*

```
> conic(c6, [R, F, 4/3], [x, y]):
> form(c6);
```

$$hyperbola2d$$

```
> Equation(c6);
```

$$-\frac{1}{3}x^2 - \frac{85}{3} + y^2 - \frac{34}{3}x + 4y = 0$$

6.8 Geometria Analítica tridimensional

O pacote `geom3d` define vários objetos e comandos relacionados com a geometria euclidiana espacial.

```
> with(geom3d);
```

Warning, the name polar has been redefined

[Archimedean, AreCollinear, AreConcurrent, AreConjugate, AreCoplanar, AreDistinct, AreParallel, ArePerpendicular, AreSameObjects, AreSamePlane, AreSkewLines, DefinedAs, DirectionRatios, Equation, FindAngle, FixedPoint, GlideReflect, GlideReflection, GreatDodecahedron, GreatIcosahedron, GreatRhombicuboctahedron, GreatRhombiicosidodecahedron, GreatStellatedDodecahedron, HarmonicConjugate, HexakisIcosahedron, HexakisOctahedron, InRadius, IsArchimedean, IsEquilateral, IsFacetted, IsOnObject, IsQuasi, IsRegular, IsRightTriangle, IsStellated, IsTangent, MidRadius, NormalVector, OnSegment, ParallelVector, PentagonalHexacontahedron, PentagonalIcositetrahedron, PentakisDodecahedron, QuasiRegularPolyhedron, RadicalCenter, RadicalLine, RadicalPlane, RegularPolyhedron, RhombicDodecahedron, RhombicTriacontahedron, RotatoryReflect, RotatoryReflection, ScrewDisplace, ScrewDisplacement, SmallRhombicuboctahedron, SmallRhombiicosidodecahedron, SmallStellatedDodecahedron, SnubCube, SnubDodecahedron, StereographicProjection, StretchRotate, TangentPlane, TetrakisHexahedron, TrapezoidalHexecontahedron, TrapezoidalIcositetrahedron, TriakisIcosahedron, TriakisOctahedron, TriakisTetrahedron, TruncatedCuboctahedron, TruncatedDodecahedron, TruncatedHexahedron, TruncatedIcosahedron, TruncatedIcosidodecahedron, TruncatedOctahedron,

TruncatedTetrahedron, altitude, area, center, centroid, circle, coordinates, cube, cuboctahedron, detail, dilate, distance, dodecahedron, draw, dsegment, duality, faces, facet, form, gtetrahedron, hexahedron, homology, homothety, icosahedron, icosidodecahedron, identity, incident, intersection, inverse, inversion, line, midpoint, octahedron, parallel, parallelepiped, parallelpiped, plane, point, polar, pole, powerps, projection, radius, randpoint, reflect, reflection, rotate, rotation, schlafli, segment, sides, sphere, stellate, tetrahedron, tname, transform, translate, translation, transprod, triangle, unit, valuesubs, vertices, volume, xcoord, xname, ycoord, yname, zcoord, zname]

6.8.1 Pontos, retas e planos

Um ponto no espaço tridimensional é definido de modo semelhante ao que é feito no caso plano. Por exemplo, `point(`*X, a, b, c*`)` define para o programa um ponto $X = (a, b, c)$.

Um plano `P`, do qual sejam conhecidos (1) três de seus pontos `A, B, C` ou (2) um de seus pontos `A` e o vetor normal `v` ou (3) duas retas `L1, L2` distintas (se as retas forem paralelas ou concorrentes, então a equação calculada é a do plano que contém `L1` e `L2`; se as retas forem reversas, então é calculada a equação do plano que passa por `L1` e é paralelo a `L2`) ou (4) um ponto `A` pelo qual ele passa e duas retas `L1, L2` distintas paralelas a ele ou (5) sua *equação* cartesiana, pode ser definido de um dos seguintes modos:

- `plane(P, [A, B, C], [`*variáveis*`])`
- `plane(P, [A, v], [`*variáveis*`])`
- `plane(P, [L1, L2], [`*variáveis*`])`
- `plane(P, [A, L1, L2], [`*variáveis*`])`
- `plane(P, `*equação*`, [`*variáveis*`])`

A lista de variáveis fornecida como terceiro parâmetro do comando `plane` é opcional.

Uma reta `L`, da qual sejam conhecidos (1) dois de seus pontos `A, B` ou (2) um de seus pontos `A` e o vetor diretor `v` ou (3) um de seus pontos `A` e um plano `P` ao qual ela é normal ou (4) dois planos `P1, P2` concorrentes que passam por ela ou (5) suas *equações* paramétricas no parâmetro `t`, pode ser definida com um dos seguintes comandos:

- `line(L, [A, B], t)`
- `line(L, [A, v], t)`
- `line(L, [A, P], t)`
- `line(L, [P1, P2], t)`
- `line(L, [`*equações*`], t)`

O parâmetro `t` fornecido como terceiro parâmetro é opcional.

Podemos obter informações a respeito de objetos já definidos ou definir novos objetos com os seguintes comandos:

Equation(X) Equação de **X** (plano ou reta)

Equation(P, [x, y, z]) Equação do plano **P** nas variáveis **x**, **y**, **z**

Equation(L, t) Equação da reta **L** no parâmetro **t**

detail(X) Informações detalhadas sobre **X** (plano ou reta)

NormalVector(P) Vetor normal ao plano **P**

ParallelVector(L) Vetor paralelo à reta **L**

AreCoplanar(A, B, C) Verifica se os pontos **A**, **B**, **C** são coplanares

AreCollinear(A, B, C) Verifica se os pontos **A**, **B**, **C** são colineares

AreCoplanar(L1, L2) Verifica se as retas **L1** e **L2** são coplanares

AreConcurrent(L1, L2) Verifica se as retas **L1** e **L2** são concorrentes

AreParallel(L1, L2) Verifica se as retas **L1** e **L2** são paralelas

ArePerpendicular(L1, L2) Verifica se as retas **L1** e **L2** são perpendiculares

AreSkewLines(L1, L2) Verifica se as retas **L1** e **L2** são reversas

AreParallel(P1, P2) Verifica se os planos **P1** e **P2** são paralelos

ArePerpendicular(P1, P2) Verifica se os planos **P1** e **P2** são perpendiculares

AreDistinct(X, Y) Verifica se os objetos **X** e **Y** (pontos, retas, planos ou esferas) são distintos

distance(X, Y) Distância entre os objetos **X** e **Y** (pontos ou retas ou planos)

FindAngle(X, Y) Ângulo entre os objetos **X** e **Y** (retas ou planos)

intersection(A, X, Y) Define o objeto geométrico **A** como sendo a interseção entre os objetos **X** e **Y** (retas ou planos)

randpoint(P, x1..x2, y1..y2, z1..z2) Ponto **P** gerado aleatoriamente com coordenadas **x**, **y**, **z** nos intervalos [**x1**, **x2**], [**y1**, **y2**], [**z1**, **z2**], respectivamente

randpoint(P, X) Ponto **P** escolhido aleatoriamente no objeto **X** (plano, reta ou esfera)

IsOnObject(P, X) Testa se o ponto **P** pertence ao objeto **X** (plano, reta ou esfera)

Exemplo 6.34 *Vamos determinar um vetor normal ao plano $P1$ definido pela equação $x + 2y + 3z = 1$.*

```
> with(geom3d):
> plane(P1, x+2*y+3*z=1, [x,y,z]);
```

$$P1$$

```
> NormalVector(P1);
```

$$[1, 2, 3]$$

Agora, vamos determinar a equação e um vetor normal ao plano $P2$ que passa pelos pontos $A = (0, 1, 1)$, $B = (-1, 1, 2)$ e $C = (4, 3, 0)$.

```
> point(A, 0, 1, 1), point(B, -1, 1, 2), point(C, 4, 3, 0):
> plane(P2, [A, B, C], [x, y, z]);
```

$$P2$$

```
> Equation(P2);
```

$$-1 - 2x + 3y - 2z = 0$$

```
> NormalVector(P2);
```

$$[-2, 3, -2]$$

Calculemos o ângulo entre os planos $P1$ e $P2$:

```
> FindAngle(P1, P2);
```

$$\arccos(\frac{1}{119}\sqrt{14}\sqrt{17})$$

Peguemos aleatoriamente um ponto X no plano $P1$ e um ponto Y em $P2$ e calculemos a distância entre esses pontos.

```
> randpoint(X, P1);
```

$$X$$

```
> randpoint(Y, P2);
```

$$Y$$

```
> evalf(distance(X, Y));
```

$$.7558255869$$

Finalmente, vamos testar se os pontos $Z = (0, 7, -11)$ e $W = (0, 7, -23/3)$ pertencem ao plano $P1$:

```
> IsOnObject(point(Z, 0, 7, -11), P1);
```

$$false$$

```
> IsOnObject(point(W, 0, 7, -13/3), P1);
```

$$true$$

Exemplo 6.35 *Neste exemplo, definimos os planos $\alpha : 2x + ry + sz + 5 = 0$ e $\beta : 2x + ry + sz - 7 = 0$, calculamos a distância de um ponto $p = (a, b, c)$ ao plano α, verificamos se os planos são paralelos e calculamos a distância entre eles.*

```
> with(geom3d):
> plane(alpha, 2*x + r*y + s*z + 5 = 0, [x, y, z]);
```

$$\alpha$$

```
> distance(point(p, a, b, c), alpha);
```

$$\frac{|2a + rb + sc + 5|}{\sqrt{4 + r^2 + s^2}}$$

```
> plane(beta, 2*x + r*y + s*z - 7 = 0, [x, y, z]);
```

$$\beta$$

```
> AreParallel(alpha, beta); # são paralelos?
```

$$true$$

```
> distance(alpha, beta);
```

$$12\frac{1}{\sqrt{4+r^2+s^2}}$$

Exemplo 6.36 *Determinar a equação e um vetor paralelo à reta $R1$ que passa pelos pontos $P=(0,3,-8)$ e $Q=(-4,0,1)$.*

```
> with(geom3d):
> point(P, 0, 3, -8): point(Q, -4, 0, 1):
> line(R1, [P, Q], t);
```

$$R1$$

```
> Equation(R1);
```

$$[-4t, 3-3t, -8+9t]$$

```
> ParallelVector(R1);
```

$$[-4, -3, 9]$$

Exemplo 6.37 *Obter a equação da reta $R2$ que passa por um ponto $A=(1,2,3)$ dado e é perpendicular a um plano π cuja equação é $2x-5y+3z=9$.*

```
> with(geom3d):
> point(A, 1, 2, 3), plane(pi, 2*x - 5*y + 3*z = 9, [x,y,z]):
> line(R2, [A, pi]):
> Equation(R2, t);
```

$$[1+2t,\ 2-5t,\ 3+3t]$$

Exemplo 6.38 *Obter as equações paramétricas da reta L definida pela interseção de dois planos $P1:\ x+y=4$ e $P2:\ -4y+3z=1$.*

```
> with(geom3d):
> plane(P1, x + y = 4, [x, y, z]), plane(P2, -4*y + 3*z = 1, [x, y,z]):
> line(L, [P1, P2]):
> Equation(L, s);  # equação de L com parâmetro s
```

$$\left[\frac{17}{4}+3s,\ -\frac{1}{4}-3s,\ -4s\right]$$

Exemplo 6.39 *Seja $R3$ a reta que passa pelo ponto $P2=(-1,4,1/2)$ com vetor diretor $v2=(4,-3,6)$. Determine as equações paramétricas de $R3$ e sua interseção com o plano $2x-y+4z=5$.*

```
> with(geom3d):
> point(P2, -1, 4, 1/2);
```

$$P2$$

```
> v2 := vector([4, -3, 6]);
```

$$v2 := [4, -3, 6]$$

```
> line(R3, [P2, v2], s);
```

$$R3$$

```
> EqReta := Equation(R3);
```

$$EqReta := [-1+4s,\ 4-3s,\ \frac{1}{2}+6s]$$

```
> EqPlano := 2*x - y + 4*z = 5;
```

$$EqPlano := 2x - y + 4z = 5$$

Para determinar a interseção, basta resolver o sistema formado pelas equações da reta e do plano dado:

```
> solve({x = EqReta[1], y = EqReta[2], z = EqReta[3], EqPlano});
```

$$\{s = \frac{9}{35},\ y = \frac{113}{35},\ z = \frac{143}{70},\ x = \frac{1}{35}\}$$

Logo, a interseção de R2 com P2 é o ponto $\left(\frac{1}{35}, \frac{113}{35}, \frac{143}{70}\right)$.

Uma outra solução é usar o comando `intersection`*. Para isso, precisamos definir o plano dado como um objeto geométrico do pacote* `geom3d`*:*

```
> plane(P, 2*x-y+4*z=5, [x,y,z]):
> intersection(Q, R3, P);
> coordinates(Q);
```

$$\left[\frac{1}{35}, \frac{113}{35}, \frac{143}{70}\right]$$

Os parâmetros do comando `intersection` *devem ser objetos geométricos e não equações. Por isso, obtém-se uma mensagem de erro do Maple ao ser digitado algo como*

```
> intersection(Q, R3, EqPlano);   # erro
```

Exemplo 6.40 *Sejam L1 e L2 as retas parametrizadas por* $[2t-4, -t+4, -2t-1]$ *e* $[4t-5, -3t+5, -5t+5]$*, respectivamente. Calcule a distância entre essas retas e verifique se elas são paralelas ou reversas.*

```
> with(geom3d):
> line(L1, [ 2*t - 4, -t+4, -2*t-1], t);
```

$$L1$$

```
> line(L2, [4*t - 5, -3*t + 5, -5*t + 5], t);
```

$$L2$$

```
> distance(L1, L2);
```

$$3$$

```
> AreParallel(L1, L2); # são retas paralelas?
```

$$false$$

```
> AreSkewLines(L1,L2); # são retas reversas?
```

$$true$$

Concluímos então que as retas dadas são reversas e que a distância entre elas é 3*.*

Exemplo 6.41 *Sejam* $P = (4, 1, 6)$ *um ponto no espaço tridimensional e R4 a reta que passa pelos pontos* $A = (8, 3, 2)$ *e* $B = (2, -3, 5)$*. Verifique se P pertence a R4 e calcule a distância entre eles.*

```
> with(geom3d):
> point(P, 4, 1, 6), point(A, 8, 3, 2), point(B, 2, -3, 5):
> line(R4, [A, B]):
> IsOnObject(P, R4);
```

$$false$$

```
> distance(P, R4);
```

$$\frac{2}{3}\sqrt{17}$$

6.8.2 Esferas

Uma esfera `E`, da qual sejam conhecidos (1) quatro de seus pontos `A, B, C, D` ou (2) dois de seus pontos `A, B` diametralmente opostos ou (3) o centro `C` e o raio `R` ou (4) o centro `C` e um plano tangente `P` ou (5) sua *equação*, pode ser definida com um comando `sphere` de uma das seguintes maneiras:

- `sphere(E, [A, B, C, D], [`*variáveis*`])`
- `sphere(E, [A, B], [`*variáveis*`])`
- `sphere(E, [C, R], [`*variáveis*`])`
- `sphere(E, [C, P], [`*variáveis*`])`
- `sphere(E,` *equação*`, [`*variáveis*`])`

O comando `sphere` admite ainda um quarto parâmetro opcional com o nome do centro que deve ser escrito na forma `centername =` *centro*.

Podemos obter informações a respeito de uma esfera E já definida com os seguintes comandos:

Equation(E) Equação de **E**

center(E) Centro de **E**

radius(E) Raio de **E**

area(E) Área de **E**

volume(E) Volume de **E**

detail(E) Informações detalhadas sobre **E**

Exemplo 6.42 *Definimos uma esfera de centro $O = (3, 2, 1)$ e raio 3 e verificamos os detalhes do objeto geométrico assim construído.*

```
> with(geom3d):
> sphere(E, [point(O, 3, 2, 1), 3], [x, y, z]):
> detail(E);
```

name of the object: E
form of the object: sphere3d
name of the center: O
coordinates of the center: [3, 2, 1]
radius of the sphere: 3

*surface area of the sphere: 36*Pi*
*volume of the sphere: 36*Pi*
*equation of the sphere: x^2+y^2+z^2+5-6*x-4*y-2*z = 0*

Exemplo 6.43 *Calcule a equação, o centro e o raio da esfera que passa pelos pontos* $A = (1, 1, 1)$, $B = (2, 2, 2)$, $C = (-3, 2, 1)$ *e* $D = (0, 0, -3)$.

```
> point(A, 1, 1, 1), point(B, 2, 2, 2),
>            point(C, -3, 2, 1), point(D, 0, 0, -3)]:
> sphere(E2, [A, B, C, D], [x, y, z]):
> Equation(E2);
```

$$x^2 + y^2 + z^2 + 6 - \frac{3}{5}x - \frac{67}{5}y + 5z = 0$$

```
> coordinates(center(E2));
```

$$[\frac{3}{10}, \frac{67}{10}, -\frac{5}{2}]$$

```
> radius(E2);
```

$$\frac{1}{10}\sqrt{4523}$$

Agora, vamos gerar aleatoriamente um ponto P *com coordenadas no intervalo* $[1, 5]$ *e verificar se* P *pertence à esfera* $E2$.

```
> randpoint(P, 1..5, 1..5, 1..5):
> evalf(coordinates(P));
```

$$[3.967867632, 1.444277308, 2.453229479]$$

```
> IsOnObject(P, E2);
```

$$false$$

Finalmente, vamos escolher aleatoriamente um ponto Q *da esfera.*

```
> randpoint(Q, E2):
> evalf(coordinates(Q));
```

$$[-2.494929023, 8.110524397, -8.452209059]$$

```
> IsOnObject(Q, E2);
```

$$true$$

Exemplo 6.44 *Determinar a equação da esfera* $E3$ *com centro* $C = (3, 4, -11)$ *e tangente ao plano* $\pi : 3x - y + 4z = 8$.

```
> with(geom3d):
> point(C, 3, 4, -11):
> plane(pi, 3*x - y + 4*z = 8, [x, y, z]):
> sphere(E3, [C, pi], [x, y, z], centername = C):
> Equation(E3);
```

$$x^2 + y^2 + z^2 + \frac{1587}{26} - 6x - 8y + 22z = 0$$

```
> coordinates(C);
```

$$[3,\ 4,\ -11]$$

```
> radius(E3);
```

$$\frac{47}{26}\sqrt{26}$$

```
> distance(C, pi);
```

$$\frac{47}{26}\sqrt{26}$$

6.9 Classificação de cônicas e quádricas

A equação geral do segundo grau nas variáveis x e y

$$Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0$$

(onde A ou B ou C é diferente de 0) pode ser escrita na forma

$$\begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} A & B/2 \\ B/2 & C \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} D & E \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + F = 0$$

Em geral, essa equação representa uma cônica ou um caso degenerado (conjunto vazio, par de retas, etc.)

Através de uma rotação dos eixos coordenados, é possível a eliminação do termo misto Bxy e, dessa forma, podemos obter uma equação mais simples que permita sua identificação. Para isso, determinamos os autovetores v_1 e v_2 da matriz $M = \begin{bmatrix} A & B/2 \\ B/2 & C \end{bmatrix}$ e, a partir deles, construímos uma base ortonormal $\{w_1, w_2\}$. Sendo P a matriz de determinante igual a 1 cujas colunas são formadas pelos vetores w_1 e w_2, fazemos a mudança de variáveis

$$\begin{bmatrix} x \\ y \end{bmatrix} = P \begin{bmatrix} r \\ s \end{bmatrix}$$

na equação nas variáveis x, y e obtemos uma nova equação nas variáveis r, s sem o termo misto.

