Introdução ao Maple

Renato Portugal[1]
Coordenação de Ciência da Computação
Laboratório Nacional de Computação Científica
Av. Getúlio Vargas, 333
25651-070, Quitandinha, Petrópolis-RJ

[1]Portugal@Lncc.Br

Contents

1 Introdução **3**
- 1.1 Aspectos do Sistema Maple . . . 3
- 1.2 A *Worksheet* . . . 4
- 1.3 *Help on Line* . . . 5
- 1.4 Referências sobre Maple . . . 5

2 Noções Básicas **6**
- 2.1 Números . . . 6
- 2.2 Atribuição de um Nome a uma Expressão . . . 8
- 2.3 Avaliação Completa . . . 9
- 2.4 Tipos de Objetos . . . 10

3 Simplificação de Expressões **14**
- 3.1 Introdução . . . 14
- 3.2 *Expand* . . . 15
- 3.3 *Combine* . . . 16
- 3.4 *Convert* . . . 17
- 3.5 *Simplify* . . . 18

4 Álgebra Linear **20**
- 4.1 Introdução . . . 20
- 4.2 Definindo uma Matriz . . . 20
- 4.3 Matrizes Especiais . . . 23
- 4.4 Autovalores e Autovetores . . . 25
- 4.5 Manipulação Estrutural de Matrizes . . . 27
- 4.6 Cálculo Vetorial . . . 28

5 Gráficos **34**
- 5.1 Introdução . . . 34
- 5.2 Gráficos em 2 Dimensões . . . 36
  - 5.2.1 Introdução . . . 36
  - 5.2.2 Gráficos de Funções Parametrizadas . . . 39
  - 5.2.3 Gráficos em Coordenadas Polares . . . 40
  - 5.2.4 Gráficos de Funções Contínuas por Partes . . . 40
  - 5.2.5 Gráficos de Pontos Isolados . . . 41

5.2.6 Gráficos de Funções Definidas Implicitamente 44
5.3 Gráficos em 3 Dimensões 44
5.3.1 Introdução 44
5.3.2 Gráficos de Funções Parametrizadas 45
5.3.3 Gráficos em Coordenadas Esféricas 46
5.4 Exibindo vários Gráficos Simultaneamente 46
5.5 Animando Gráficos em duas ou três Dimensões 47
5.6 Colocando Textos em Gráficos 50
5.7 Imprimindo Gráficos 51
5.8 Manipulando Gráficos 52

6 Cálculo Diferencial e Integral 53
6.1 Funções Matemáticas 53
6.1.1 Definindo uma Função 53
6.1.2 Álgebra e Composição de Funções (@ e @@) 57
6.2 Integral 59
6.3 Séries 61
6.4 Séries de Fourier e transformadas 62

7 Equações diferenciais ordinárias 64
7.1 Introdução 64
7.2 Método de Classificação 66
7.2.1 EDO's de ordem 1 66
7.2.2 EDO's de ordem 2 ou maior 67
7.3 Pacote DEtools 73
7.3.1 Comandos para manipulação de EDO's 73
7.3.2 Comandos para visualização 74
7.3.3 Outros comandos que retornam soluções de EDO's 77
7.4 Método Numérico 77
7.5 Método de Séries 78

8 Equações diferenciais parciais 80
8.1 Introdução 80
8.2 Solução Geral e Solução Quase-geral 80
8.3 O Pacote PDEtools 82
8.3.1 Comando build 82
8.3.2 Comando dchange 82
8.3.3 Comando PDEplot 84
8.4 Limitações do Comando pdsolve 84

Referências Bibliográficas 85

Chapter 1

Introdução

1.1 Aspectos do Sistema Maple

O Maple é uma linguagem de computação que possui quatro aspectos gerais que são:

- Aspectos algébricos
- Aspectos numéricos
- Aspectos gráficos
- Aspectos de programação

Todos estes aspectos estão integrados formando um corpo único. Por exemplo, a partir de um resultado algébrico, uma análise numérica ou gráfica pode imediatamente ser feita. Em geral, na análise de um problema, várias ferramentas são necessárias. Se estas ferramentas não estiverem no mesmo *software*, um usuário enfrentará uma série de dificuldades para compatibilizar a saída de um *software* com a entrada de outro, além de ser obrigado a familiarizar-se com diferentes notações e estilos. É claro que o Maple não elimina completamente o uso de linguagens numéricas ou gráficas. Em aplicações mais elaboradas pode ser necessário usar recursos de linguagens como C ou Fortran. O Maple tem interface com estas linguagens no sentido de que um resultado algébrico encontrado no Maple pode ser convertido para a sintaxe da linguagem C ou Fortran 77.