Exemplo 6.45 *Classificar a cônica cuja equação é*

$$5x^2 - 4xy + 8y^2 + 4\sqrt{5}x - 16\sqrt{5}y + 4 = 0.$$

Definimos a matriz M associada da forma quadrática e calculamos seus autovetores:

```
> restart: with(linalg):
> eq := 5*x^2 - 4*x*y + 8*y^2 + 4*sqrt(5)*x - 16*sqrt(5)*y + 4 = 0:
> M := matrix([[5, -2], [-2, 8]]);
```

$$M := \begin{bmatrix} 5 & -2 \\ -2 & 8 \end{bmatrix}$$

```
> autovet := eigenvectors(M);
```

$$autovet \ := \ [9, 1, \{[1, -2]\}],\ [4, 1, \{[2, 1]\}]$$

```
> v1 := autovet[1][3][1];
```

$$v1 := [1, -2]$$

```
> v2 := autovet[2][3][1];
```

$$v2 := [2, 1]$$

Dividimos cada autovetor encontrado pela sua norma e construímos uma matriz Q cujas linhas são os autovetores normalizados e que tenha determinante 1*:*

```
> w1 := evalm(v1/norm(v1, 2)); w2 := evalm(v2/norm(v2, 2));
```

$$w1 := \left[\frac{\sqrt{5}}{5}, -\frac{2\sqrt{5}}{5}\right]$$

$$w2 := \left[\frac{2\sqrt{5}}{5}, \frac{\sqrt{5}}{5}\right]$$

```
> Q := matrix([w2, w1]);
```

$$Q := \begin{bmatrix} \frac{2\sqrt{5}}{5} & \frac{\sqrt{5}}{5} \\ \frac{\sqrt{5}}{5} & -\frac{2\sqrt{5}}{5} \end{bmatrix}$$

```
> det(Q);
```

$$1$$

```
> P := transpose(Q);
```

$$P := \begin{bmatrix} \frac{2\sqrt{5}}{5} & \frac{\sqrt{5}}{5} \\ \frac{\sqrt{5}}{5} & -\frac{2\sqrt{5}}{5} \end{bmatrix}$$

Definimos a matriz com as novas variáveis:

```
> var := matrix([[r], [s]]);
```

$$var := \begin{bmatrix} r \\ s \end{bmatrix}$$

```
> R := evalm(P &* var);
```

$$R := \begin{bmatrix} \frac{2\sqrt{5}r}{5} + \frac{\sqrt{5}s}{5} \\ \frac{\sqrt{5}r}{5} - \frac{2\sqrt{5}s}{5} \end{bmatrix}$$

Fazemos agora a mudança de variáveis na equação dada:

```
> subs(x = R[1, 1], y = R[2, 1], eq);
```

$$5\left(\frac{2\sqrt{5}r}{5} + \frac{\sqrt{5}s}{5}\right)^2 - 4\left(\frac{2\sqrt{5}r}{5} + \frac{\sqrt{5}s}{5}\right)\left(\frac{\sqrt{5}r}{5} - \frac{2\sqrt{5}s}{5}\right) + 8\left(\frac{\sqrt{5}r}{5} - \frac{2\sqrt{5}s}{5}\right)^2 + 4\sqrt{5}\left(\frac{2\sqrt{5}r}{5} + \frac{\sqrt{5}s}{5}\right) - 16\sqrt{5}\left(\frac{\sqrt{5}r}{5} - \frac{2\sqrt{5}s}{5}\right) + 4 = 0$$

```
> eq2 := simplify(%);
```

$$eq2 := 4r^2 + 9s^2 - 8r + 36s + 4 = 0$$

Agora, vamos escrever a equação em outro formato, fazendo o completamento de alguns quadrados.

```
> with(student):
> eq2 := completesquare(eq2, r);
```

$$eq2 := 4(r-1)^2 + 9s^2 + 36s = 0$$

```
> eq2 := completesquare(eq2, s);
```

$$eq2 := 9(s+2)^2 - 36 + 4(r-1)^2 = 0$$

```
> eq2 := lhs(eq2) + 36 = rhs(eq2) + 36;
```

$$eq2 := 9(s+2)^2 + 4(r-1)^2 = 36$$

```
> eq2 := eq2/36;
```

$$eq2 := \frac{(s+2)^2}{4} + \frac{(r-1)^2}{9} = 1$$

Vemos assim que a equação dada corresponde a uma elipse.

A equação geral do segundo grau nas variáveis x, y e z

$$Ax^2 + By^2 + Cz^2 + Dxy + Exz + Fyz + Gx + Hy + Iz + J = 0$$

(onde A ou B ou C ou D ou E ou F é diferente de 0) pode ser escrita na forma

$$\begin{bmatrix} x & y & z \end{bmatrix} \begin{bmatrix} A & D/2 & E/2 \\ D/2 & B & F/2 \\ E/2 & F/2 & C \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} G & H & I \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} + J = 0$$

Em geral, ela representa uma quádrica ou um caso degenerado.

Para classificá-la, podemos fazer uma rotação de eixos através da mudança de variáveis $\begin{bmatrix} x \\ y \\ z \end{bmatrix} = P \begin{bmatrix} r \\ s \\ t \end{bmatrix}$, onde P é a matriz 3×3 de determinante 1 cujas colunas correspondem a uma base ortonormal de autovetores da matriz $M = \begin{bmatrix} A & D/2 & E/2 \\ D/2 & B & F/2 \\ E/2 & F/2 & C \end{bmatrix}$.

Em uma matriz simétrica, autovetores associados a autovalores distintos são ortogonais. Neste caso, basta dividir cada autovetor pela sua norma para obtermos autovetores ortonormais.

Autovetores associados a um mesmo autovalor podem ser ortogonalizados com a aplicação da ortogonalização de Gram-Schmidt restrito a eles.

Exemplo 6.46 *Classificar a quádrica cuja equação é*

$$2x^2 + 3y^2 + 23z^2 + 72xz + 150 = 0.$$

```
> with(linalg):
> eq := 2*x^2 + 3*y^2 + 23*z^2 + 72*x*z + 150 = 0:
```

Construímos a matriz da forma quadrática e calculamos seus autovetores:

```
> M := matrix([[2, 0, 36], [0, 3, 0], [36, 0, 23]]);
```

$$M := \begin{bmatrix} 2 & 0 & 36 \\ 0 & 3 & 0 \\ 36 & 0 & 23 \end{bmatrix}$$

```
> autovet := eigenvectors(mat);
```

$$autovet := [3, 1, \{[0, 1, 0]\}],\ [50, 1, \{[1, 0, 4/3]\}],\ [-25, 1, \{[-4/3, 0, 1]\}]$$

```
> v1 := autovet[1][3][1];
```

$$v1 := [0, 1, 0]$$

```
> v2 := autovet[2][3][1];
```

$$v2 := [1, 0, \frac{4}{3}]$$

```
> v3 := autovet[3][3][1];
```

$$v3 := [-\frac{4}{3}, 0, 1]$$

Como esses autovetores são associados a autovalores distintos, então eles são ortogonais. Logo, dividindo cada um deles pela sua norma obtemos um conjunto ortonormal.

```
> w1 := evalm(v1/norm(v1, 2));
```

$$w1 := [0,\ 1,\ 0]$$

```
> w2 := evalm(v2/norm(v2, 2));
```

$$w2 := \left[1,\ 0,\ \frac{4}{3}\right]$$

```
> w3 := evalm(v3/norm(v3, 2));
```

$$w3 := \left[-\frac{4}{3},\ 0,\ 1\right]$$

Formamos a matriz cujas linhas são $w1$, $w2$ e $w3$ e calculamos seu determinante.

```
> Q := matrix([w1, w2, w3]);
```

$$Q := \begin{bmatrix} 0 & 1 & 0 \\ \frac{3}{5} & 0 & \frac{4}{5} \\ -\frac{4}{5} & 0 & \frac{3}{5} \end{bmatrix}$$

```
> det(Q)
```

$$-1$$

Como o determinante de Q é igual a -1, devemos trocar um par de linhas para obtermos uma matriz de determinante 1:

```
> Q := matrix([w2, w1, w3]);
```

$$Q := \begin{bmatrix} \frac{3}{5} & 0 & \frac{4}{5} \\ 0 & 1 & 0 \\ -\frac{4}{5} & 0 & \frac{3}{5} \end{bmatrix}$$

```
> P := transpose(Q);
```

$$P := \begin{bmatrix} \frac{3}{5} & 0 & -\frac{4}{5} \\ 0 & 1 & 0 \\ \frac{4}{5} & 0 & \frac{3}{5} \end{bmatrix}$$

Definimos agora a matriz com as novas variáveis e fazemos a substituição na equação original.

```
> var := matrix([[r], [s], [t]]);
```

$$var := \begin{bmatrix} r \\ s \\ t \end{bmatrix}$$

```
> R := evalm(P &* var);
```

$$R := \begin{bmatrix} \frac{3r}{5} - \frac{4t}{5} \\ s \\ \frac{4r}{5} + \frac{3t}{5} \end{bmatrix}$$

```
> subs(x = R[1,1], y = R[2,1], z = R[3,1], eq);
```

$$2\left(\frac{3r}{5} - \frac{4t}{5}\right)^2 + 3s^2 + 23\left(\frac{4r}{5} + \frac{3t}{5}\right)^2 + 72\left(\frac{3r}{5} - \frac{4t}{5}\right)\left(\frac{4r}{5} + \frac{3t}{5}\right) + 150 = 0$$

```
> simplify(%);
```

$$50r^2 - 25t^2 + 3s^2 + 150 = 0$$

Vemos assim que se trata de um hiperbolóide de duas folhas. *Se quiséssemos fazer o gráfico, bastaria usar um comando* `implicitplot3d` *do pacote* `plots`.

Exemplo 6.47 *Identificar a quádrica* $4x^2+4y^2+4z^2+4xz+4xy+4yz-5=0$. *Queremos destacar que temos dois autovetores* $v1$ *e* $v2$ *associados ao autovalor* 2 *e por isso foi usada a ortogonalização de Gram-Schmidt em* $v1$ *e* $v2$.

```
> with(linalg):   eq := 4*x^2+4*y^2+4*z^2+4*x*z+4*x*y+4*y*z-5 = 0:
> M := matrix([[4, 2, 2], [2, 4, 2], [2, 2, 4]]):
> autovet := eigenvectors(M);
```

$$autovet := [2, 2, \{[-1, 0, 1], [-1, 1, 0]\}], [8, 1, \{[1, 1, 1]\}]$$

```
> v1 := autovet[1][3][1]; v2 := autovet[1][3][2]; v3 := autovet[2][3][1];
```

$$v1 := [-1, 0, 1]$$
$$v2 := [-1, 1, 0]$$
$$v3 := [1, 1, 1]$$

```
> g := GramSchmidt([v1, v2], normalized);
```

$$g := \left[\left[-\frac{\sqrt{2}}{2}, 0, \frac{\sqrt{2}}{2}\right], \left[-\frac{\sqrt{6}}{6}, \frac{\sqrt{6}}{3}, -\frac{\sqrt{6}}{6}\right]\right]$$

```
> w3 := evalm(v3/norm(v3, 2)):
> Q := matrix([g[2], g[1], w3]):
> var := matrix([[r], [s], [t]]):
> a := evalm(transpose(Q) &* var):
> subs(x=a[1,1], y=a[2,1], z=a[3,1], eq):
> simplify(%);
```

$$-5 + 2s^2 + 8t^2 + 2r^2 = 0$$

Portanto, trata-se de um elipsóide.

6.10 Exercícios

1) Calcule a integral $\int e^{\operatorname{arcsen}(x)}dx$. (*Sugestão: use duas vezes integração por partes com* $u = e^{\operatorname{arcsen}(x)}$)

2) Calcule $\int \ln^2(x+\sqrt{1+x^2})dx$. (*Sugestão: use duas vezes integração por partes com* $u = \ln(x+\sqrt{1+x^2})$)

3) Calcule $A = \int_0^{\pi} \frac{\sqrt{\operatorname{sen} x}}{\sqrt{\operatorname{sen} x}+\sqrt{\cos x}}dx$. Provavelmente o Maple não conseguirá calcular A diretamente, mas você poderá determinar como esse cálculo será efetuado:

- Inicialmente, defina B como sendo o mesmo que A depois da mudança de variável $\pi - x = u$;
- Redefina B usando a mudança de variável $u = x$;
- Calcule $A + B$; para isso, use um comando `combine` seguido de um `value`;
- Como $A + B = 2A$, obtenha o valor de A.

4) a) Simplifique a expressão $A = \frac{-1}{x^5+x+1} + \frac{x(5x^4+1)}{(x^5+x+1)^2}$;

b) Usando integração por partes com $u = x$, calcule $\int \frac{x(5x^4+1)}{(x^5+x+1)^2}dx$;

c) Calcule $\int \frac{4x^5-1}{(x^5+x+1)^2}dx$.

5) Determine a equação da reta que passa pelo centro das esferas $x^2+y^2+z^2-y+2z = 10$ e $2x^2+2y^2+2z^2+3x-10z = 17$.

6) Identifique a posição relativa das retas cujas equações paramétricas são $x = r$, $y = 3r+1$, $z = -2r-3$ e $x = 2s+3$, $y = s-1$, $z = 0$ com $r, s \in \mathbb{R}$ e calcule a distância entre elas.

7) Obtenha as equação de um plano que passa pelo ponto $P = (3, -1, 7)$ e cujo vetor normal é paralelo aos planos $x - y + 3z = 1$ e $x + 4y + 4z = 11$.

8) Identifique as seguintes cônicas:
a) $5x^2 + 4xy + 5y^2 = 9$;
b) $11x^2 + 24xy + 4y^2 - 15$;
c) $5xy - 3 = 0$.

9) Identifique quádricas ou casos degenerados nas seguintes equações:
a) $xy + yz + xz = 1$;
b) $2xy + z = 0$;
c) $2x^2 + 2y^2 + 5z^2 - 4xy - 2xz + 2yz + 10x - 26y - 2z = 0$.

Capítulo 7

Equações Diferenciais

O Maple possui vários algoritmos implementados em forma de comandos que servem para resolver equações e sistemas de equações diferenciais, sejam elas ordinárias ou parciais.

Neste capítulo, damos apenas uma pequena amostra da resolução desse tipo de problema. Apresentamos brevemente alguns comandos dos pacotes `DEtools` (que tem mais de 100 comandos) e `PDEtools`.

7.1 Equações diferenciais ordinárias

O comando básico para resolução de Equações Diferenciais Ordinárias (EDO) é o `dsolve`. Ele pode ser usado de várias maneiras:

dsolve(EDO) Resolve a EDO fornecida como parâmetro.

dsolve(EDO, y(x), opções) Calcula $y(x)$ na EDO fornecida como primeiro parâmetro. Opcionalmente, pode definir o método de resolução que pode ser *implicit, explicit, parametric, useInt, quadrature, linear, homogeneous, separable, Bernoulli, Abel,* entre outros.

dsolve({EDO, CI}, y(x), opções) Calcula $y(x)$ na EDO, usando as condições iniciais CI e opções fornecidas.

dsolve({sistema, condições}, {funções}, opções) Determina as funções que são soluções do sistema de EDO usando condições iniciais e opções fornecidas.

Uma EDO pode ser definida com o operador diferencial `D` ou com o comando `diff`. Por exemplo, a equação

$$y''' - 3y'' + y' - 5y = \cos(x)$$

pode ser fornecida na forma

```
diff(y(x), x$3) - 3*diff(y(x), x$2) + diff(y(x), x) - 5*y(x) = cos(x)
```

ou na forma

```
(D@@3)(y)(x) - 3*(D@@2)(y)(x) + D(y)(x) - 5*y(x) = cos(x)
```

As constantes reais arbitrárias que aparecem após a resolução das equações são denotadas pelo Maple por `_C1, _C2, _C3, ...`

Exemplo 7.1 *Vamos resolver a equação diferencial de primeira ordem linear* $y' + y = y^2$ *escrevendo-a inicialemente com o operador diferencial* `D`*. A função y deve ser escrita acompanhada de uma variável, como em* $y(x)$*.*

```
> eq1 := D(y)(x) + y(x) = y(x)^2;
```

$$eq1 := D(y)(x) + y(x) = y(x)^2$$

```
> dsolve(eq1);     # resolve a EDO definida em eq1
```

$$y(x) = \frac{1}{1 + e^x \_C1}$$

Escrevendo a mesma equação com o comando `diff`*:*

```
> eq2 := diff(y(x), x) + y(x) = y(x)^2;
```

$$eq2 := \left(\frac{\partial}{\partial x} y(x)\right) + y(x) = y(x)^2$$

```
> dsolve(eq2);   # resolve a EDO definida em eq2
```

$$y(x) = \frac{1}{1 + e^x \_C1}$$

Obtendo solução escrita na forma de integral:

```
> dsolve(eq1, useInt);
```

$$y(x) = \frac{1}{e^x \left(\int -\frac{1}{e^x} dx + \_C1\right)}$$

Obtendo agora solução na forma implícita:

```
> dsolve(eq1, implicit);
```

$$\frac{1}{y(x)} - 1 - e^x \_C1 = 0$$

Os comandos `odeadvisor(`*equação*`)` e `intfactor(`*equação*`)` do pacote `DEtools` classifica e calcula um fator integrante para a equação diferencial fornecida como parâmetro, respectivamente.

Exemplo 7.2 *Consideremos a equação diferencial linear homogênea de quarta ordem com coeficientes constantes* $y'''' + 8y''' + 24y'' + 32y' + 16y = 0$*. Vamos resolvê-la com o* `dsolve` *e classificá-la com o* `odeadvisor`*.*

```
> eq3 := diff(y(x), x$4) + 8*diff(y(x), x$3) + 24*diff(y(x), x$2) +
>                                   32*diff(y(x), x) + 16*y(x) = 0;
```

$$eq3 := \left(\frac{\partial^4}{\partial x^4} y(x)\right) + 8\left(\frac{\partial^3}{\partial x^3} y(x)\right) + 24\left(\frac{\partial^2}{\partial x^2} y(x)\right) + 32\left(\frac{\partial}{\partial x} y(x)\right) + 16y(x) = 0$$

```
> dsolve(eq3);
```

$$y(x) = \_C1 e^{(-2x)} + \_C2 e^{(-2x)} x + \_C3 e^{(-2x)} x^2 + \_C4 e^{(-2x)} x^3$$

```
> with(DEtools): odeadvisor(eq3);
```

$$[[\_high\_order, \_missing\_x]]$$

A classificação mostrada indica que a equação dada é de ordem superior ("higher order") com coeficientes constantes ("missing x").

Exemplo 7.3 *Vamos obter um fator integrante, resolver e classificar a equação* $y' = \dfrac{3yx^2}{x^3 - 2y^4}$.

```
> with(DEtools):
> eq4 := D(y)(x) = 3*y(x)*x^2/(x^3 + 2*y(x)^4);
```

$$eq4 := D(y)(x) = 3\frac{y(x)x^2}{x^3 + 2y(x)^4}$$

```
> intfactor(eq4); # obtém fator integrante
```

$$\frac{x^3 + 2y(x)^4}{y(x)^2}$$

```
> dsolve(eq4);
```

$$\ln(x) - \_C1 - \frac{4}{9}\ln\left(\frac{y(x)}{x^{3/4}}\right) + \frac{4}{9}\ln\left(-\frac{3x^3 - 2y(x)^4}{x^3}\right) = 0$$

Note que a solução fornecida está na forma implícita.

```
> odeadvisor(eq4); # classifica a equação
```

$$[[\_homogeneous, classG], \_rational]$$

Para informações sobre as classificação das equações, consulte as telas de ajuda do programa digitando

```
> ?odeadvisor,types
```

O comando `declare` do pacote `PDEtools` simplifica as notações de derivada que aparecem nas equações diferenciais, deixando a apresentação da equação mais próxima do formato usual.

Exemplo 7.4 *Neste exemplo, escrevemos algumas EDOs dadas no formato usual.*

```
> with(PDEtools):
> declare(y(x), prime=x);
```

y(x) will now be displayed as y
derivatives with respect to: x of functions of one variable will now be displayed with '

```
> diff(y(x), x$4) - 10*diff(y(x), x$2) + 9 = 0;
```

$$y'''' - 10y'' + 9 = 0$$

```
> diff(y(x), x$4) + 8*diff(y(x), x$3) + 24*diff(y(x), x$2) +
>                          32*diff(y(x), x) + 16*y(x) = 0;
```

$$y'''' + 8y''' + 24y'' + 32y' + 16y = 0$$

```
> diff(y(x), x) = 2*x*y(x) - x;
```

$$y' = 2xy - x$$

Para testar se uma solução encontrada está correta, podemos substituí-la diretamente na equação e observar a igualdade obtida. Outra maneira, mais fácil, é usar um comando `odetest(`*solução, equação*`)`. Se esse comando retornar uma resposta igual a zero, então a solução encontrada é realmente uma solução da equação dada.

Exemplo 7.5 *Vamos resolver a equação linear de primeira ordem* $y' = 2xy - x$.

```
> eq5  := diff(y(x),x) = 2*x*y(x) - x;
```

$$eq5 := \frac{\partial}{\partial x} y(x) = 2xy(x) - x$$

```
> sol := dsolve(eq5);
```

$$sol := y(x) = \frac{1}{2} + e^{(x^2)}\_C1$$

Agora vamos substituir a resposta obtida na equação dada e simplificar:

```
> subs(sol, eq5);
```

$$\frac{\partial}{\partial x}\left(\frac{1}{2} + e^{(x^2)}\_C1\right) = 2x(\frac{1}{2} + e^{(x^2)}\_C1) - x$$

```
> simplify(%);
```

$$2xe^{(x^2)}\_C1 = 2xe^{(x^2)}\_C1$$

```
> odetest(sol, eq1);  # retorna 0 se sol for realmente uma solução
```

$$0$$

7.2 Problemas de valor inicial

Valores iniciais das soluções de EDO podem ser fornecidas juntamente com a própria equação. A equação e as condições iniciais devem formar um conjunto a ser fornecido como primeiro parâmetro do `dsolve`.

Exemplo 7.6 *Determine a solução de* $y'' + 4y' + 13y = 2t + 3e^{-2t}\cos(3t)$, $y(0) = 0$, $y'(0) = -1$.

```
> pvi := {diff(y(t), t$2) + 4*diff(y(t), t) + 13*y(t) = 2*t +
>                       3*exp(-2*t)* cos(3*t), y(0) = 0, D(y)(0) = -1};
```

$$pvi := \left(\frac{\partial^2}{\partial t^2} y(t)\right) + 4\left(\frac{\partial}{\partial t} y(t)\right) + 13y(t) = 2t + 3e^{(-2t)}\cos(3t),$$
$$y(0) = 0,\ D(y)(0) = -1$$

```
> solucao := dsolve({pvi});
```

$$solucao := y(t) = -\frac{179}{507}e^{(-2t)}\sin(3t) - \frac{121}{1014}e^{(-2t)}\cos(3t) +$$
$$\frac{1}{1014}(-48e^{(2t)} + 507\sin(3t)t + 156te^{(2t)} + 169\cos(3t))e^{(-2t)}$$

```
> combine(solucao);
```

$$y(t) = -\frac{179}{507}e^{(-2t)}\sin(3t) + \frac{8}{169}e^{(-2t)}\cos(3t) - \frac{8}{169} + \frac{1}{2}e^{(-2t)}\sin(3t)t + \frac{2}{13}t$$

Exemplo 7.7 *Vamos resolver o problema de valor inicial* $y^{(4)} - 10y'' + 9 = 0$, $y(0) = y'(0) = y''(0) = y'''(0) = 0$ *e usar a resposta para definir uma função* $F(x)$.

```
> eq := (D@@4)(y)(x)- 10*(D@@2)(y)(x) + 9 = 0;
```

$$eq := (D^{(4)})(y)(x) - 10(D^{(2)})(y)(x) + 9 = 0$$

```
cond := y(0) = 0, D(y)(0) = 0, (D@@2)(y)(0) = 0, (D@@3)(y)(0) = 0;
```

$$cond := y(0) = 0, D(y)(0) = 0, (D^{(2)})(y)(0) = 0, (D^{(3)})(y)(0) = 0$$

```
> sol := dsolve({eq, cond});
```

$$sol := y(x) = -\frac{9}{200}e^{(\sqrt{10}x)} - \frac{9}{200}e^{(-\sqrt{10}x)} + \frac{9}{20}x^2 + \frac{9}{100}$$

```
> F := unapply(rhs(sol), x);
```

$$F := x \to -\frac{9}{200}e^{(\sqrt{10}x)} - \frac{9}{200}e^{(-\sqrt{10}x)} + \frac{9}{20}x^2 + \frac{9}{100}$$

Dessa forma, podem ser feitos outros cálculos com a função $F(x)$ bem como a construção do seu gráfico.

7.3 Sistemas de equações diferenciais

Sistemas de equações diferenciais podem ser resolvido de maneira semelhante à resolução de uma única EDO. Para isso, basta escrever todas as equações envolvidas entre chaves e usar como primeiro parâmetro do `dsolve`. Se houver condições iniciais, elas também devem aparecer entre chaves, juntamente com as equações.

Exemplo 7.8 *Consideremos o sistema*

$$\begin{cases} f'(x) &= f(x) + g(x) \\ g'(x) &= 3f(x) - g(x) \end{cases}$$

Vamos formar um conjunto com essas equações e resolvê-lo com o `dsolve`.

```
> sis1 := {D(f)(x) = f(x) + g(x), D(g)(x) = 3*f(x) - g(x)};
```

$$sis1 := \{D(f)(x) = f(x) + g(x),\ D(g)(x) = 3f(x) - g(x)\}$$

```
> dsolve(sis1);
```

$$\{g(x) = -3\_C1e^{(-2x)} + \_C2e^{(2x)},\ f(x) = \_C1e^{(-2x)} + \_C2e^{(2x)}\}$$

Exemplo 7.9 *Resolver o sistema*

$$\begin{cases} y''(t) + x'(t) &= \cos(t) \\ x''(t) - y(t) &= \operatorname{sen}(t) \end{cases}$$

com as condições iniciais $y(0) = -1, y'(0) = -1, x(0) = 1, x'(0) = 0$.

```
> sis2 :=  diff(y(t), t$2) + diff(x(t), t) = cos(t),
>          diff(x(t), t$2) - y(t) = sin(t);
```

$$sis2 := \left(\frac{\partial^2}{\partial t^2}y(t)\right) + \left(\frac{\partial}{\partial t}x(t)\right) = \cos(t),\ \left(\frac{\partial^2}{\partial t^2}x(t)\right) - y(t) = \sin(t)$$

```
> ini2 :=  y(0) = -1, D(y)(0) = -1, x(0) = 1, D(x)(0) = 0;
```

$$ini2 := x(0) = 1,\ y(0) = -1,\ D(y)(0) = -1,\ D(x)(0) = 0$$

```
> dsolve({sis2, ini2});
```

$$\{y(t) = -\cos(t) - \sin(t),\ x(t) = \cos(t)\}$$

7.4 Solução em série de potências

Há duas maneiras de se obter a solução de uma equação diferencial em série de potências: usando o comando `dsolve` ou o comando `powsolve` do pacote `powseries`.

Para obter séries de potências com o `dsolve`, basta acrescentar uma opção `series` ou `type=series` como terceiro parâmetro:

`dsolve({`*equação, condições*`}, função,` `series)`

Um comando do tipo `Order =` n obtém desenvolvimento em série a ser feito posteriormente até ordem n.

Para obtermos séries de potências de $(x-a)$, devemos fornecer as condições iniciais definidas em a (por exemplo, $y(a), y'(a), y''(a), \ldots$). Também podem ser resolvidos dessa forma sistemas de equações diferenciais.

A desvantagem no uso do comando `dsolve` é que ele não fornece o termo geral da série apresentada como solução da EDO dada.

Exemplo 7.10 *Vamos obter o desenvolvimento em série de potências da solução de* $y''' - xy' = x^2 + e^x$, *com as condições* $y(1) = 0$, $y'(1) = 0$ *e* $y''(1) = 0$.

```
> Order := 6;
> EDO := (D@@3)(y)(x) - x*D(y)(x) = x^2 + exp(x);
```

$$EDO := (D^{(3)})(y)(x) - xD(y)(x) = x^2 + e^x$$

```
> cond1 := y(1)=0, D(y)(1)=0, (D@@2)(y)(1) = 0;
```

$$cond1 := y(1) = 0, D(y)(1) = 0, (D^{(2)})(y)(1) = 0$$

```
> dsolve({EDO, cond1}, y(x), series);
```

$$y(x) = (\frac{1}{6} + \frac{1}{6}e)(x-1)^3 + (\frac{1}{12} + \frac{1}{24}e)(x-1)^4 + (\frac{1}{40} + \frac{1}{60}e)(x-1)^5 + O((x-1)^6)$$

Exemplo 7.11 *Obter a solução em série de potências de* $y''' - xy' = x^2 - e^x$ *com as condições iniciais* $y(0) = 0$, $y'(0) = 1$, $y''(0) = -3$.

```
> EDO := (D@@3)(y)(x) - x*D(y)(x) = x^2 + exp(x):
> Order := 9:
> cond2 := y(0)=0, D(y)(0)=1, (D@@2)(y)(0) = -3;
```

$$cond2 := y(0) = 0, D(y)(0) = 1, (D^{(2)})(y)(0) = -3$$

```
> dsolve({EDO, cond2}, y(x), series);
```

$$y(x) = x - \frac{3}{2}x^2 + \frac{1}{6}x^3 + \frac{1}{12}x^4 - \frac{1}{40}x^5 + \frac{1}{180}x^6 + \frac{1}{560}x^7 - \frac{1}{2880}x^8 + O(x^9)$$

O pacote `powseries` possui o comando `powsolve` para resolver EDO em séries de potências:

`powsolve(`*equação*`)` ou `powsolve({`*equação, condições*`})`

As séries obtidas podem ser mostradas com o comando `tpsform(`*série, variável, ordem*`)` ou fornecidas como parâmetros para outros comandos desse pacote (veja seção 5.6.3).

Devem ser observados os seguintes itens:

- A equação fornecida ao `powsolve` deve ser definida com o comando `diff`;

- Se houver condições iniciais, elas devem ser calculadas em 0 ($y(0)$, $y'(0)$, $y''(0)$, etc.) e definidas com o operador diferencial `D`;
- O termo geral pode ser mostrado se for usado um comando da forma $a(\_k) = X(\_k)$ onde X deve ser o nome da série (primeiro parâmetro do `tpsform`).

Exemplo 7.12 *Vamos obter a solução da Equação de Legendre*

$$(1-x^2)y'' - 2xy' + n(n+1)y = 0$$

em série de potências.

```
> restart;
> legendre := (1-x^2)*diff(y(x),x$2) - 2*x*diff(y(x),x) + n*(n+1)*y(x)=0;
```

$$legendre := (1-x^2)\left(\frac{\partial^2}{\partial x^2}y(x)\right) - 2x\left(\frac{\partial}{\partial x}y(x)\right) + n(n+1)y(x) = 0$$

```
> with(powseries):
> solucao := powsolve(legendre):
> tpsform(solucao, x, 6);
```

$$C0 + C1x - \frac{1}{2}C0(n^2+n)x^2 - \frac{1}{6}C1(n^2+n-2)x^3 + \frac{1}{24}C0(n^2+n)(n^2+n-6)x^4 + \frac{1}{120}C1(n^2+n-2)(n^2+n-12)x^5 + O(x^6)$$

Para obter o termo geral da série obtida:

```
> a(_k) = solucao(_k);
```

$$a(\_k) = -\frac{a(\_k-2)(n^2+n+3\_k-2-\_k^2)}{\_k(\_k-1)}$$

Exemplo 7.13 *Resolver o problema de valor inicial $y'' - xy = 0$, $y(0) = 1$, $y'(0) = 1$ obtendo o desenvolvimento da solução em série de potências e o seu termo geral.*

```
> EDO := diff(y(x),x$2) - x*y(x) = 0;
```

$$EDO := \left(\frac{\partial^2}{\partial x^2}y(x)\right) - xy(x) = 0$$

```
> CI := y(0)=1, D(y)(0)=1;
```

$$CI := y(0) = 1,\ D(y)(0) = 1$$

```
> with(powseries):
> sol := powsolve({EDO, CI}):
> tpsform(sol,x,10);
```

$$1 + x + \frac{1}{6}x^3 + \frac{1}{12}x^4 + \frac{1}{180}x^6 + \frac{1}{504}x^7 + \frac{1}{12960}x^9 + O(x^{10})$$

```
> a(_k) = sol(_k);
```

$$a(\_k) = \frac{a(\_k-3)}{\_k(\_k-1)}$$

7.5 Resolução numérica de equações diferenciais

Uma grande variedade de equações diferenciais não pode ser resolvida de forma exata. Em algumas, a solução encontrada não é uma função elementar. Nesses casos, uma boa opção pode ser encontrar uma solução numérica para a equação que permita verificarmos valores em pontos particulares do seu domínio ou a construção de seu gráfico.

Para resolver numericamente um PVI, basta acrescentarmos ao `dsolve` uma opção `type = numeric` ou simplesmente `numeric`.

Exemplo 7.14 *Determinar a solução numérica da equação* $y' = x^3 + y^3$ *com a condição inicial* $y(0) = 0$.

```
> eq := diff(y(x), x) = x^3 + y(x)^3; ini := y(0) = 0;
```

$$eq := \frac{d}{dx}y(x) = x^3 + y(x)^3$$
$$ini := y(0) = 0$$

Esta simples equação não pode ser resolvida de forma exata. Note que o Maple não fornece resposta para o comando a seguir:

```
> dsolve({eq, ini}, y(x));
>
```

Agora, ao fazermos a opção `type=numeric`*, a resolução torna-se possível. A solução é retornada em forma de procedimento.*

```
> s := dsolve({eq, ini}, y(x), type=numeric);
```

$$s := \mathbf{proc}(x\_rkf45) \cdots \mathbf{end\ proc}$$

Podemos obter valores da solução em pontos escolhidos do domínio. Por exemplo, para $x = 0,5$ *e para* $x = 1$ *obtemos os seguintes valores:*

```
> s(0.5);
```

$$[x = 0.5,\ y(x) = 0.0156250895424338010]$$

```
> s(1);
```

$$[x = 1.,\ y(x) = 0.251212470149853429]$$

Pode ser usada uma opção do tipo `method`=*método* com o *método* a ser utilizado na resolução que pode ser `rkf45, rosenbrock, bvp, dverk78, lsode, gear, classical` ou `taylorseries`.

O formato da saída dos resultados pode ser especificado com uma opção `output`=*formato* onde o *formato* pode ser `procedurelist, listprocedure, piecewise` ou `array`.

Exemplo 7.15 *Resolver o PVI* $y'' - x^2y' + y^3 = 1,\ y(1) = 5,\ y'(1) = -3$. *Inicialmente, observe que se não for especificado uma resolução numérica, o Maple não consegue encontrar a solução.*

```
> eq2 := diff(y(x), x$2) - x^2*diff(y(x), x) + y(x)^3 = 1;
```

$$eq2 := \left(\frac{d^2}{dx^2}y(x)\right) - x^2\left(\frac{d}{dx}y(x)\right) + y(x)^3 = 1$$

```
> ini2 := y(1) = -3, D(y)(1) = 5;
```

$$ini2 := y(1) = -3,\ D(y)(1) = 5$$

```
> dsolve({eq2, ini2}, y(x));  # nao consegue solucao exata
>
```

Escolhendo agora uma resolução numérica com o método da série de Taylor e apresentação do resultado em forma de procedimento, verificamos os valores encontrados em $x = 1$ e em $x = 2$.

```
> s2 := dsolve({eq2, ini2}, y(x), type=numeric, method=taylorseries,
>                                                 output=procedurelist);
```

$$s2 := \mathbf{proc}(x\_taylorseries) \cdots \mathbf{end\ proc}$$

```
> s2(1);
```

$$[x = 1.,\ y(x) = -3.,\ \frac{d}{dx}y(x) = 5.]$$

```
> s2(2);
```

$$[x = 2.,\ y(x) = -0.018744924690623,\ \frac{d}{dx}y(x) = -43.198420324453]$$

Resolvemos o PVI novamente mudando apenas a forma de apresentação do resultado. Agora, escolhemos uma apresentação em forma de tabela. Escolhemos os valores do domínio mostrados na tabela como sendo $x = 0,5,\ 1,\ 1,5,\ 2,\ 2,5$ e 3.

```
> s2 := dsolve({eq2, ini2}, y(x), type=numeric, method=taylorseries,
>                         output=array([0.5, 1.0, 1.5, 2.0, 2.5, 3.0]));
```

$$s2 := \left[\begin{array}{c} \left[x,\ y(x),\ \frac{d}{dx}y(x)\right] \\ \left[\begin{array}{ccc} 0.5 & -1.5393634913541 & -7.4298332457591 \\ 1.0 & -3. & 5. \\ 1.5 & 2.4668554591271 & 17.023128340132 \\ 2.0 & -0.018744924689859 & -43.198420324446 \\ 2.5 & 17.259490131826 & -81.155139242810 \\ 3.0 & -58.459486061670 & -1943.3065403541 \end{array}\right] \end{array}\right]$$

Podemos construir o gráfico de uma solução numérica obtida com um comando `dsolve`. Para isso, basta usar um comando `odeplot` (veja exemplo 7.17).

7.6 Gráficos de soluções de EDO

O pacote `DEtools` possui um comando `DEplot` para construção de gráficos de soluções de EDO, até mesmo em casos em que o `dsolve` não consegue resolver a equação de forma exata:

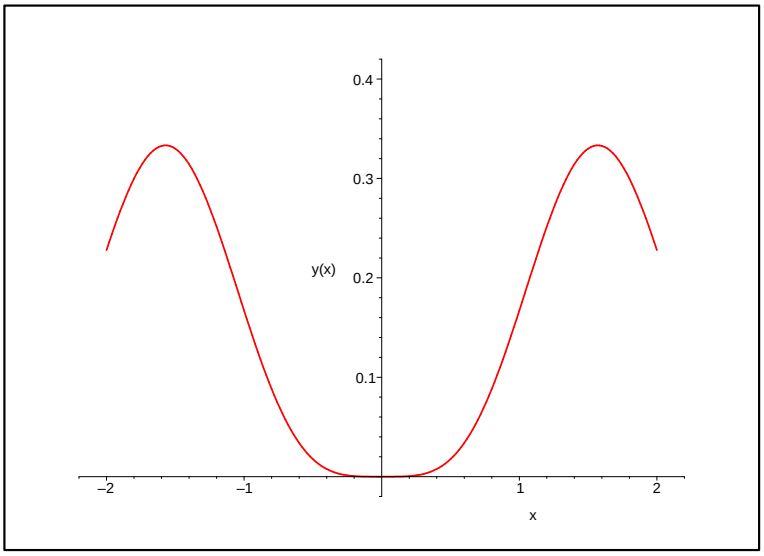
`DEplot(`*EDO, y(x), x=a..b,* `[[`*condições*`]],` *y=c..d, opções*`)`

Exemplo 7.16 *Construir o gráfico da solução de* $y'' + 4y = \operatorname{sen}^2(2x)$ *com as condições iniciais* $y(\pi) = 0$ *e* $y'(\pi) = 0$.

```
> with(DEtools):
> edo  := diff(y(x), x$2) + 4*y(x) = (sin(2*x))^2:
> ini := y(Pi) = 0, D(y)(Pi) = 0:
> opcoes := linecolor=red, stepsize=0.02:
> DEplot(edo, y(x), x=-2..2, [[ini]], y=0..0.4, opcoes);
```

Veja o gráfico na Figura 7.1.

Figura 7.1:



Outra opção é usar o comando `odeplot(`*solução, opções*`)` do pacote `plots`. Ele constrói o gráfico da solução retornada por um `dsolve`, conforme exemplificamos a seguir. Neste caso, é preciso que o `dsolve` tenha sido usado com um parâmetro `numeric` ou `type=numeric`.