Os aspectos novos trazidos pelo Maple juntamente com outros sistemas algébricos são a computação algébrica e a programação simbólica. A computação algébrica é uma área que teve um forte impulso nas décadas de 60 e 70, onde foram criados importantes algoritmos para integração analítica e fatoração de polinômios. Estes algoritmos estão baseados na Álgebra Moderna, que guia toda a implementação do núcleo de qualquer sistema algébrico.

O Maple é uma linguagem de programação simbólica. Os construtores deste sistema optaram em desenvolver um pequeno núcleo escrito na linguagem C gerenciando as operações que necessitam de maior velocidade de processamento, e a partir deste núcleo, desenvolveram uma nova linguagem. O próprio Maple foi escrito nesta nova linguagem. Mais do que 95% dos algoritmos estão escritos na linguagem Maple, estando acessíveis ao usuário. Esta opção dos seus arquitetos é muito saudável, pois uma linguagem que pode gerar todo um sistema algébrico do porte do Maple certamente é uma boa linguagem de programação.

Neste curso faremos uma introdução a alguns destes aspectos e analisaremos como eles se inter-relacionam. As versões utilizadas foram *Maple 6* e *7*.

1.2 A *Worksheet*

Nos microcomputadores com o Maple instalado, a *worksheet* é disparada clicando-se no ícone do programa. Em outros sistemas, ela é disparada pelo comando *xmaple* (ou *maple*) dado no sinal de pronto do sistema operacional. *Ela é o principal meio para gravar e ler os trabalhos desenvolvidos no Maple.*

A *worksheet* utiliza os recursos de janelas para facilitar a interação do usuário com o Maple. Por exemplo, um comando batido errado pode ser facilmente corrigido voltando-se o cursor para a posição do erro e substituindo os caracteres errados. Não há necessidade de digitar todo o comando novamente. Na *worksheet*, o usuário pode tecer comentários, colar gráficos e gravar todo o conjunto em um arquivo para ser lido e eventualmente modificado posteriormente. A *worksheet* pode ser impressa selecionando-se a opção *Print...* depois de clicar em *File*, ou pode ser convertida em um arquivo LaTeX[1]. Um exemplo de uso das *worksheets* é este curso. Ele foi desenvolvido em *worksheets* e posteriormente convertido em LaTeX para ser impresso.

A *worksheet* é um caderno virtual de anotações de cálculos. A vantagem do caderno virtual é que qualquer coisa já escrita pode ser modificada sem necessidade de fazer outras alterações, pois o trabalho se ajusta automaticamente às mudanças. Essa idéia é a mesma dos processadores de textos que substituíram as máquinas de escrever. A *worksheet* não é um processador de textos. Ela funciona de maneira satisfatória como um editor de textos, e a parte referente ao processamento de textos pode ser feita em LaTeX. No desenvolvimento de um trabalho usando a *worksheet*, é importante que ele seja feito em ordem e que todo rascunho seja apagado assim que cumprido seu objetivo. O comando *restart* pode encabeçar o trabalho. Depois de gravar a *worksheet*, o usuário pode sair do Maple. No momento em que a *worksheet* é lida novamente, os resultados que aparecem na tela não estão na memória ativa do Maple. É necessário processar os comandos novamente para ativar os resultados. A *worksheet* é gravada com a terminação *.mws*.

A *worksheet* tem quatro tipos de linhas: 1. linhas de entrada de comandos que usam a cor vermelha e são precedidas pelo sinal de pronto ">", 2. linhas de saída dos comandos na cor azul, 3. linhas de texto na cor preta e 4. linhas de gráfico. Algumas dessas linhas podem ser convertidas umas nas outras. Os botões *Copy*, *Paste* e *Cut* são bastantes úteis neste contexto. Nos casos que envolvem tipos diferentes de linha, os botões *Copy as Maple Text* e *Paste as Maple Text* podem ser o único meio de executar a tarefa. As linhas de saída usam recursos gráficos das janelas para escrever as letras, os símbolos e desenhar os gráficos. O sinal de integral aparece na tela como $\int$, o somatório como $\sum$ e as letras gregas como α, β, γ, $\cdots$. Existe uma opção que faz com que as linhas de saída usem os mesmos caracteres do teclado. Essa opção é útil para gravar resultados em um arquivo *ASCII* (acrônimo de *American Standard Code for Information Interchange*). A *worksheet* possui diversos recursos para escrever textos. É possível criar seções e sub-seções. As letras podem ter diversos tamanhos e estilos, e podem ser em

[1] LaTeX é um processador de textos de domínio público. Veja o *site http://www.latex-project.org.*

itálico ou em negrito. É possível criar *hiperlinks* que conectam diversas *worksheets*. A partir desses *hiperlinks* pode-se "navegar" através das *worksheets*.

Mais detalhes sobre as *worksheets* podem ser encontrados no *New User's Tour* depois de clicar em *Help*.