Exemplo 7.17 *Vamos contruir o gráfico do PVI*

$$u''(t) + 0,125u'(t) + u(t) = 3\cos(2t),\ u(0) = 2,\ u'(0) = 0$$

(problema conhecido como Vibração Forçada com Amortecimento*). O gráfico é o da Figura 7.2.*

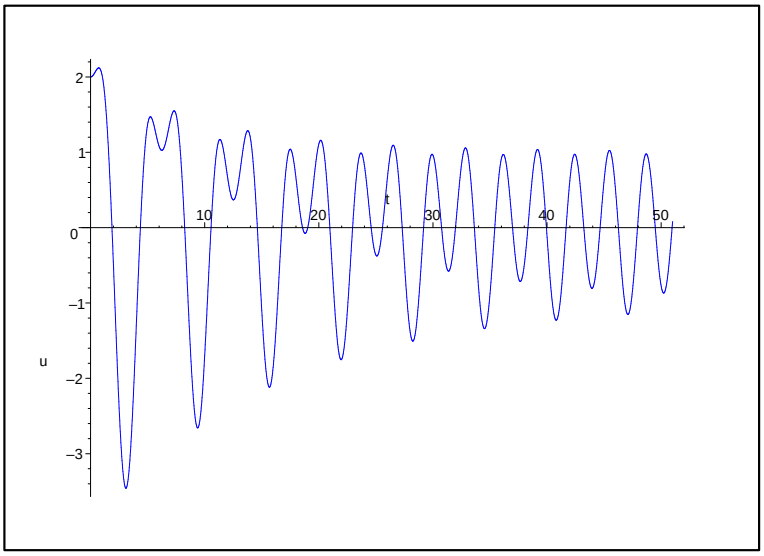
```
> with(plots):
> eq:= diff(u(t),t,t) + 0.125*diff(u(t),t) + u(t) = 3*cos(2*t):
> ini:= u(0)=2, D(u)(0)=0:
> sol: = dsolve({eq, ini}, u(t), numeric):
> odeplot(sol, t=0..51, thickness=2, color=blue, numpoints=2000);
```

O comando `dfieldplot` do pacote `DEtools` permite a construção de um campo de direções de uma EDO. O modo de usá-lo é

`dfieldplot(`*edo, var, domínio, opções*`)`

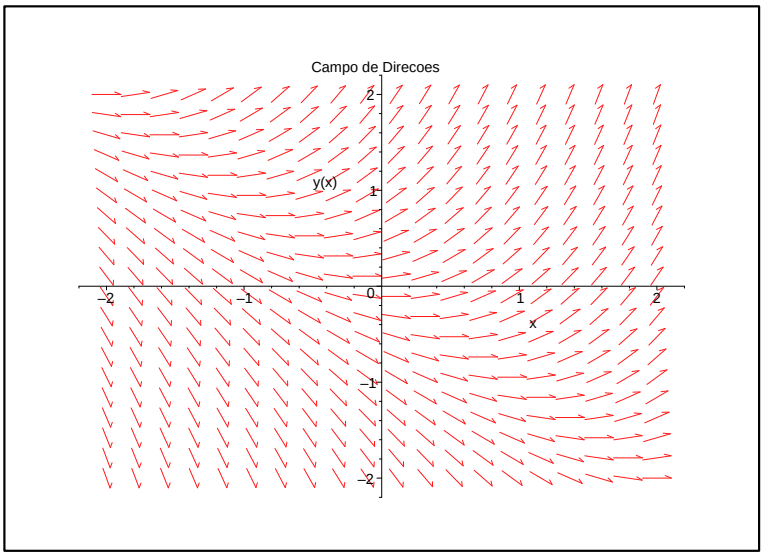
onde *edo* é a equação diferencial, *var* é a variável dependente, *domínio* é a variação da variável independente e *opções* controlam a apresentação do gráfico (como as do comando `plot`).

Figura 7.2:



Exemplo 7.18 *Vamos construir o campo de direções e algumas curvas integrais da equação* $y' - y = x$.

Figura 7.3:




```
> restart: with(plots): with(DEtools):
Warning, the name changecoords has been redefined
> eq := diff(y(x),x) - y(x) = x;
```

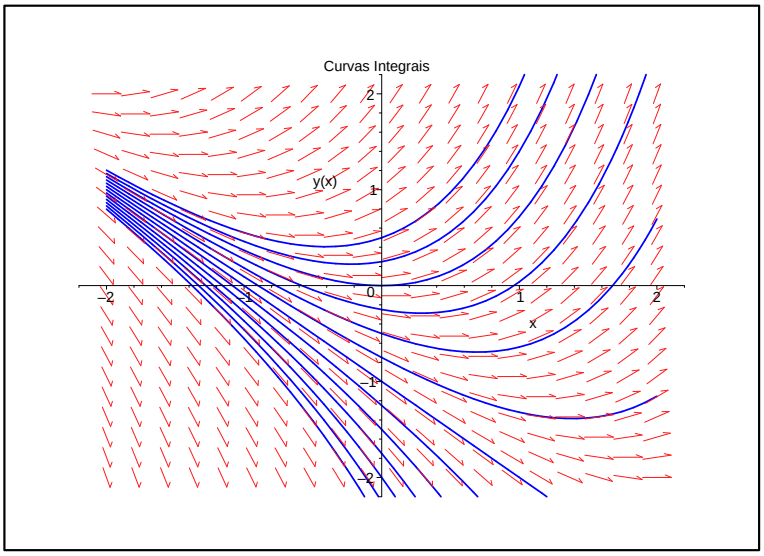
$$eq := \left(\frac{d}{dx}y(x)\right) - y(x) = x$$

```
> direcoes := dfieldplot(eq, y(x), x=-2..2, y=-2..2):
> display(direcoes, title="Campo de Direcoes", thickness=2);
```

Com isso, o Maple contrói o campo de direções da equação. Veja Figura 7.3.

Agora, vamos construir algumas curvas integrais da equação. Para isso, resolvemos a equação com um comando `dsolve`*:*

Figura 7.4:




```
> solucao := dsolve(eq);
```

$$solucao := y(x) = -x - 1 + e^x\_C1$$

Na `solucao`*, trocamos a constante genérica* $\_C1$ *por outra constante que vamos denotar por* $k/4$*:*

```
> solucao := rhs(subs(_C1=k/4, solucao));
```

$$solucao := -x - 1 + \frac{1}{4}e^x k$$

O comando `for k from -6 to 6` *faz com que o* k *assuma valores inteiros de* -6 *a* 6*. Para cada valor de* k *nesse intervalo, construímos um gráfico da* `solucao` *com o comando* `plot` *e guardamos cada resultado em* $g[k]$*:*

```
> for k from -6 to 6 do
>   g[k] := plot(solucao, x=-2..2, y=-2..2, color=blue, thickness=3):
> end do:  # para encerrar o comando "do" iniciado acima
> display(direcoes, seq(g[k],k=-6..6), title="Curvas Integrais" );
```

O comando `display` *mostra todos os gráficos construídos anteriormente. Para mostrar todos os gráficos* $g[k]$ *das curvas integrais, usamos um comando* `seq(g[k], k=-6..6)`*. O resultado é mostrado na Figura 7.4.*

Exemplo 7.19 *Vamos construir o campo de direções e algumas curvas integrais da equação* $y' = \dfrac{x^2}{1-y^2}$.

O campo de direções é construído com os comandos a seguir. Veja o resultado na Figura 7.5.

```
> with(plots): with(DEtools):
> eqdif := diff(y(x),x)=x^2/(1-y(x)^2);
```

$$eqdif := \frac{d}{dx}y(x) = \frac{x^2}{1-y(x)^2}$$

Figura 7.5:

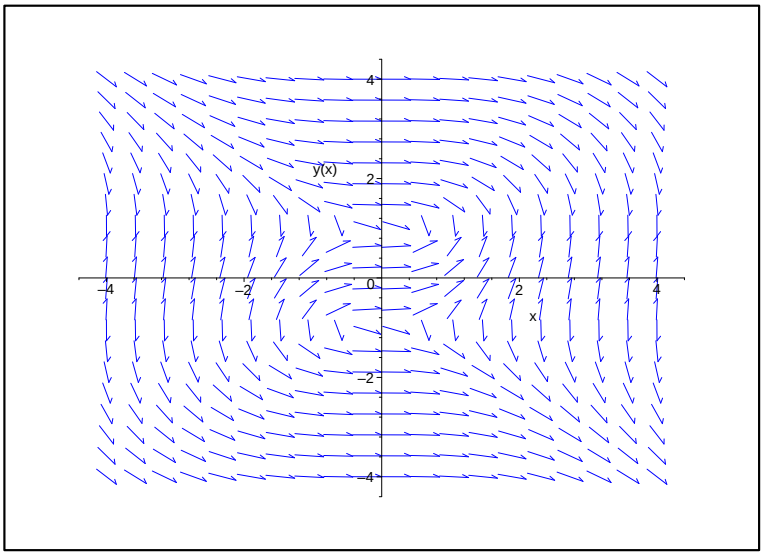
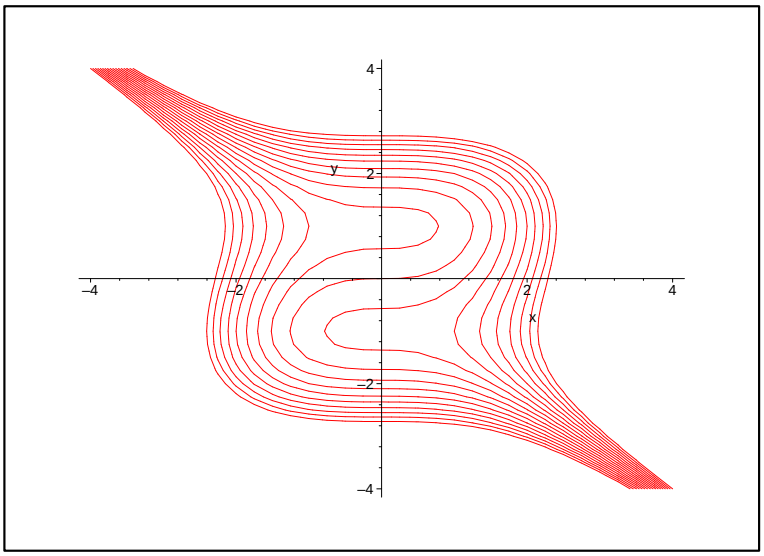


Figura 7.6:




```
> direcoes := dfieldplot(eqdif, y(x), x=-4..4, y=-4..4, color=blue):
> display(direcoes, thickness=1);
```

Agora, resolvemos a equação com um comando `dsolve`. *Obtemos uma solução na forma implícita onde aparece a constante genérica* $\_C1$. *Trocamos o* $\_C1$ *por outra constante que preferimos chamar* $(k-8)/2$.

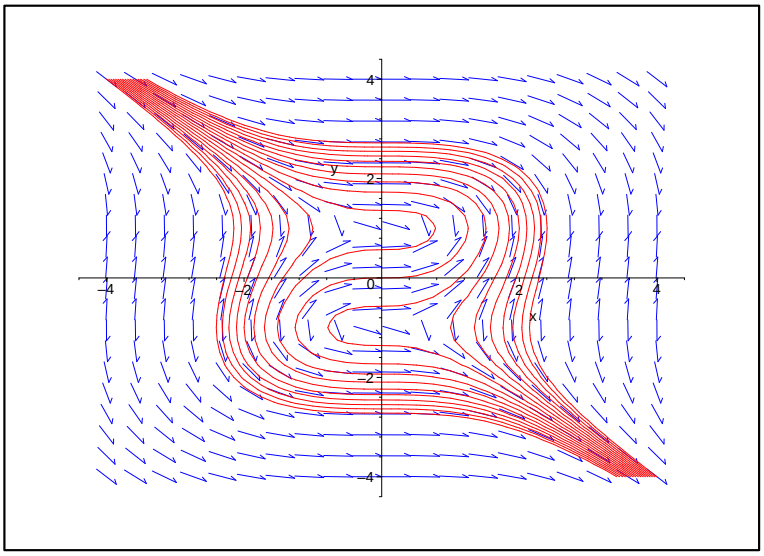
```
> sol := dsolve(eqdif, y(x), implicit);
```

$$sol := \frac{x^3}{3} - y(x) + \frac{1}{3}y(x)^3 + \_C1 = 0$$

```
> sol := subs(_C1 = (k-8)/2, sol);
```

$$sol := \frac{x^3}{3} - y(x) + \frac{1}{3}y(x)^3 + \frac{k}{2} - 4 = 0$$

Figura 7.7:



Usamos um comando `for` *para fazer o* k *variar de* 0 *a* 16. *A cada valor de* k *assim obtido, construímos o gráfico da curva integral com um comando* `implicitplot` *e guardamos cada resultado em* `graf[k]`.

```
> for k from 0 to 16 do
>   graf[k] := implicitplot(sol, x=-4..4, y=-4..4, thickness=2,
>              numpoints=1500):
> end do:
```

Usamos um comando `display` *para mostrar toda a seqüência de gráficos* `graf[k]`. *O resultado é o da Figura 7.6.*

```
> display(seq(graf[k], k=0..16));
```

Para mostrar as curvas integrais juntamente com o campo de direções, basta usar um comando `display` *com parâmetros formados pela seqüência de gráficos e pelo gráfico das* `direcoes`. *Veja a Figura 7.7.*

```
> display(seq(graf[k], k=0..16), direcoes);
```

7.7 Gráficos de soluções de sistemas de EDO

Um gráfico das soluções de um sistema de equações diferenciais pode ser construído com o comando `DEplot` do pacote `DEtools`:

`DEplot`({*equações*}, {*variáveis*}, *domínio, condições iniciais, opções*)

onde especificamos as *equações* do sistema entre chaves, as *variáveis* (funções) dependentes, o *domínio* (variação dos parâmetros), as *condições iniciais* na forma de lista de listas ($[[x(t_0) = x_0, y(t_0) = y_0], [x(t_1) = x_1, y(t_1) = y_1], \cdots]$) e as *opções* que controlam a apresentação do gráfico (`color`: cor do campo de vetores, `linecolor`: cor da solução, `stepsize`: incremento do parâmetro, entre outras).

Exemplo 7.20 *Vamos construir o gráfico da solução do sistema*

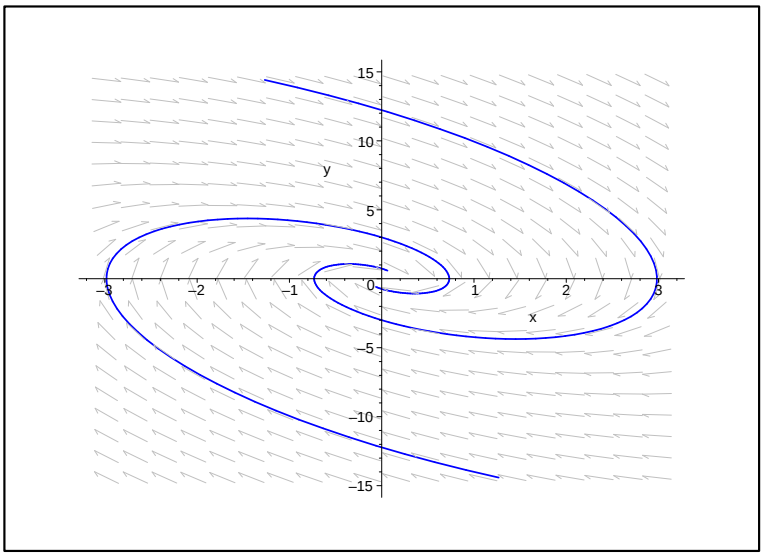
$$\begin{cases} y'(t) + y(t) + 3x(t) &= 0 \\ y(t) &= 2x'(t) \end{cases}$$

com as condições iniciais $x(0) = 0, y(0) = 3$ *para a primeira solução,* $x(0) = 0, y(0) = -3$ *para a segunda e o parâmetro* t *da solução variando de* -3 *a* 3 *com passo* $0,05$. *Queremos um gráfico na cor azul* `(linecolor = blue)` *e um campo de vetores na cor cinza* `(color = gray)`. *Ver Figura 7.8.*

Como padrão, é desenhado um campo de direções juntamente com o gráfico da solução, a não ser que seja acrescentado ao comando `DEplot` *uma opção* `arrows=NONE`.

```
> with(DEtools):
> eq1 := diff(y(t), t) + y(t) + 3*x(t) = 0:
> eq2 := y(t) = 2*diff(x(t), t):
> ini1 := x(0) = 0, y(0) = 3:
> ini2 := x(0) = 0, y(0) = -3:
> DEplot({eq1, eq2}, [x(t), y(t)], t=-3..3, [[ini1], [ini2]],
>                       linecolor=blue, stepsize=0.05, color=gray);
```

Figura 7.8:



Os gráficos das soluções de sistemas com um número maior de equações podem ser construídos de maneira semelhante ao que foi feito no exemplo anterior.

7.8 Equações diferenciais parciais

Diversos tipos de Equações Diferenciais Parciais (EDP) podem ser resolvidas com o comando `pdsolve(equação, opções)`. Na definição da equação, podemos usar o `diff` ou o operador diferencial `D`.

Nas soluções apresentadas pelo `pdsolve`, as constantes arbitrárias são denotadas por $\_C1, \_C2, \ldots, \_c_1, \_c_2, \ldots$ e as funções arbitrárias por $\_F1, \_F2, \ldots$

Exemplo 7.21 *Iniciamos resolvendo a equação* $x\dfrac{\partial}{\partial y} f(x,y) - y\dfrac{\partial}{\partial x} f(x,y) = 0$.

```
> edp1 := x*D[2](f)(x,y)-y*D[1](f)(x,y) = 0;
```

$$edp1 := xD[2](f)(x,y) - yD[1](f)(x,y) = 0$$

```
> pdsolve(edp1);
```

$$f(x,y) = \_F1(x^2+y^2)$$

Exemplo 7.22 *Usamos o* `declare` *do pacote* `PDEtools` *para indicar em forma de índices as derivadas que aparecem na equação da onda* $u_{tt} = c^2 u_{xx}$.

```
> with(PDEtools): declare(u(x, t));
```

$$u(x,t) \text{ will now be displayed as } u$$

```
> edp2 := diff(u(x, t), t$2) = c^2*diff(u(x, t), x$2);
```

$$edp2 := u_{t,t} = c^2 u_{x,x}$$

```
> pdsolve(edp2);
```

$$u = \_F1(ct+x) + \_F2(ct-x)$$

```
> undeclare(u);
```

$$u(x,t) \text{ will now be displayed *as is*}$$

Exemplo 7.23 *A equação do calor pode ser resolvida acrescentando-se uma opção HINT=‘*‘ ao comando* `pdsolve` *para ele usar separação de variáveis e uma opção* `build` *para ele resolver qualquer EDO que apareça na resolução.*

```
> edp3 := diff(u(x, t), t) - k*diff(u(x, t), x$2) = 0;
```

$$edp3 := \left(\frac{\partial}{\partial t}u(x,t)\right) - k\left(\frac{\partial^2}{\partial x^2}u(x,t)\right) = 0$$

```
> sol := pdsolve(edp3, u(x, t), HINT=‘*‘, build);
```

$$sol := u(x,t) = \_C3 e^{(k\_c_1 t)} \_C1 e^{(\sqrt{\_c_1}x)} + \frac{\_C3 e^{(k\_c_1 t)} \_C2}{e^{(\sqrt{\_c_1}x)}}$$

Fazendo as constantes iguais a 1, obtemos:

```
> S := eval(rhs(sol), {_C1 = 1, _C2 = 1, _C3 = 1, k = 1, _c[1] = 1});
```

$$S := e^t e^x + \frac{e^t}{e^x}$$

Exemplo 7.24 *Vamos resolver agora a equação de Laplace no caso bidimensional.*

```
> edp4 := diff(f(x, y), x$2) + diff(f(x, y), y$2) = 0;
```

$$edp4 := \left(\frac{\partial^2}{\partial x^2}f(x,y)\right) + \left(\frac{\partial^2}{\partial y^2}f(x,y)\right) = 0$$

```
> pdsolve(edp4);
```

$$f(x,y) = \_F1(y+Ix) + \_F2(y-Ix)$$

```
> pdsolve(edp4, f(x, y), HINT=‘*‘, build);
```

$$f(x,y) = e^{(\sqrt{\_c_1}x)} \_C1\_C3 \sin(\sqrt{\_c_1}y) + e^{(\sqrt{\_c_1}x)} \_C1\_C4 \cos(\sqrt{\_c_1}y) + \frac{\_C2\_C3 \sin(\sqrt{\_c_1}y)}{e^{(\sqrt{\_c_1}x)}} + \frac{\_C2\_C4 \cos(\sqrt{\_c_1})y)}{e^{(\sqrt{\_c_1}x)}}$$

7.9 Gráficos de soluções de EDP

O comando `PDEplot` do pacote `PDEtools` permite construir gráfico de solução de uma EDP:

`PDEplot(`*edp, var, ini, domínio, opções*`)`

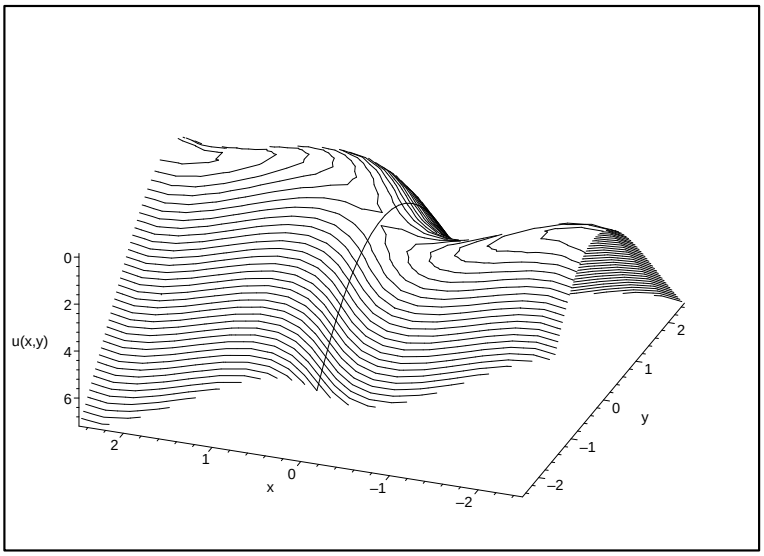
onde *edp* é a equação, *var* é a variável (função) dependente, *ini* é uma lista com as equações paramétricas de uma curva no espaço tridimensional, *domínio* é a variação do parâmetro da curva e *opções* são as opções que controlam a apresentação do gráfico (algumas coincidem com as opções do comando `plot3d`, veja seção 4.5.2).

Exemplo 7.25 *Vamos construir o gráfico da solução $u(x, y)$ da EDP*

$$\frac{\partial}{\partial x}u(x,y) + \cos(2x)\frac{\partial}{\partial y}u(x,y) = -\sin(y),$$

com condição inicial dada pela curva $f(s) = (0, s, 1 + s^2)$, $-2 \leq s \leq 2$.

Figura 7.9:




```
> restart: with(PDEtools):
> edp := diff(u(x, y), x) + cos(2*x)*diff(u(x, y), y) = -sin(y);
```

$$edp := \left(\frac{\partial}{\partial x}u(x,y)\right) + \cos(2x)\left(\frac{\partial}{\partial y}u(x,y)\right) = -\sin(y)$$

```
> ini := [0, s, 1 + s^2]:
> PDEplot(edp, u(x, y), ini, s=-2..2, style=patchcontour,
                             orientation=[70,-140], contours=20);
```

O gráfico é o da Figura 7.9.

7.10 Equações diferenciais e a transformada de Laplace

O pacote `inttrans` fornece comandos para o cálculo de várias transformadas integrais e suas inversas.

A Transformada de Laplace de uma função F(t) é definida por

$$\mathscr{L}\{F(t)\} = \int_0^\infty F(t)e^{-st}dt$$

O comando de `inttrans` que corresponde a isso é o `laplace(`*F(t), t, s*`)`.

Se $\mathscr{L}\{F(t)\} = f(s)$ então $F(t)$ chama-se a Transformada de Laplace Inversa de $f(s)$ e é usual denotar-se por $F(t) = \mathscr{L}^{-1}\{f(s)\}$. O comando do pacote `inttrans` correspondente é o `invlaplace(`*f(s), s, t*`)`.

Exemplo 7.26 *Fazemos um cálculo simples de Transformada de Laplace de uma função polinomial. Depois, a Transformada de Laplace Inversa aplicada ao resultado recupera a função inicial.*

```
> with(inttrans):
> y := laplace( 1 + 3*t - 2*t^5, t, s);
```

$$y := \frac{1}{s} + \frac{3}{s^2} - \frac{240}{s^6}$$

```
> invlaplace(y, s, t);
```

$$1 + 3t - 2t^5$$

Diversos tipos de Equações Diferenciais Ordinárias podem ser resolvidas com a Transformada de Laplace. Em linhas gerais, a resolução por esse método consiste no cálculo da transformada da equação usando-se a propriedade

$$\mathscr{L}\{F^n(t)\} = s^n\mathscr{L}\{F(t)\} - s^{n-1}F(0) - s^{n-2}F'(0) - F^{n-1}(0).$$

Em seguida, isola-se o valor de $\mathscr{L}\{F(t)\}$ e calcula-se a transformada inversa para obter a solução da equação dada.

Podemos fornecer $f^{(n)}(t)$ na forma `diff(y(t),t$n)` e as condições de contorno $y'(a) = b, \cdots, y^{(n)}(a) = c$ na forma `D(y)(a)=b`, $\cdots$, `(D@@n)(y)(a)=c`.

Exemplo 7.27 *Vamos resolver o problema de valor inicial*

$$y''(t) + 5y'(t) + 6y(t) + e^t = \operatorname{sen} t, \quad y(0) = 0,\ y'(0) = 1.$$

Inicialmente, definimos uma variável auxiliar `equacao` *com a equação dada:*

```
> restart;
> with(inttrans):
> equacao := diff(y(t),t$2) + 5*diff(y(t),t) + 6*y(t) + exp(t) = sin(t);
```

$$equacao := \frac{\partial^2}{\partial t^2}y(t) + 5\frac{\partial}{\partial t}y(t) + 6y(t) + \exp(t) = \sin(t)$$

Calculamos a transformada da `equacao`*:*

```
> Leq := laplace(equacao, t, s);
```

$$Leq := s\,(s\,laplace(y(t),t,s) - y(0)) - D(y)(0) + 5s\,laplace(y(t),t,s) - 5y(0) + 6\,laplace(y(t),t,s) + \frac{1}{s-1} = \frac{1}{s^2+1}$$

Substituímos as condições de contorno dadas e trocamos `laplace(y(t),t,s)` *por* `f(s)`*:*

```
> LeqS := subs(laplace(y(t),t,s)=f(s), y(0)=0,D(y)(0)=1, Leq);
```

$$LeqS := s^2 f(s) - 1 + 5sf(s) + 6f(s) + \frac{1}{s-1} = \frac{1}{s^2+1}$$

Isolamos o valor de `f(s)`*:*

```
> f(s) := solve(LeqS, f(s));
```

$$f(s) := \frac{s^3 + 2s - 2s^2 - 3}{s^5 + 2s^3 + 4s^4 - 2s^2 + s - 6}$$

Finalmente, calculamos a transformada inversa de `f(s)` *que é a solução do problema proposto:*

```
> y(t) = invlaplace(f(s),s,t);
```

$$y(t) = -\frac{1}{12}e^t - \frac{27}{20}e^{(-3t)} + \frac{23}{15}e^{(-2t)} - \frac{1}{10}\cos(t) + \frac{1}{10}\sin(t)$$

Para obter a solução da equação sem as passagens intermediárias (como foi feito neste exemplo) basta usar um comando `dsolve`*, acrescentando uma opção* `method=laplace`*:*

```
> dsolve({equacao, y(0)=0, D(y)(0)=1}, y(t), method=laplace):
```

7.11 Exercícios

1) Resolva as seguintes equações diferenciais:
   a) $y' = \dfrac{4y}{x} + x\sqrt{y}$;
   b) $y' = y\,\mathrm{tg}\,x + \cos x$;
   c) $x(y')^2 + 2xy' - y = 0$;
   d) $y = (1+y')x + (y')^2$;
   e) $y''' + y' = \sec x$;
   f) $y''' + y'' + y' + y = xe^x$;
   g) $x^2y'' - 4xy' + 6y = x$.

2) Mostre que $x^2 \ln y + \frac{1}{3}(y^2+1)^{3/2} = C$ é uma solução implícita da equação diferencial

$$y' = \frac{-2xy\ln y}{x^2 + y^2\sqrt{y^2+1}}.$$

3) Determine a solução geral do sistema

$$\begin{cases} \dfrac{d^2y}{dx^2} + 2y + 4z &= e^x \\ \dfrac{d^2z}{dx^2} - y - 3z &= -x \end{cases}$$

4) Resolva o sistema de equações diferenciais

$$\begin{cases} Y' - Z' - 2Y + 2Z &= \mathrm{sen}\,t \\ Y'' + 2Z' + Y &= 0 \end{cases}$$

sujeito às condições $Y(0) = Y'(0) = Z(0) = 0$.

5) Construa os campos de direções e as curvas integrais das equações:

a) $y' = \dfrac{x+y}{x-y}$;

b) $y' = \dfrac{1}{2}(-x - \sqrt{x^2+4y})$;

c) $y' = -\dfrac{\operatorname{sen} y}{y}$.

Capítulo 8

Noções de programação com o Maple

O Maple pode ser utilizado também como linguagem de programação para cálculos algébricos. Apesar de possuir poucos comandos para serem utilizados com esse fim, a combinação deles pode gerar praticamente uma infinidade de novos comandos, limitados apenas pela criatividade e habilidade do programador.

Um usuário que já tenha tido experiência anterior com outras linguagens de programação como C, Pascal, FORTRAN ou BASIC, achará tudo muito familiar e certamente não terá dificuldade em dominar esses novos conceitos.

Um dos manuais originais do Maple, o *"Programming Guide"* [18], elaborado pela Waterloo Maple, possui mais de 600 páginas. Assim, pode-se perceber que há muito o que se estudar sobre esse assunto.

Neste capítulo usaremos várias vezes o comando `print(v1, v2, ...)` cuja função é mostrar os valores das variáveis `v1, v2, ....` No modo interativo da execução, esse comando não é necessário, uma vez que ele equivale a digitar simplesmente:

```
> v1; v2; ...
```

8.1 A declaração condicional

Ao tentar codificar determinado algoritmo em comandos de uma linguagem de programação, muitas vezes ficamos na situação de precisar definir que determinado grupo de comandos vai ser executado sob determinadas condições, ou então escolher um entre vários grupos de comandos para ser executado.

Para isso, o Maple possui a declaração condicional `if` cuja forma de usar mais simplificada é:

`if` *condição* `then` *comando* `end if;`

onde *condição* é uma expressão lógica que deve ser avaliada em verdadeira ou falsa (`true` ou `false`) e o *comando* (ou grupo de comandos) escrito depois da palavra `then` e antes de `end if` será executado somente se a *condição* for verdadeira.

Exemplo 8.1 *Consideremos a seguinte linha de comando:*

```
> if x < 100 then print(‘x é menor do que 100‘) end if;
```

Neste caso, o comando `print(‘x é menor do que 100‘)` *será executado somente se tivermos* $x < 100$. *O* x *precisa ter sido definido antes desse comando, senão o Maple mostra uma mensagem de erro como a mostrada a seguir:*

```
Error, cannot evaluate boolean: x < 100
```

É comum escrever um `if` *ocupando mais de uma linha, escrevendo algumas linhas mais avançadas para a direita. Por exemplo, é usual escrever o* `if` *anterior na forma:*

```
> if x < 0 then
>   print(‘x é menor do que 100‘)
> end if;
```

Exemplo 8.2 *A expressão lógica do* `if` *pode ser composta de várias condições ligadas por operadores lógicos como* `and`, `or` *ou* `not`. *Consideremos o seguinte* `if` *distribuído por várias linhas:*

```
> if type(x, numeric) and x > 0 then
>   y := sqrt(x);
>   z := ln(x);
>   w := x^x;
> end if;
```

Neste caso, a expressão lógica a ser avaliada é `type(x, numeric) and x > 0` *que é verdadeira somente se* x *for do tipo numérico e for positivo. Se a expressão for falsa, as atribuições de valores a* y, z *e* w *serão ignoradas.*

Observe que é mais fácil entender um comando desses escrito em várias linhas do que se ele ocupasse uma única linha.

Exemplo 8.3 *A seguir, definimos os valores de inteiros* $m = 4$ *e* $n = 10$ *e definimos um bloco de comandos que será executado somente se* $m < n$.

```
> m := 4: n := 10:
> if (m < n) then
>   temp := m;
>   m := n;
>   n := temp;
> end if:
> 'm' = m, 'n' = n;
```

$$m = 10, n = 4$$

Os comandos depois do `then` *têm por objetivo trocar os valores das variáveis* m *e* n.

O que determina se os comandos "dentro" do `if` *vão ser mostrados na sua execução ou não é a utilização de um ponto e vírgula ou dois pontos depois do* `end if`.

Uma forma mais completa da declaração condicional `if` é

`if` *condição* `then` *comando1* `else` *comando2* `end if;`

Neste caso, se a *condição* for verdadeira será executado o *comando1*; senão, será executado o *comando2*. No lugar de *comando1* ou *comando2* podemos ter vários comandos.

Exemplo 8.4 *No* `if` *mostrado a seguir, temos* $y := x$ *se* $x \geq 0$ *e* $y := -x$ *se* $x < 0$.

```
> x := -5:
> if x >= 0 then y := x else y := -x end if;
```

$$y := 5$$

Não importa que seja colocado sinal de `:` *ou* `;` *antes do* `else` *ou antes do* `end if`, *de modo que o* `if` *anterior é equivalente a*

```
> if x >= 0 then y := x; else y := -x; end if;
```

e também é equivalente a

```
> if x >= 0 then y := x: else y := -x: end if;
```

Exemplo 8.5 *Podemos ter* `if` *"dentro" de outro* `if`*, conforme mostrado a seguir. Neste exemplo, o M é escolhido como sendo o maior entre os valores de a, b e c. Para isso, inicialmente é feita uma comparação entre a e b. Se a for o maior, então b não tem chance de ser o maior dos três e, por isso, resta comparar a com c para saber qual é o maior. Se a não for maior do que b, então a deve ser descartado do cálculo do maior dos três, restando apenas comparar b com c.*

```
> a := -4:  b := 1:  c := 3:
>
> if a > b then
>   if a > c then
>     M := a
>   else
>     M := c
>   end if
> else
>   if c > b then
>     M := c
>   else
>     M := b
>   end if
> end if:
>
> 'M' = M;
```

$$M = 3$$

A forma mais completa do `if` é a mostrada a seguir, onde pode aparecer uma *condição1* e uma ou várias outras condições *condição2*, *condição3*, ..., *condiçãoN*.

```
if      condição1  then  comando1
elif    condição2  then  comando2
elif    condição3  then  comando3
...
else    comandoN
end if;
```

Se *condição1* for verdadeira, então é executado o *comando1* e o `if` encerra. Senão, é verificada a *condição2* e, se ela for verdadeira, é executado o *comando2* e o `if` encerra. E assim todas as condições vão sendo testadas até chegar em uma condição verdadeira. Se todas as condições testadas forem falsas então será executado o comando que estiver depois do `else`, se esse comando tiver sido definido.

Exemplo 8.6 *O* `if` *a seguir testa se um inteiro n dado é divisível por 2, 3, 5 ou 7. Se for, é mostrada uma mensagem específica e o quociente Q da divisão. A função* $\text{frac}(x)$ *calcula a parte fracionária de x.*

```
> n := 35:   # inteiro dado
>
> if   frac(n/2) = 0 then print(‘É divisível por 2‘); Q := n/2;
> elif frac(n/3) = 0 then print(‘É divisível por 3‘); Q := n/3;
> elif frac(n/5) = 0 then print(‘É divisível por 5‘); Q := n/5;
> elif frac(n/7) = 0 then print(‘É divisível por 7‘); Q := n/7;
> else print(‘Não é divisível por 2, nem por 3, nem por 5 e nem por 7‘)
> end if;
```

É divisível por 5

$$Q := 7$$

Note que a estrutura condicional acima detecta apenas que n é divisível por 5*. O fato dele também ser divisível por* 7 *foi ignorado.*

8.2 Estruturas de repetição

A repetição na execução de comandos é essencial para quase toda computação. O Maple possui os comandos `for` e `while` que são bastante eficientes para a repetição de determinadas ações.

A forma simplificada e a mais utilizada do comando `for` é

`for` *variável* `from` *valor1* `to` *valor2* `do` *comando* `end do`

e funciona da seguinte maneira: o *comando* (ou grupo de comandos) escrito depois do `do` e antes do `end do` é executado para cada valor que a *variável* assumir no intervalo [*valor1*, *valor2*]. A *variável* inicia com *valor1*, vai sendo aumentada sucessivamente de 1 em 1 e, enquanto seu valor for menor ou igual a *valor2*, o *comando* (ou grupo de comandos) vai sendo executado. Assim que a variável assumir um valor maior do que *valor2*, o `for` encerra.

Exemplo 8.7 *O* `for` *a seguir mostra* $\sqrt{i}$*, com i assumindo todos os valores inteiros de 2 a 9.*

```
> for i from 2 to 9 do print(sqrt(i)) end do;
```

$$\sqrt{2}$$
$$\sqrt{3}$$
$$2$$
$$\sqrt{5}$$
$$\sqrt{6}$$
$$\sqrt{7}$$
$$2\sqrt{2}$$
$$3$$

Exemplo 8.8 *Dadas duas listas* $L1$ *e* $L2$ *de mesmo tamanho, substituir* $L1[i]$*, o i-ésimo elemento de* $L1$*, por* $(L1[i] + L2[i])/2$*, a média aritmética entre o i-ésimo elemento de* $L1$ *e o i-ésimo elemento de* $L2$*. Depois do cálculo de cada média, atribuir o valor* 0 *a cada elemento de* $L2$*.*

```
> L1 := [5, 7, 1, 1, -3, 4, 10]:
> L2 := [1, 2, 0, 8, 1, 2, 6]:
>
> for n from 1 to nops(L1) do
>   x := L1[n]:
>   y := L2[n]:
>   z := (x + y)/2:
>   L1[n] := z:
>   L2[n] := 0:
> end do:
>
> L1, L2;
```

$$[3, \frac{9}{2}, \frac{1}{2}, \frac{9}{2}, -1, 3, 8],\ [0, 0, 0, 0, 0, 0, 0]$$

O `for` admite que seja definido um incremento para a variável:

`for` *variável* `from` *valor1* `to` *valor2* `by` *incremento* `do` *comando* `end do;`

O valor do *incremento* fornecido é usado para ir alterando o valor da *variável*, até que ela ultrapasse *valor2*. A seqüência de valores da *variável* é crescente se o incremento for positivo e é decrescente se ele for negativo.

Exemplo 8.9 *Calcular o valor de* $[n, \ln(n), \operatorname{sen}(\frac{\pi}{4}n)]$ *para n iniciando em 1 e aumentando de 3 em 3 enquanto for menor ou igual a 12.*

```
> for n from 1 to 12 by 3 do [n, ln(n), sin(Pi/4*n)] end do;
```

$$[1, 0, \frac{1}{2}\sqrt{2}]$$
$$[4, \ln(4), 0]$$
$$[7, \ln(7), -\frac{1}{2}\sqrt{2}]$$
$$[10, \ln(10), 1]$$

Observe que a última lista mostrada corresponde a $n = 10$*. Dessa forma, o valor* $n = 12$ *não é atingido.*

Exemplo 8.10 *Se o valor do incremento for negativo, então a seqüência dos valores assumidos pela variável é decrescente. Neste exemplo, k inicia com valor 10 e vai decrescendo de 4 em 4 enquanto seu valor não for menor do que 1. Para cada valor de k assim obtido, é calculada* $\sqrt{k}$*.*

```
> for k from 10 to 1 by -4 do evalf(sqrt(k)) end do;
```

$$3.162277660$$
$$2.449489743$$
$$1.414213562$$

Os valores mostrados correspondem a $k = 10$, $k = 6$ *e* $k = 2$*.*

Exemplo 8.11 *Vamos calcular o somatório* $\displaystyle\sum_{n=1}^{10} \frac{1}{n^2}$*. Para isso, iniciamos com o valor 0 para uma variável* `soma` *e, para cada valor de n de* 1 *a* 10*, calculamos* $1.0/n^2$ *e adicionamos o resultado a* `soma`*.*

```
> soma := 0:
> for n from 1 to 10 do
>   soma := soma + 1.0/n^2:
> end do;
```

$$soma := 1.0$$

$$soma := 1.250000000$$

$$soma := 1.361111111$$

$$soma := 1.423611111$$

$$soma := 1.463611111$$

$$soma := 1.491388889$$

$$soma := 1.511797052$$

$$soma := 1.527422052$$

$$soma := 1.539767731$$

$$soma := 1.549767731$$

Devido ao `;` *depois do* `end do` *os resultados parciais* 1.0, $1.0+1.0/4$, $1.0+1.0/4+1.0/9$, $1.0+1.0/4+1.0/9+1.0/16$, ... *são mostrados. A última soma mostrada,* $1,549767731$, *corresponde ao valor procurado.*

Podemos ter um comando `for` "dentro" de outro `for`. Isso se faz necessário quando usamos vários índices $i, j, \ldots$ para ter acesso aos elementos de vários conjuntos.

Exemplo 8.12 *Calcular o produto cartesiano dos conjuntos* $A = \{a, b, c\}$ *e* $B = \{1, 2, 3\}$. *Neste caso, precisamos de dois* `for` *encaixados, um para "pegar" cada elemento* $A[i]$ *do conjunto* A *e outro para "pegar" cada elemento* $B[j]$ *do conjunto* B. *Seja* $P = A \times B$. *Inicialmente, antes de percorrer os conjuntos* A *e* B, *o* P *não tem elemento, isto é,* $P = \{\ \}$ *(conjunto vazio). À medida que os pares de elementos* $[A[i], B[j]]$ *forem sendo formados, vamos anexando-os ao conjunto* P, *ou seja, substituindo* P *pela união* $P \cup \{[A[i], B[j]]\}$.

```
> A := {a, b, c}:  B := {1, 2, 3}:
> P := {}:   # conjunto vazio
>
> for i from 1 to nops(A) do
>   for j from 1 to nops(B) do
>     elemento := [A[i], B[j]];
>     P := P union {elemento}
>   end do
> end do;
>
> P;
```

$$\{[a, 1], [a, 2], [a, 3], [b, 1], [b, 2], [b, 3], [c, 1], [c, 2], [c, 3]\}$$

Exemplo 8.13 *Dada uma matriz M, queremos calcular a soma e o produto de todos os seus elementos. Como os elementos de uma matriz possuem dois índices, precisamos de dois* `for` *encaixados para percorrer toda a matriz. Inicialmente, atribuímos o valor inicial da soma S dos elementos da matriz como sendo 0 e o produto P como sendo 1. À medida que matriz for sendo percorrida, adicionamos cada elemento $M[i, j]$ a S e também multiplicamos P por $M[i, j]$.*

Por exemplo, como S vale 0 inicialmente, quando pegarmos o primeiro elemento da matriz, o $M_{11} = M[1, 1]$ e somarmos a S, S passará a valer $0 + M_{11} = M_{11}$. Como P vale inicialmente 1, quando esse valor for multiplicado por M_{11}, P passará a valer $1 \cdot M_{11} = M_{11}$. Quando pegarmos o segundo elemento $M_{12} = M[1, 2]$ e somarmos a S passará a valer $M_{11} + M_{12}$ e quando multiplicarmos esse segundo elemento por P, P passará a valer $M_{11} \cdot M_{12}$. No final, S valerá a soma de todos os elementos da matriz e P valerá o produto deles.

Optamos por gerar a matriz M aleatoriamente e com ordem 4×7.

```
> S := 0: P := 1:
> with(LinearAlgebra): M := RandomMatrix(4, 7);
```

$$M := \begin{bmatrix} 89 & -52 & 72 & 55 & -54 & -11 & 53 \\ -70 & 37 & 22 & -12 & -88 & 35 & 61 \\ -44 & 58 & -34 & 98 & -90 & 61 & -70 \\ -48 & -45 & 69 & 56 & -66 & 96 & 22 \end{bmatrix}$$

```
> for i from 1 to 4 do
>   for j from 1 to 7 do
>     S := S + M[i, j];
>     P := P*M[i, j];
>   end do:
> end do:
> S;    # mostra a soma dos elementos de M
```

$$200$$

```
> P;    # mostra o produto dos elementos de M
```

$$-2817862897571087794562066418372263049953280000000$$

Além do `for-from` visto anteriormente, uma outra forma do `for` é o `for-in`:

`for` *variável* `in` *lista* `do` *comando* `end do;`

Nessa forma, a *variável* percorre toda a lista e, para cada valor assumido, é executado o *comando* (ou grupo de comandos). No lugar de lista podemos ter também um conjunto ou uma seqüência.

Exemplo 8.14 *Percorrer com uma variável n uma lista de inteiros, e, para cada valor que n assuma, calcular n^2.*

```
> lista := [-3, 0, 1, 7, 11, 97]:
> for n in lista do print(n^2) end do;
```

$$9$$

$$0$$

$$1$$

49

121

9409

Outra forma de criar uma estrutura de repetição é usando um comando

`while` *condição* `do` *comando* `end do;`

que funciona da seguinte maneira: a *condição* é avaliada e, se ela for verdadeira, o *comando* (ou grupo de comandos) escrito após o `do` e antes do `end do` é executado. Se ela for falsa, o *comando* será ignorado. Sempre que o comando for executado, a condição será avaliada e, enquanto ela for verdadeira, o comando será executado novamente. O laço avalia condição, executa comando, avalia condição, ... se repetirá até que a condição seja falsa.

O *comando* deve ter instruções que possam em algum momento modificar o valor da *condição*, senão o laço `while` se repetirá indefinidamente.

A vantagem em usar o `while` é que não nos preocupamos em saber quantas vezes o comando será repetido. No `for` é diferente: devemos especificar *a priori* o valor inicial e o valor final de determinadas variáveis e isso equivale a saber quantas vezes o comando será repetido.

Exemplo 8.15 *Neste exemplo vamos calcular mais um somatório. Mas, desta vez será diferente: não vamos especificar quantos termos devemos somar. Vamos fornecer a condição que deve ser observada para continuar somando termos. Dados uma função* $f(n) = \dfrac{\ln(n+1)}{n^3+1}$ *(termo geral do somatório) e um valor real* $\epsilon = 10^{-8}$*, queremos somar todos os termos* $f(n)$*,* $n = 1, 2, 3, \cdots$ *cujo módulos sejam maiores do que* ϵ*. Como não sabemos quantos termos devemos somar, temos um caso típico de uso do* `while`*. Como no cálculo de todo somatório, iniciamos atribuindo um valor 0 a uma variável auxiliar Soma.*

```
> f := n -> ln(n + 1.0)/(n^3 + 1);
```

$$f := n \to \frac{\ln(n+1.0)}{n^3+1}$$

```
> Soma := 0:   # valor inicial da Soma
> n := 1:      # valor inicial de n
> while abs(f(n)) > 10^(-8) do
>   Soma := Soma + f(n);
>   n := n + 1;
> end do:
> n, Soma;    # mostra os valores finais de n e de Soma
```

879, .5951569400

O valor $0,5951569400$ *mostrado corresponde à soma procurada.*

Agora, quantos termos foram somados? 879 termos? Não. O n = 879 cujo valor foi mostrado no final corresponde ao primeiro termo da série que não satisfaz a condição do `while`*, ou seja, o f(879) foi o primeiro termo deixado fora do cálculo da soma. Portanto, foram somados 878 termos.*

Tanto o for quanto o while são casos particulares de um tipo de laço mais geral cuja sintaxe é:

`for` *variável* `from` *valor1* `to` *valor2* `by` *incremento*
`while` *condição* `do` *comando*
`end do;`

Pode-se deixar de escrever várias partes desse comando. O único trecho que não pode ser omitido é o `do` e o `end do`. Se os trechos `from` *valor1*, `to` *valor2*, `by` *incremento* e `while` *condição* forem omitidos, serão assumidos `from 1`, `to infinity`, `by 1` e `while true` nos lugares deles, respectivamente.

Além disso, a ordem em que esses trechos aparecem no for pode ser alterada à vontade. Por exemplo, usar um `for i from 1 to 10 by 2 ...` é o mesmo que usar `for i from 1 by 2 to 10 ...`.

Exemplo 8.16 *Calcular a soma* $17 + 22 + 27 + \cdots$ *de todos os inteiros da forma* $17 + 5n$, $n \geq 0$, *que são menores do que 1000.*

```
> total := 0:   # valor inicial da soma
> for i from 17 by 5 while i < 1000 do
>   total := total + i;
> end do:
> total;        # mostra a soma total obtida
```

$$378902$$

Exemplo 8.17 *Listar todos os* n *inteiros primos que satisfaçam* $17 \leq n < 2000$.

```
> for i from 17 by 2
>    while i < 2000 do
>      if isprime(i) then print(i) end if
>    end do;
```

$$17$$
$$19$$
$$23$$
$$\ldots$$
$$1999$$

Exemplo 8.18 *Determinar o menor primo maior do que* 10^9 *(1 bilhão). A idéia é iniciar com* $j = 10^9$ *e, enquanto* j *não for primo, ir aumentando seu valor de 1 em 1.*

```
> for j from 10^9 while not isprime(j) do end do;
> j;
```

$$1000000007$$

Toda estrutura de repetição admite várias maneiras equivalentes de ser codificada. Neste caso, se quiséssemos usar só a versão simplificada do `while` *teríamos a seguinte codificação:*

```
> j := 10^9:
> while not isprime(j) do j := j + 1 end do:
```

De modo semelhante, para determinar o maior primo menor do que 1 bilhão basta usar o seguinte `for-while`*:*

```
> for k from 10^9 by -1 while not isprime(k) do end do;
> k;
```

999999937

Exemplo 8.19 *O uso da forma completa do* `for` *pode simplificar a codificação de determinados trechos. Veja como fica um pouco mais simples o somatório calculado no Exemplo 8.15.*

```
> f := n -> ln(n + 1.0)/(n^3 + 1):   # definição de f(n)
> Soma := 0;        # valor inicial da Soma
> for n from 1 while abs(f(n)) > 10^(-8) do
>   Soma := Soma + f(n);
> end do:
```

O Maple possui os comandos `break` e `next` para serem usados em estruturas de repetição, sejam elas do tipo `for` ou `while`.

O `break` faz com que a repetição encerre (prematuramente). Quando um `break` é executado, o próximo comando será o que estiver depois do `end do`.

O `next` faz iniciar uma nova iteração, sem necessariamente sair da estrutura de repetição. Quando um `next` é executado, ele pega o próximo valor da variável e retorna à execução do início do laço.

Exemplo 8.20 *Neste exemplo a variável n é aumentada de 1 em 1 a partir de 1000. Estava programado inicialmente para o n chegar até* 2000*, mas um* `break` *encontrado "pelo caminho" faz com que o laço* `for` *seja encerrado bem antes disso. O* `break` *é encontrado na hora em que a parte fracionária de* $n/17$ *for 0, ou seja, quando n for um múltiplo de 17.*

```
> for n from 1000 to 2000 do
>   if frac(n/17) = 0 then break end if;
>   print(n);
> end do;
```

1000

1001

1002

Os valores mostrados levam-nos a concluir que 1003 é o menor múltiplo de 17 maior do que 1000.

Exemplo 8.21 *O laço* `for` *deste exemplo lista todos os inteiros de 1 a 100, exceto os múltiplos de 7.*

```
> for i from 1 to 100 do
>   if frac(i/7) = 0 then next end if;
>   print(i);
> end do;
```

No caso de vários `for` ou `while` encaixados, o `break` ou o `next` atua apenas no nível em que eles estiverem. Por exemplo,

```
> for i from ... do
>   from j from ...
>     break;
```

O `break` faz sair somente do `for j ...` de modo que o `for i ...` continua sendo executado.

8.3 Tipos pré-definidos

Em várias situações é muito importante saber quais são os tipos dos objetos envolvidos. Por exemplo, uma função $f(x)$ com x inteiro deve fornecer resposta inválida se o x for fracionário. Um procedimento para somar vetores u e v também não deverá funcionar se u e v forem conjuntos ou listas. Nesta seção apresentamos algumas funções (comandos) que podem ser usadas para detectar a tempo esse tipo de situação.

Além dos tipos `float` (real), `integer` (inteiro), ... o Maple possui mais de 160 tipos considerados "básicos". Uma listagem de uma pequena parte deles está mostrada a seguir. Para uma listagem completa consulte a documentação *on line* digitando `?type` .

TIPO BÁSICO *BREVE DESCRIÇÃO; EXEMPLO*

integer Inteiro; 0

posint Inteiro positivo; 3

negint Inteiro negativo; -3

odd Inteiro ímpar; 5

even Inteiro par; 4

float Número real (ponto flutuante); 3.14159

realcons Inteiro, racional, real, infinity ou -infinity; 2.71828

complex $a + b * I$, onde a e b são do tipo realcons; $5 + 4 * I$

fraction Fração p/q com p e q inteiros; 2/3

rational Fração ou inteiro; 5

numeric Numérico (inteiro, fração ou real); 4.5/3

positive Numérico positivo; 1.2345

negative Numérico negativo; -11

algebraic Algébrico; $sqrt(2) * I$

polynom Polinômio; xˆ2+5*x+3

list Lista; $[v, w, 1, 2, 3]$

listlist Lista de listas; $[[1, 2], [a, b]]$

set Conjunto; $\{-1, 0, x, y\}$

series Série; $series(\sin(x), x = 0, 4)$

exprseq Seqüência; m, n, p

anything Qualquer tipo, exceto seqüência

symbol Símbolo; alpha

‘+‘ Soma; $a + b$

‘*‘ Produto; $x * y$

‘ˆ‘ Potência; $2\hat{}x$

‘<‘ Desigualdade; $x < 0$

‘=‘ Igualdade; $y = 4$

name Nome de variável; x

function Função; $f(x)$

procedure Procedimento; f

Matrix Matriz; $Matrix([[1, 2], [3, 4]])$

string Caracteres entre aspas; “teste”

indexed Variável indexada; $a[n]$

Para mostrar o tipo de uma variável, constante ou expressão X podemos usar o comando `whattype(`X`)`.

Exemplo 8.22 *Usamos várias vezes o comando* `whattype` *para obter o tipo básico das expressões fornecidas como parâmetro desse comando.*

```
> whattype(cos(x));
```

$$function$$

```
> whattype(cos);
```

$$symbol$$

```
> whattype([a, b, c]);
```

$$list$$

```
> whattype(x > 1);
```

$$<$$

```
> whattype(a[i, j]);
```

$$indexed$$

```
> whattype(-4);
```

$$integer$$

```
> whattype(-4.0);
```

$$float$$

```
> whattype(-4*I);
```

$$complex$$

```
> whattype(x = 1);
```

$$=$$

Exemplo 8.23 *Se uma expressão contiver várias operações, então o tipo da expressão é o da operação que tiver menor prioridade no cálculo. Por exemplo, uma soma de potências é do tipo soma e não do tipo potência.*

```
> whattype(a + 4);
```

$$+$$

```
> whattype(x*f(x));
```

$$*$$

```
> whattype(2^(x + 1));
```

$$\hat{}$$

```
> whattype(2^x + 1);
```

$$+$$

```
> whattype(ln(2));
```

$$function$$

```
> whattype(3 + ln(2));
```

$$+$$

O comando `type(`*expressão, tipo*`)` pode ser usado para checar se uma *expressão* é de determinado *tipo*. Dependendo do resultado do teste efetuado, o `type` retorna sempre *true* ou *false*, ou seja, verdadeiro ou falso.

Exemplo 8.24 *Usamos o* `type` *várias vezes para verificar se a expressão dada pertence ao tipo fornecido como segundo parâmetro.*

```
> type( {1, 2, 3}, list);
```

$$false$$

```
> type( {1, 2, 3}, Matrix);
```

$$false$$

```
> type( {1, 2, 3}, set);
```

$$true$$

```
> type(4, integer);
```

$$true$$

```
> type(4, positive);
```

$$true$$

```
> type(4, numeric);
```

$$true$$

```
> type(4, float);
```

$$false$$

```
> type(x, name);
```

$$true$$

```
> type(x, string);
```

$$false$$

```
> type("x", string);
```

$$true$$

Podemos agrupar os tipos básicos e, a partir deles, construir novos tipos que são chamados *tipos compostos* ou *tipos estruturados*. Praticamente não há limite para a quantidade e variedade de tipos estruturados que podem ser criados. Vamos mencionar aqui apenas alguns poucos:

- Um novo tipo formado pela união de outros tipos pode ser definido colocando-se os nomes dos tipos entre chaves e separados por vírgulas. Por exemplo, {`T1, T2`} é um tipo composto que corresponde à união dos tipos `T1` e `T2`.
- O tipo "*lista de elementos do tipo T*" pode ser definido na forma `list(T)`.
- O tipo "*conjunto de elementos do tipo T*" pode ser definido na forma `set(T)`.
- O resultado da operação $var1\hat{}var2$ onde $var1, var2$ são objetos dos tipos T1 e T2, respectivamente, pode ser considerado como sendo do tipo T1^T2. No lugar de "^" (potenciação) poderíamos ter outras operações como "&+" (adição) ou "&*" (multiplicação).

Exemplo 8.25 *Definimos um tipo* `POTENCIA` *que corresponde a uma potência na qual a base pode ser do tipo nome de variável (*`name`*) e o expoente pode ser do tipo inteiro, real ou do tipo nome de variável. Depois, testamos se algumas expressões pertencem ao novo tipo assim definido.*

```
> POTENCIA := name^{integer, float, name};
```

$$POTENCIA := name^{\{float,\, integer,\, name\}}$$

```
> type(x^4, POTENCIA);
```

$$true$$

```
> type(x^x, POTENCIA);
```

$$true$$

```
> type(y^2.0, POTENCIA);
```

$$true$$

```
> type(2^x, POTENCIA);
```

$$false$$

Exemplo 8.26 *Definimos um tipo* `listaR` *como sendo uma "lista de reais" e verificamos se alguns elementos pertencem ao novo tipo assim definido.*

```
> listaR := list(float);    # definição do novo tipo
```

$$listaR := list(float)$$

```
> type([3.0], listaR);
```

$$true$$

```
> type([3.0, 5.0, 1.0, 7.0], listaR);
```

$$true$$

```
> type([3, 5, 1, 7], listaR);
```

$$false$$

Exemplo 8.27 *O tipo de uma variável pode mudar, dependendo das atribuições que lhe sejam feitas, conforme demonstramos neste exemplo.*

```
> whattype(x);
```

$$symbol$$

```
> x := 3:  whattype(x);
```

$$integer$$

```
> x := 2.718: whattype(x);
```

$$float$$

8.4 Procedimentos

A elaboração de procedimentos (*procedures*) constitui a parte mais importante da programação com o Maple. Podemos até dizer que "programar com o Maple" é o mesmo que "elaborar procedimentos".

Suponhamos que existam alguns comandos, digamos *comando1*, *comando2* e *comando3* que devam ser executados pelo Maple. Que tal dar um nome a esse conjunto de comandos e executá-los apenas usando o nome dado? É muito fácil fazer isso: basta "envolver" os comandos a serem executados pelas palavras `proc()` e `end proc` e atribuí-los a um nome de variável. Esse nome de variável será o nome do procedimento e será usado como um novo comando para chamar à execução os comandos definidos no procedimento.

Digamos que o nome escolhido para o procedimento seja *Inicio*. Então ele pode ser definido assim:

```
> Inicio := proc() comando1; comando2; comando3; end proc;
```

Depois do `end proc;` o Maple responde:

Inicio := **proc***() comando1; comando2; comando3;* **end proc;**

Assim, em vez de digitar cada um dos comandos *comando1*, *comando2* e *comando3* eles podem ser executados digitando-se apenas o nome do procedimento:

```
> Inicio();
```

Para caracterizar uma chamada a um procedimento é preciso que tenha parênteses acompanhando o nome *Inicio*, mesmo que estejam vazios.

Fica mais fácil ler e compreender um procedimento cuja definição se espalhe por várias linhas:

```
> Inicio := proc()
>   comando1;
>   comando2;
>   comando3;
> end proc:
```

Na digitação de um procedimento em várias linhas, a partir da segunda linha aparece a mensagem

Warning, premature end of input

Essa mensagem desaparece automaticamente depois da digitação do `end proc`.

Com dois pontos depois do `end proc`, o Maple não ecoa a definição do procedimento.

É comum um procedimento usar internamente valores de variáveis que lhe sejam fornecidos "de fora", ou seja, fornecidos pelo usuário ou por outros comandos que tenham chamado o procedimento à execução. Para isso, basta colocar a relação de variáveis *var1, var2, ...* nos parênteses iniciais, junto à palavra `proc`:

Nome_do_procedimento := `proc(`*var1, var2, ...*`)`
  comando1;
  comando2;
  ...
  `return` *valor*;
`end proc;`

Com essa definição, o procedimento deve ser chamado através do seu nome seguido de uma relação de valores *valor1, valor2, ...* entre parênteses:

Nome_do_procedimento`(`*valor1, valor2, ...*`)`.

O *valor1* será atribuído a *var1*, o *valor2* será atribuído a *var2*, etc. Os comandos do procedimento serão executados usando esses valores fornecidos.

É comum também um procedimento retornar valores "para fora", ou seja, retornar para o usuário ou para os comandos que o tiverem chamado. Isso pode ser feito com um comando `return` *valor*; . O `return` é opcional e, se ele não for especificado, o procedimento retornará o último valor que ele tiver calculado durante sua execução.

Exemplo 8.28 *A definição de um procedimento parece muito com a de uma função de uma ou várias variáveis. Às vezes um procedimento pode até ser usado como se fosse uma função.*

Vamos definir um procedimento `F` *que receba um valor x e devolva $y = x^2$:*

```
> F := proc(x)
>   y := x^2;
>   return y;
> end proc:
```

O procedimento assim definido se comporta como se fosse a função $F(x) = x^2$. Pode-se calcular valores em pontos do domínio, calcular derivada, integral, fazer gráfico, etc.

Exemplo 8.29 *Vamos definir um procedimento que receba uma função definida por uma expressão algébrica y, a variável x da função e as extremidades a e b de um intervalo, calcule a derivada da função dada e construa os gráficos da função e sua derivada no intervalo $[a, b]$ em um mesmo sistema de eixos.*

Chamando o procedimento de `ConstroiGraficos`*, ele deve ser definido com quatro parâmetros y, x, a e b.*

```
> ConstroiGraficos := proc(y, x, a, b)
>    z := diff(y, x);        # calcula a derivada z(x) = y'(x)
>    plot([y, z], x=a..b);   # constrói os gráficos de y(x) e z(x)
> end proc:
```

Depois de definido, podemos usá-lo para construir os gráficos de $y = x^2$ e sua derivada no intervalo $[-2, 2]$, bastando digitar:

```
> ConstroiGraficos(x^2, x, -2, 2);
```

Apesar do nome da variável definida no procedimento ter sido x, ele pode ser usado com funções com outro nome de variável. Para isso, basta informar o nome da variável da função como segundo parâmetro na chamada do procedimento. Aqui, usamos o mesmo procedimento para construir os gráficos de $f(t) = \operatorname{sen} t + \cos(2t)$ e de $f'(t)$ com $t \in [-\pi, \pi]$.

```
> ConstroiGraficos(sin(t) + cos(2*t), t, -Pi, Pi);
```

Exemplo 8.30 *Na definição de um procedimento, devemos ter muito cuidado com a redefinição dos valores dos parâmetros. O procedimento* `Teste` *definido neste exemplo tenta reduzir o valor do parâmetro x à metade, mas isso gera erro na sua execução.*

```
> Teste := proc(x)
>   x := x/2;
> end proc:
>
> Teste(4);
```

Error, (in Teste) illegal use of a formal parameter

Dentro do procedimento `Teste` *é como se tivesse sido feito a atribuição $4 := 4/2$, o que está errado.*

É altamente recomendável que um procedimento tenha cuidado com os valores que lhe são passados. Um procedimento definido para efetuar cálculos com números inteiros pode gerar resultados desastrosos se forem passados para ele números fracionários, por exemplo. Essa checagem pode ser feita facilmente, bastando para isso escrever cada parâmetro X em `proc` na forma X :: *tipo*.

Os procedimentos definidos nos Exemplos 8.28 e 8.29 assumem que um usuário saiba chamá-los de forma correta, porque eles não fazem nenhuma checagem nos tipos dos parâmetros fornecidos. Uma maneira mais eficiente de defini-los seria:

`F := proc(x::float)` ..., para exigir que o x fornecido seja do tipo `float`.

`ConstroiGraficos := proc(y::algebraic, x::name, a::float, b::float)` ..., para exigir que y seja do tipo expressão algébrica (algebraic), x seja do tipo nome de variável (name) e a e b sejam reais (float).

Exemplo 8.31 *O seguinte procedimento P, apesar de elaborado para trabalhar somente com inteiros positivos x, não faz nenhum tipo de checagem do tipo do parâmetro passado.*

```
> P := proc(x)
>   if isprime(x) then
>     print(‘É primo‘)
>   else
>     print(‘Não é primo‘);
>   end if;
> end proc:
```

Ele funciona bem com inteiros positivos:

```
> P(4);
```

Não é primo

```
> P(11);
```

É primo

```
> P(19999999);
```

É primo

Mas, veja o que acontece quando ele é chamado com um parâmetro fracionário. Ele repassa o valor fracionário para a função `isprime`*, que é quem detecta que algo está indo errado.*

```
> P(3.4);
```

Error, (in isprime) argument must be an integer

Para melhorar significativamente a definição desse procedimento, basta trocar o "x" do início por um "x::posint". Com isso, valores que não sejam inteiros positivos serão rejeitados logo na "porta de entrada" do procedimento. Dessa forma, o procedimento não vai perder tempo chamando a função `isprime` *para atuar com valores fracionários.*

```
> P := proc(x::posint)
>   if isprime(x) then
>     print(‘É primo‘)
>   else
>     print(‘Não é primo‘);
>   end if;
> end proc:
```

```
> P(3.4);
```

Error, invalid input: P expects its 1st argument, x, to be of type posint, but received 3.4

Outra opção, mais trabalhosa, é fazer uma checagem de tipos dentro do próprio procedimento com um comando `if` *usando* `type` *ou* `whattype`*:*

```
> P := proc(x)
>   if not type(x, posint) then
>     print(‘ERRO: o parâmetro deve ser inteiro positivo‘)
>   else
>     if isprime(x) then
>       print(‘É primo‘)
>     else
>       print(‘Não é primo‘);
>     end if;
>   end if;
> end proc:
```

```
> P(3.4);
```

ERRO: o parâmetro deve ser inteiro positivo

Exemplo 8.32 *Neste exemplo definimos um procedimento* `Calcula` *que deve receber como parâmetros uma expressão algébrica F, um nome de variável x e um ponto de seu domínio a, transformar a expressão algébrica em função f, calcular sua derivada usando o operador diferencial e retornar o valor $f(a) + f'(a)$.*

```
> Calcula := proc(F::algebraic, x::name, a::numeric)
>   f := unapply(F, x);
>   f(a) + D(f)(a);     #  ou  return f(a)+D(f)(a);
> end proc:
```

Exemplificando o uso desse procedimento em vários casos:

```
> Calcula(x^2, x, 1);
```

$$3$$

```
> Calcula(sin(t), t, 3));
```

$$\sin(3) + \cos(3)$$

```
> Calcula(alpha^3 - 3*alpha, alpha, 0);
```

$$-3$$

8.5 Variáveis locais e globais

Em geral, toda linguagem de programação tem dois tipos de variáveis: as locais e as globais. Como o nome já diz tudo, a diferença é só na região do programa onde elas são conhecidas. As globais são conhecidas em toda parte e as locais são restritas aos procedimentos ou às funções onde elas estiverem definidas.

No Maple, uma variável x é definida como sendo local, se for declarada na forma `local` x; no início do procedimento. Analogamente, x será global se for definida na forma `global` x. Se não for especificado nada a respeito de uma variável de um procedimento, o Maple tentará defini-la como sendo local.

Exemplo 8.33 *Definimos dois procedimentos* `CalcX` *e* `CalcX2` *que definem um valor para x. No* `CalcX` *o x é global, enquanto que no* `CalcX2` *ele é local.*

```
> CalcX := proc()
>   global x;
>   x := cos(Pi/10);
> end proc:
>
> CalcX2 := proc()
>   local x;
>   x := ln(2);
> end proc:
```

Agora vamos usá-los para ver o que acontece. Inicialmente, chamamos `CalcX` *e depois queremos ver o valor de x que retorna desse procedimento.*

```
> CalcX():
> x;
```

$$\cos\left(\frac{1}{10}\pi\right)$$

Veja o que acontece com `CalcX2`. *Internamente, ele altera o valor de* x *para* $\ln(2)$, *mas como o* x *dele é local isso não afeta o valor que havia sido definido antes dele ser chamado.*

```
> CalcX2():
> x;
```

$$\cos\left(\frac{1}{10}\pi\right)$$

Exemplo 8.34 *Dados um inteiro* n, *uma função* $f(x)$ *e um intervalo aberto* $I =]a, b[$, *entre os* n *primeiros termos da seqüência* $(f(1), f(2), f(3), \cdots)$, *queremos saber quantos pertencem ao intervalo* I. *Para isso, usamos um* `for` *para percorrer a seqüência e um* `if` *para testar se cada termo percorrido satisfaz as condições desejadas, ou seja, se* $f(i) \in]a, b[$. *Se pertencer, incrementamos de uma unidade uma variável auxiliar* `cont` *que foi iniciada em 0. A função dessa variável auxiliar é contar quantos termos satisfazem a propriedade desejada. Além disso, usamos um comando* `print` *para mostrar a posição e o valor do termo que satisfaz a propriedade desejada. Para codificar isso, definimos um procedimento de nome* `ContaTermos` *que recebe 4 parâmetros: a função* f, *o inteiro* n *e as extremidades* a *e* b *do intervalo.*

```
> ContaTermos := proc(f::procedure, n::posint, a::numeric, b::numeric)
>   local i, cont;
>   cont := 0;
>   for i from 1 to n do
>     if (a < evalf(f(i)) and evalf(f(i)) < b) then
>       cont := cont + 1;
>       print(i, evalf(f(i)));
>     end if;
>   end do;
>   return cont;
> end proc:
```

Para exemplificar, vamos determinar entre os 100 primeiros termos da seqüência $(a_n) = (n\cos n - n^2 \operatorname{sen}(2n))$, *quais os que satisfazem* $-5 < a_n < 5$.

```
> f := x -> x*cos(x) - x^2*sin(2*x):
> ContaTermos(f, 100, -5, 5);
```

$$1,\ -.3689951209$$

$$2,\ 2.194916308$$

$$3,\ -.455238006$$

$$11,\ 1.119691102$$

$$4$$

Portanto, temos que um total de 4 termos da seqüência dada pertencem ao intervalo aberto $]-5, 5[$: *o primeiro, o segundo, o terceiro e o décimo primeiro termos.*

Exemplo 8.35 *Vamos aperfeiçoar o que foi feito no Exemplo 8.11. Dada uma função* f *e dois inteiros positivos* $i1$ *e* $i2$, *determinar um procedimento que calcule* $\sum_{n=i1}^{i2} f(n)$.

```
> Soma := proc(f::procedure, i1::posint, i2::posint)
>
>   local s, n;
>
>   s := 0;
>   for n from i1 to i2 do
>     s := s + f(n);
>   end do;
>   return s;
> end proc;

> f := x -> 1/x^2;
```

$$f := x \to \frac{1}{x^2}$$

```
> Soma(f, 1, 10);
```

$$\frac{1968329}{1270080}$$

O comando Soma(f, a, b) aqui definido é semelhante ao comando padrão `sum(`*f(n), n=a..b*`)`*.*

Exemplo 8.36 *Vamos refazer o Exemplo 8.12. Dados dois conjuntos A e B, definir um procedimento Cartesiano(A, B) que retorne o produto cartesiano $A \times B$.*

```
> Cartesiano := proc(A::set, B::set)
>   local i, j, P, elemento;
>
>   P := {};
>   for i from 1 to nops(A) do
>     for j from 1 to nops(B) do
>       elemento := [A[i], B[j]];
>       P := P union {elemento}
>     end do
>   end do;
>
>   return P;
> end proc;
```

Testando o procedimento definido:

```
> Cartesiano({x, y, z}, {alpha, beta, epsilon});
```

$$\{[x,\epsilon],[x,\alpha],[x,\beta],[y,\epsilon],[y,\alpha],[y,\beta],[z,\epsilon],[z,\alpha],[z,\beta]\}$$

No início de um procedimento, além das declarações de variáveis locais ou globais, podemos ter também algumas opções e uma descrição. Vamos citar aqui apenas a opção de "direitos autorais". Se essa opção estiver presente na forma `option 'Copyright ...'` então o Maple não mostra a definição do procedimento (a não ser que se exija que ele mostre, alterando-se o valor da variável `verboseproc` – ver seção B.4). A descrição do procedimento pode aparecer na forma `description '...'`, como no exemplo a seguir.

Exemplo 8.37 *Dados uma função f(x) contínua e um intervalo $[a, b]$ no qual a função muda de sinal (isto é, $f(a)f(b) < 0$) definimos um procedimento que calcula uma raiz aproximada de $f(x)$. O algoritmo utilizado é simples: é calculado o ponto médio $M = (A + B)/2$ do intervalo $[A, B]$, onde $A = a$ e $B = b$. Se $f(A)f(M) < 0$ então a raiz está no intervalo $[A, M]$ e redefinimos $B = M$; senão, a raiz está em $[M, B]$ e redefinimos $A = M$. Verificamos se o intervalo $[A, B]$ assim redefinido ficou com comprimento menor do que um ϵ suficientemente pequeno. Se ficou, então o ponto médio desse intervalo pode ser considerado uma raiz aproximada de f. Se o comprimento do intervalo ainda não está suficientemente pequeno, então calculamos o ponto médio dele e redefinimos A ou B de maneira semelhante ao que foi feito anteriormente (observando se $f(A)f(M) < 0$).*

```
> CalculaRaiz := proc(f::procedure, a::float, b::float)
>
>   option ‘Copyright (C) 2003 by Lenimar N. Andrade‘;
>   description ‘Calcula uma raiz de f(x) em [a, b]‘;
>
>   local epsilon, A, B, M;
>   epsilon := 0.000000001;
>   A := a; B := b;
>
>   if f(a)*f(b) >= 0 then
>      print(‘Erro: deveríamos ter f(a)f(b) < 0‘);
>      return infinity;
>   end if;
>
>   while abs(A - B) >= epsilon do
>     M := (A + B)/2.0;
>     if abs(f(M)) < epsilon then
>        return M;
>      else
>        if f(A)*f(M) < 0 then
>          B := M
>        else
>          A := M;
>       end if
>     end if
>   end do;
> end proc;
```

CalculaRaiz := **proc***(f::procedure, a::float, b::float) description ‘Calcula uma raiz de f(x) em [a, b]‘ . . .* **end proc**

Escolhemos o valor de retorno `infinity` *para sinalizar que houve algum tipo de erro na execução.*

Vamos usar o procedimento `CalculaRaiz` *para determinar as raízes de $2^x = x^2$.*

```
> f := x -> 2^x - x^2;
```

$$f := x \to 2^x - x^2$$

```
> CalculaRaiz(f, 3.5, 6.0);
```

$$3.999999999$$

```
> CalculaRaiz(f, 0.0, 3.0);
```

$$2.000000000$$

```
> CalculaRaiz(f, -1.0, 0.0);
```

$$-.7666646961$$

```
> CalculaRaiz(f, -2.0, -1.0);
```

$$\textit{Erro: deveríamos ter } f(a)f(b) < 0$$

$$\infty$$

Exemplo 8.38 *Definimos um procedimento* `Pertence` *booleano que verifica se um elemento x do tipo* `anything` *pertence a um objeto L que pode ser uma lista ou um conjunto. Além disso, calculamos sua posição na lista que é retornada como terceiro parâmetro do procedimento. Se x não pertencer a L então a posição retornada é 0.*

```
> restart;
> Pertence := proc(x::anything, L::{list, set}, posicao::name)
>   description 'Verifica se x pertence a L';
>   local k;
>   for k from 1 to nops(L) do
>     if x = op(k, L) then
>       posicao := k;
>       return true;
>     end if;
>   end do;
>   posicao := 0;
>   return false;
> end proc:
```

Para testar o procedimento assim definido, consideremos a lista definida a seguir. Vamos verificar se os elementos -11, 20 e 2002 pertencem a esta lista e calcular suas posições.

```
> lista := [x, y, z, 1, 2, -2.6, 0, -11, 2002, alpha];
```

$$lista := [x, y, z, 1, 2, -2.6, 0, -11, 2002, \alpha]$$

```
> Pertence(-11, lista, k);
```

$$true$$

```
> k;
```

$$8$$

```
> Pertence(20, lista, j);
```

$$false$$

```
> j;
```

$$0$$

```
> if Pertence(2002, lista, p) then
>   print(‘2002 aparece na lista na posição‘, p);
> else
>   print(‘2002 não aparece na lista‘);
> end if;
```

2002 aparece na lista na posição, 9

Utilizamos o nome do terceiro parâmetro como sendo k, j e p. Isso porque o procedimento exige um terceiro parâmetro do tipo `name` *e ao retornar do procedimento cada uma dessas variáveis retorna como sendo do tipo inteiro.*

Exemplo 8.39 *Definimos um tipo* `Float2` *que corresponde a um "procedimento com dois parâmetros reais" e verificamos se dois procedimentos* `f` *e* `g` *dados são desse tipo.*

```
> f := proc(x::float, y::float) return x + y; end proc:
> g := proc(i::integer, j::integer) return i*j; end proc:
> Float2 := procedure(float, float);
```

$$Float2 := procedure(float, float)$$

```
> type(f, Float2);
```

$$true$$

```
> type(g, Float2);
```

$$false$$

Exemplo 8.40 *Este exemplo deve ser visto com atenção. Inicialmente, observe o que acontece se for digitado no aviso (*prompt*) do Maple o seguinte:*

```
> a := b + 1:
> b := c + 1:
> a;
```

$$c + 2$$

Ao perguntar pelo valor de a, o Maple substituiu a por $b+1$ *e b por* $c+1$*, obtendo o valor de a como sendo* $(c+1)+1 = c+2$*, o que era de se esperar.*

Agora, veja o que acontece se os comandos forem agrupados em um procedimento com variáveis locais:

```
> F := proc()
>   local a, b, c;
>   a := b + 1;
>   b := c + 1;
>   return a;
> end proc;
```

Executando o procedimento F que retorna o valor de a depois das substituições feitas no procedimento:

```
> F();
```

$$b + 1$$

Note que obtivemos $b + 1$ como sendo o valor de a e não $c + 2$ que foi obtido sem usar o procedimento. Isso ocorreu porque a quantidade de substituições feitas em um procedimento não é a mesma quantidade de substituições do modo interativo (sem o uso de procedimentos). No modo interativo são feitas todas as substituições possíveis enquanto que em um procedimento, as variáveis são substituídas apenas uma vez, ou seja, a substituição é feita apenas no primeiro nível.

Exemplo 8.41 *A quantidade de parâmetros passados para um procedimento pode ser obtida através da variável* `nargs`*. A seqüência de parâmetros passados para um procedimento é guardada na variável* `args`*. Logo, o i-ésimo parâmetro é dado por* `args[i]`*.*

```
> f := proc(x, y, z)
>   local i;
>   printf("Procedimento com %d argumentos:", nargs);
>   for i from 1 to nargs do
>     print(args[i], whattype(args[i]));
>   end do;
> end proc:
```

O Maple possui uma função `printf` *semelhante à da linguagem C. Com ela, é possível fazer uma impressão formatada de um* string*, colocando o valor de* `nargs` *no local assinalado com um* `%d`*.*

Testando o procedimento f:

```
> f(1, 2.9, ln(c));
```

Procedimento com 3 argumentos:

$$1,\ integer$$

$$2.9,\ float$$

$$\ln(c),\ function$$

8.6 Procedimentos recursivos

Um procedimento pode fazer referência a si mesmo na sua definição. Essa propriedade é chamada *recursividade* do procedimento.

Um dos exemplos mais conhecidos de recusividade é a definição da função fatorial:

$$n! = \begin{cases} 1 & \text{se} \quad n = 0 \quad \text{ou} \quad n = 1 \\ n \cdot (n-1)! & \text{se} \quad n > 1 \end{cases}$$

Exemplo 8.42 *Vamos definir recursivamente a função fatorial de um inteiro positivo n.*

```
> Fatorial := proc(n::posint)
>   if n < 2 then
>     return 1;
>   else
>     return n*Fatorial(n - 1);
>   end if;
> end proc;
```

Fatorial := **proc***(n::posint) if* $n < 2$ *then return 1 else return n*Fatorial(n - 1) end if* **end proc**

Exemplificando o seu uso:

```
> Fatorial(7);
```

5040

```
> Fatorial(-3);
```

Error, invalid input: Fatorial expects its 1st argument, n, to be of type posint, but received -3

Exemplo 8.43 *Definimos agora outra função recursiva definida nos inteiros positivos:*

$$Fib(n) = \begin{cases} 1, & \text{se} \quad n = 1 \quad \text{ou} \quad n = 2 \\ Fib(n-1) + Fib(n-2), & \text{se} \quad n \geq 3 \end{cases}$$

```
> Fib := proc(n::posint)
>   if n <= 2 then
>     return 1;
>   else
>     return Fib(n - 1) + Fib(n - 2);
>   end if;
> end proc;
```

Fib := **proc***(n::posint) if n <= 2 then return 1 else return Fib(n - 1) + Fib(n - 2) end if* **end proc**

Depois de definida, podemos gerar alguns termos da seqüência $(a_n) = (Fib(n))$*:*

```
> seq(Fib(n), n = 1..20);
```

$$1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765$$

Apesar de fáceis de codificar, nem sempre as funções recursivas são eficientes. Quase sempre é possível definir uma função análoga de forma não recursiva. As versões não recursivas podem ser mais complicadas em termos de definição, mas sob o ponto de vista computacional são mais eficientes: não fazem cálculos desnecessários, gastam menos memória e menos tempo de computação.

Exemplo 8.44 *Definição da função fatorial de forma não recursiva.*

```
> FAT := proc(n::posint)
>   local k, prod;
>   prod := 1;
>   for k from 2 to n do
>     prod := prod*k;
>   end do;
>   return prod;
> end proc:
```

```
> FAT(7);
```

5040

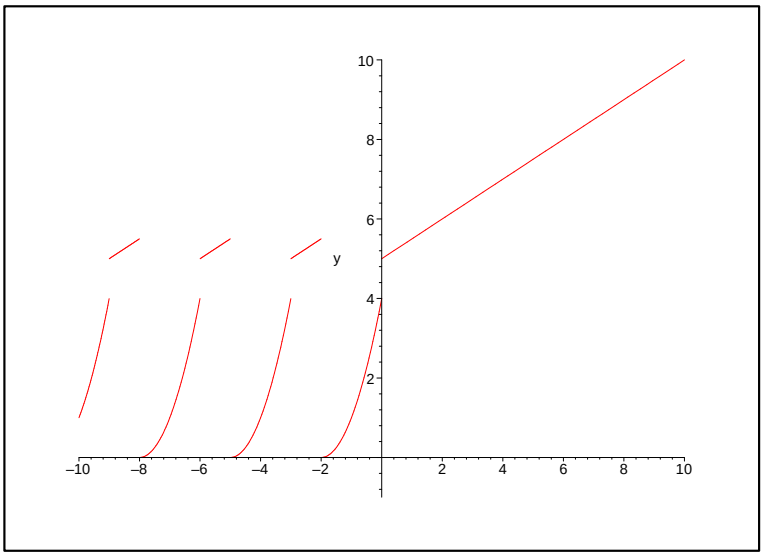
Exemplo 8.45 *Neste exemplo definimos duas funções F e G mutuamente recursivas: F dependendo de G e G dependendo de F nas suas definições.*

$$F(x) = \begin{cases} G(x+2) & \text{se} \quad x < 0 \\ x/2 + 5 & \text{se} \quad x \geq 0 \end{cases} \quad \text{e} \quad G(x) = \begin{cases} F(x+1) & \text{se} \quad x < 0 \\ x^2 & \text{se} \quad x \geq 0 \end{cases} .$$

Por exemplo, $G(-8) = F(-7) = G(-5) = F(-4) = G(-2) = F(-1) = G(1) = 1$.

```
> F := proc(x::numeric)
>   if (x < 0) then
>     return G(x + 2);
>   else
>     return (x/2 + 5);
>   end if;
> end proc:
>
> G := proc(x::numeric)
>   if (x < 0) then
>     return F(x + 1);
>   else
>     return (x*x);
>   end if;
> end proc:
```

Figura 8.1:



Depois de definidas, podemos fazer operações com essas funções, como por exemplo construir o gráfico de uma delas (Figura 8.1).

```
> plot(F, -10..10, y=-1..10, discont=true);
```

8.7 Exercícios

1) Dado um conjunto de n pontos $(x_1, y_1), \cdots, (x_n, y_n)$ do plano, com a aplicação do Método dos Mínimos Quadrados obtemos os valores $a = \dfrac{n\sum_{i=1}^{n} x_i y_i - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i}{n\sum_{i=1}^{n} x_i^2 - (\sum_{i=1}^{n} x_i)^2}$ e

$b = \dfrac{\sum_{i=1}^{n} x_i^2 \sum_{i=1}^{n} y_i - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} x_i y_i}{n \sum_{i=1}^{n} x_i^2 - (\sum_{i=1}^{n} x_i)^2}$ de modo que a reta cuja equação é $y = ax + b$ seja próxima dos pontos dados. Interpretando o conjunto de pontos como a lista de listas $[[x_1, y_1], \cdots, [x_n, y_n]]$, elabore um procedimento com três parâmetros L (do tipo `listlist`), $x_{\min}$ e $x_{\max}$ (do tipo `numeric`) que construa em um mesmo sistema de eixos o gráfico da reta $y = ax+b$, $x \in [x_{\min}, x_{\max}]$ e os pontos dados. (*Sugestão: os pontos podem ser desenhados com um comando* `plot` *com opções* `style=point`, `symbol=circle`).

2) Sabendo que a seqüência

$$x_{n+1} = \frac{1}{p}\left((p-1)x_n + \frac{a}{x_n^{p-1}}\right)$$

converge para $\sqrt[p]{a}$, onde $x_0 = 1$ e $a > 0$, elabore um procedimento em que dados a, p e $\epsilon > 0$ (como parâmetros) sejam mostrados todos os termos da seqüência que satisfazem a desigualdade $|x_{n+1} - x_n| < \epsilon$. Compare o valor do último x_{n+1} mostrado pelo procedimento com o que é obtido aplicando-se um `evalf` a `root`$[p](a)$.

3) Consideremos $x_0 \in \mathbb{R}$ como sendo uma aproximação para uma raiz da equação da forma $f(x) = 0$ onde $f(x)$ é derivável tal que $f'(x_0) \neq 0$. A resolução de uma equação dessa forma através do Método de Newton consiste em calcular vários termos da seqüência dada pela fórmula de recorrência

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}.$$

Dado $\epsilon > 0$, o cálculo dos termos da seqüência é efetuado até que `erro`$= |x_{n+1} - x_n|$ seja menor do que ϵ. Neste caso, dizemos que o último valor de x_{n+1} encontrado é a raiz aproximada da equação. Quanto menor o valor de ϵ, melhor será a aproximação.

Elabore um procedimento `Newton` com cabeçalho

```
Newton := proc(f::procedure, x0::numeric, epsilon::numeric)
```

que implemente o Método de Newton descrito anteriormente e que funcione conforme mostrado no exemplo a seguir em que encontramos uma raiz da equação $x^3+x^2-x-3 = 0$, partindo da aproximação inicial $x_0 = 2$ e usando $\epsilon = 0,0000001$.

```
> f := x -> x^3 + x^2 - x - 3;
```

$$f := x \to x^3 + x^2 - x - 3$$

```
> Newton(f, 2.0, 0.0000001);

 n = 1,  xn = 1.533333, erro = .466667
 n = 2,  xn = 1.377323, erro = .156010
 n = 3,  xn = 1.359527, erro = .017796
 n = 4,  xn = 1.359304, erro = .000223
 n = 5,  xn = 1.359304, erro = .000000

A raiz aproximada de f(x) = 0 e´ x = 1.359304

>
```

Apêndice A

Os pacotes do Maple

O Maple possui cerca de 320 funções ou comandos na biblioteca padrão, mais outros 1670 distribuídos por vários pacotes.

Listamos aqui todos os pacotes do Maple, versão 7, com uma breve descrição de cada um. Listamos também entre parênteses a quantidade de funções que cada pacote contém.

Para maiores informações sobre determinado pacote, use o *help on line* digitando um ponto de interrogação seguido do nome do pacote, por exemplo, `?algcurves`, `?CurveFitting`, etc. Isso mostra uma lista com todas as funções do pacote.

Para usar as funções de um pacote deve-se usar antes um comando `with(`*pacote*`)` ou usar o nome da função na forma *pacote*[*função*] (veja a seção 1.18).

algcurves Curvas algébricas (17)

codegen Geração de códigos para outras linguagens (26)

combinat Funções combinatórias (32)

combstruct Estruturas combinatórias (9)

context Ferramentas para construção de menus (9)

CurveFitting Curvas definidas por conjuntos de pontos (7)

DEtools Ferramentas para resolução de equações diferenciais (103)

diffalg Álgebra diferencial (26)

difforms Formas diferenciais (12)

Domains Criação de domínios de computação

ExternalCalling Chamadas a funções externas (2)

finance Matemática financeira (12)

GaussInt Inteiros gaussianos (29)

genfunc Funções geradoras racionais (12)

geom3d Geometria euclidiana tridimensional (164)

geometry Geometria euclidiana plana (112)

Groebner Bases de Groebner (22)

group Grupos de permutações (34)

inttrans Transformadas integrais (13)

liesymm Simetrias de Lie em sistemas de EDP (30)

linalg Álgebra Linear com estruturas de dados *array* (105)

LinearAlgebra Álgebra Linear com estruturas de dados *rtable* (103)

LinearFunctionalSystems Sistemas de equações com funcionais lineares (11)

LinearOperators Equações funcionais lineares (14)

ListTools Ferramentas para manipulação de listas (19)

LREtools Relações de recorrência lineares (18)

MathML Importação e exportação de dados no formato *MathML* (5)

Matlab Conexões com o Matlab (19)

networks Redes de grafos (75)

numapprox Aproximação numérica (14)

numtheory Teoria dos números (46)

Ore_algebra Álgebras de operadores lineares (18)

OrthogonalSeries Séries de polinômios ortogonais (16)

orthopoly Polinômios ortogonais (6)

padic Números p-ádicos (9)

PDEtools Resolução de EDPs (15)

plots Ferramentas para construção de gráficos (52)

plottools Objetos gráficos básicos (35)

PolynomialTools Ferramentas para manipulação com polinômios (10)

powseries Séries de potências (22)

process Multi processamento no Unix (8)

RandomTools Objetos gerados aleatoriamente (29)

RationalNormalForms Formas racionais normais (5)

RealDomain Domínio real (38)

simplex Otimização linear (17)

Sockets Ferramentas para comunicação em redes (21)

SolveTools Ferramentas para resolução de equações (5)

Spread Criação de planilhas (20)

stats Funções para Estatística (10)

StringTools Manipulação de strings (64)

student Funções úteis no ensino de Cálculo (32)

sumtools Cálculo de somatórios (10)

tensor Tensores e aplicações à Relatividade Geral (51)

Units Conversão de unidades (19)

XMLTools Ferramentas para documentos no formato XML (58)

Apêndice B

Outros itens

B.1 Salvando e recuperando expressões

Durante uma sessão de trabalho com o Maple, pode-se salvar em disco a sessão toda ou somente uma parte dela. Para salvar as expressões x, y, z, ... escolhidas, deve-se usar um comando

`save` *x, y, z, ..., "Nome_do_Arquivo"* .

Depois, essas expressões podem ser recuperadas com um comando

`read` *"Nome_do_Arquivo"*

Exemplo B.1 *Vamos definir três variáveis eq, S e R e salvar seus valores em um arquivo c:\teste.m.*

```
> eq := x^10 + 1000*x^3 - x + 1999;
```

$$eq := x^{10} + 1000x^3 - x + 1999$$

```
> S := fsolve(eq);
```

$$S := -2.638858805,\ -1.262126102$$

```
> R := convert(sin(Pi/40), radical);
```

$$R := \frac{1}{8}\sqrt{2-\sqrt{2}}\sqrt{2}\sqrt{5+\sqrt{5}} + \frac{1}{2}\sqrt{2+\sqrt{2}}\left(-\frac{1}{4}\sqrt{5}+\frac{1}{4}\right)$$

```
>  save eq, S, R, "c:\\teste.m";
```

Vamos agora "limpar" todas as variáveis da memória com um comando `restart`.

```
> restart;
> eq;
```

$$eq$$

```
> S;
```

$$S$$

```
> R;
```

$$R$$

E agora, vamos recuperar as variáveis que foram salvas em c:\teste.m:

```
> read "c:\\teste.m";
```

Conferindo os valores de cada variável:

```
> R;
```

$$\frac{1}{8}\sqrt{2-\sqrt{2}}\sqrt{2}\sqrt{5+\sqrt{5}}+\frac{1}{2}\sqrt{2+\sqrt{2}}\left(-\frac{1}{4}\sqrt{5}+\frac{1}{4}\right)$$

```
> S;
```

$$-2.638858805,\ -1.262126102$$

```
> eq;
```

$$x^{10}+1000x^3-x+1999$$

Pode-se também salvar dessa forma funções e procedimentos, basta fornecer seus nomes como parâmetros do `save`.

B.2 Gerando expressões nos formatos de outros programas

O Maple gera arquivos que sejam compatíveis com outros programas, como por exemplo arquivos no formato do LaTeX, arquivos que possam ser incorporados em programas em C ou em FORTRAN ou arquivos em HTML para serem anexados a páginas da Internet.

Para gerar uma listagem da expressão X nos formato do LaTeX, C ou FORTRAN, basta usar um comando `latex(`X`)`, `C(`X`)` ou `fortran(`X`)`, respectivamente.

O comando `latex` faz parte da biblioteca padrão, logo não precisa chamar nenhum pacote antes de utilizá-lo.

Exemplo B.2 *Dadas as expressões L e M, escrevê-las no formato do LaTeX.*

```
> L := Limit((x + 1)/sin(x + 1), x = -1);
```

$$L := \lim_{x \to -1} \frac{x+1}{\sin(x+1)}$$

```
> latex(L);

\lim _{x\rightarrow -1}{\frac {x+1}{\sin \left( x+1 \right) }}

>
> latex(L, "arquivo.tex");   # salva L em arquivo.tex
>
> M := Matrix([[1, 2], [3, 4]]);
```

$$M := \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

```
> latex(M);

 \left[ \begin {array}{cc} 1&2\\\noalign{\medskip}3&4\end {array}
 \right]
```

Os comandos `C` e `fortran` fazem parte do pacote `codegen`. Portanto, é recomendável usar um `with(codegen)` antes de usá-los.

Exemplo B.3 *Listar uma expressão dada no formato das linguagens de programação C e FORTRAN.*

```
> with(codegen):
> expr := x^4 + abs(cos(x + sqrt(x)))/(x^2 + 1);
```

$$expr := x^4 + \frac{|\cos(x+\sqrt{x})|}{x^2+1}$$

```
> C(expr);

   t0 = x*x*x*x+fabs(cos(x+sqrt(x)))/(x*x+1.0);

> fortran(expr);

   t0 = x**4+abs(cos(x+sqrt(x)))/(x**2+1)
```

B.3 Calculando o tempo gasto na execução

Para saber qual foi o tempo gasto na execução de determinado *comando* basta usar um `time(`*comando*`)`.

Exemplo B.4 *Calcular o tempo gasto na execução do seguinte comando* `fsolve`*:*

```
> time(fsolve(x^40 + 4*x^21 - 3*x^7 + 1 =0, x, complex));
```

.099

Para determinar o tempo gasto na execução de um grupo de comandos, pode-se criar uma variável auxiliar e atribuir a ela um comando `time()` para guardar o momento inicial da execução. Depois, no final da execução, chamar o `time()` novamente e fazer a diferença entre o momento inicial e o momento final da execução.

Exemplo B.5 *Neste exemplo calculamos o tempo gasto para contar e mostrar uma relação com todos os pares de primos da forma* $(i, i+2)$ *com* $1 \leq i \leq 10000$.

```
> inicio := time():
> quant := 0:
> for i from 1 to 10000 do
>   if isprime(i) and isprime(i + 2) then
>     print("( " || i || ", " || (i + 2) || ") são primos");
>     quant := quant + 1;
>   end if;
> end do;
> fim := time():
> TempoGasto := fim - inicio;
```

Outra opção para verificar o tempo de computação gasto é usar um `showtime()`. Neste caso, o aviso do Maple muda temporariamente para `O1 :=`, `O2 :=`, `O3 :=`, ... Para encerrar o `showtime` deve-se usar um comando `off`. O `showtime` mostra o tempo gasto e a quantidade de bytes utilizada na execução de cada comando.

Exemplo B.6 *Exemplificando o uso do* `showtime`*.*

```
> showtime();
O1 := int( 1/(x*(x^10 + 1)), x);
```

$$\ln(x) - \frac{1}{10}\ln(x^2+1) - \frac{1}{10}\ln(x^8 - x^6 + x^4 - x^2 + 1)$$

```
time = 0.35, bytes = 245042

O2 := series(arctan(x), x=0, 8);
```

$$x - \frac{1}{3}x^3 + \frac{1}{5}x^5 - \frac{1}{7}x^7 + O(x^8)$$

```
time = 0.03, bytes = 11934

O3 := off;
>
```

B.4 Listagem de funções dos pacotes do Maple

É possível listar a maior parte das funções pré-definidas do Maple. Para isso, basta ajustar a variável `verboseproc` para 2 ou 3 através do comando `interface`. Depois, listamos a função ou procedimento com um comando `print`, conforme fazemos nos exemplos a seguir.

Exemplo B.7 *Listar o procedimento* `group[Sylow]`*. Para isso, iniciamos ajustando o valor de* `verboseproc`*.*

```
> restart;
> interface(verboseproc=3);
```

Agora, usamos um comando `print` *para listar o procedimento desejado. Devido ao seu tamanho, apresentamos aqui somente uma pequena parte da listagem.*

```
> print(`group/Sylow`);

proc(pg, p)
local Cg, P, U, c, el, g, ij, l, lm, m, m1, m2, n, oCg, ord,
pel, q, sCg, u, ul, v1, U1, mh, m1h, m2h, mh1, m1h1, m2h1, cP
;
option `Copyright (c) 1990 by the University of Waterloo. A\
ll rights reserved.`;
    if not type(pg, function) or op(0, pg) <> 'permgroup'
    then error "wrong type of arguments"
    end if;
    ord := group['grouporder'](pg);
    if not isprime(p) or ord mod p <> 0 then error "the se\
        cond argument must be a prime divisor of the order\
         of the group"
    end if;
    n := op(1, pg);
    pel := `group/random`(pg, p, n);

    mh := ifactor(ord);
```

```
    if type(mh, `*`) then mh1 := {op(mh)}
    else mh1 := {mh}
    end if;

    ...     ...     ...    ...      ...

    return P
end proc
```

Exemplo B.8 *Listar o procedimento* `laplace` *do pacote* `inttrans`.

```
> interface(verboseproc=3);  # se ainda não tiver sido usado antes ...

> print(`inttrans/laplace`);

proc(expr::{algebraic, equation, set}, t::name, s::algebraic)
local opt, rats, soln, transforms, i;
option `Copyright (c) 1991 by the University of Waterloo. A\
ll rights reserved.`;
    userinfo(3, 'laplace', `Entering top-level laplace with `,
        expr, t, s);
    if nargs = 4 then opt := args[4] else opt := 'INT' end if
    ;
    if not type(s, freeof(t)) then error "Third parameter \
        %1 must be free of the second parameter %2", s, t
    end if;

    ...     ...     ...      ...      ...

    soln := subs(`laplace/internal` = 'laplace',
        `invlaplace/internal` = 'invlaplace', soln);
    procname(args[1 .. nargs]) := soln;
    soln
end proc
```

Devido à sua extensão, a listagem apresentada mostra apenas o início e o final do procedimento.

Referências Bibliográficas

[1] Bauldry, W. C., Evans, B., Johnson, J. (1995) *Linear Algebra with Maple*, John Wiley & Sons, Inc., New York.

[2] Cláudio, D. M., Marins, J. M. (1994) *Cálculo Numérico Computacional*, Editora Atlas.

[3] Corless, R. M. (2001) *Essential Maple 7*, University of Western Ontario, disponível em www.mapleapps.com.

[4] Dorofeev, G., Potapov, M., Rozov, N. (1973) *Elementary Mathematics – Selected Topics and Problem Solving*, Mir Publishers.

[5] Gray, T. W., Glynn, J. (1991) *Exploring Mathematics with Mathematica – Dialogs Concerning Computers and Mathematics*, Addison-Wesley Publishing Company.

[6] Harper, D., Wooff, C., Hodgkinson, D. (1991) *A Guide to Computer Algebra Systems*, John Wiley & Sons.

[7] Harris, K. (1992) *Discovering Calculus with Maple*, John Wiley & Sons.

[8] Heck, A. (1996) *Introduction to Maple*, Second Edition, Springer.

[9] Kostecki, W. (2000) *Matrices and Matrix Operations*, disponível em www.mapleapps.com.

[10] Kraft, R. (2002) *Programming in Maple*, Purdue University Calumet, disponível em www.mapleapps.com.

[11] Lidsky, V., Ovsyannikov, L., Tulaikov, A., Shabunin, M. (1973) *Problems in Elementary Mathematics*, Mir Publishers.

[12] Maron, I. A. (1973) *Problems in Calculus of One Variable*, Mir Publishers.

[13] Mathews, J. H., Howell, R. W. (2001) *Complex Analysis: Maple Worksheets*, disponível em www.mapleapps.com.

[14] Meade, D. B. (2001) *Ordinary Differential Equations Powertool*, University of South Carolina, disponível em www.math.sc.edu/∼meade/

[15] Mihailovs, A. (2002) *Abstract Algebra with Maple*, disponível em www.mapleapps.com.

[16] Portugal, R. (2002) *Introdução ao Maple*, Centro Brasileiro de Pesquisas Físicas, disponível em www.cbpf.br/∼portugal/

[17] Waterloo Maple Inc. (2001) *Maple 7 Learning Guide*.

[18] Waterloo Maple Inc. (2001) *Maple 7 Programming Guide*.

Índice Remissivo

Lenimar Nunes de Andrade nasceu em Patu, uma pequena cidade do alto sertão do Rio Grande do Norte. É Bacharel em Matemática pela Universidade Federal da Paraíba (1982), Mestre em Matemática pela Universidade Federal de Pernambuco (1987), Doutor em Engenharia Elétrica pela Universidade Estadual de Campinas (1998) e professor da Universidade Federal da Paraíba desde março de 1984. Desde 1988 vem se interessando por linguagens de programação de computadores.