1.3 *Help on Line*

O Maple possui um sistema de ajuda interativo chamado *help on line*. Para pedir ajuda sobre uma função ou qualquer assunto pertinente, deve-se preceder a função ou o assunto com um sinal de interrogoção. Por exemplo:

```
> ?maple
```

A página de ajuda contém várias partes: 1. descrição da sintaxe, 2. descrição detalhada, 3. exemplos e 4. palavras chaves relacionadas ao assunto. A partir das palavras chaves, pode-se "navegar" pelo sistema de ajuda até que a informação desejada seja encontrada.

Outra modalidade de ajuda consiste na procura por assunto. No caso do *Maple for Windows* deve-se clicar com o *mouse* no *Help* e depois *Introduction* (primeiro de cima para baixo). A nova janela terá uma seção cinza na parte superior. Clique em *Mathematics...* por exemplo. Na segunda coluna procure por um sub-tópico e assim por diante.

Os usuários mais persistentes podem porcurar ajuda clicando em *Topic Search...* ou *Full Text Search...*, depois de ter clicado em *Help*.

1.4 Referências sobre Maple

Alguns livros sobre Maple estão listados nas referências bibliográficas. As referências [1] e [2] são os principais manuais e são distribuídos junto com o *software*. As referências [3], [4] e [5] são livros de introdução ao Maple. Uma lista de livros mais completa pode ser encontrada no *site* *http://www.maplesoft.com*. Clique em *Maple Application Center* e depois em *Publications*. Aproveite também para dar uma olhada no material que usuários do mundo inteiro enviam para o *Maple Application Center*.

Chapter 2

Noções Básicas

Para obter os melhores resultados, este texto deverá ser lido junto com o uso do Maple, de forma que os exemplos possam ser testados. Neste caso, as dúvidas e as dificuldades ficarão mais visíveis.

2.1 Números

O Maple usualmente trabalha com os números de maneira exata. Nenhuma aproximação é feita:

```
> (34*3 + 7/11)^2;
```

$$\frac{1274641}{121}$$

Podemos ver que o resultado é um número racional. Para obter uma aproximação decimal, devemos usar o comando *evalf* (*evaluate in floating point*):

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

$$10534.22314$$

O sinal de percentagem % guarda o valor do último resultado calculado, que não é necessariamente o resultado do comando imediatamente acima, pois na *worksheet* podemos processar os comandos numa ordem diferente da que eles aparecem na tela. O último resultado tem 10 dígitos (valor *default*). Podemos calcular com mais dígitos significativos, por exemplo, com 50 dígitos:

```
> evalf[50](%%);
```

$$10534.223140495867768595041322314049586776859504132$$

Outra maneira de ter resultados com casas decimais, é entrar pelo menos um número com casa decimal:

```
> 4/3*sin(2.);
```

$$1.212396569$$

Porém, como podemos aumentar o número de casas, se não estamos usando o comando *evalf*? Para isso devemos atribuir o número de dígitos desejado à variável *Digits*. Primeiro, vamos verificar seu valor *default*, e depois mudá-lo:

```
> Digits;
```

```
> Digits := 20;
```

$$Digits := 20$$

Vejamos agora o número de Euler com vinte dígitos:

```
> exp(1.);
```

$$2.7182818284590452354$$

No Maple, podemos trabalhar com números irracionais:

```
> ((1+sqrt(5))/2)^2;
```

$$(\frac{1}{2} + \frac{1}{2}\sqrt{5})^2$$

```
> expand(%);
```

$$\frac{3}{2} + \frac{1}{2}\sqrt{5}$$

Observe que a entrada não foi simplificada até que isto fosse pedido. Esta é uma regra do Maple. As expressões não são simplificadas a menos que o usuário peça. Somente são feitas as simplificações mais elementares, envolvendo as operações aritméticas básicas.

Vamos ver agora um exemplo mais elaborado:

```
> sin(2*Pi*n)/5!;
```

$$\frac{1}{120}\sin(2\,\pi\,n)$$

Pedimos para o Maple calcular $\sin(2\,\pi\,n)$ e dividir por fatorial de 5. Poderíamos achar que o resultado deveria ser zero, já que o seno de um múltiplo de π é zero. Com um pouco de reflexão mudaríamos de opinião, já que n é uma letra sobre a qual nada foi dito. Não é razoável esperar que o Maple assuma que ela é uma variável inteira. Para o Maple, ela é uma letra como qualquer outra. A princípio, todas estão em pé de igualdade. Isso pode ser mudado com o seguite comando:

```
> assume(n,integer);
> sin(2*Pi*n)/5!;
```

$$0$$

```
> cos(Pi*n);
```

$$(-1)^{n\tilde{}}$$

De agora em diante, o Maple tratará n como uma variável inteira. Ela passa a ser mostrada na tela com um til, da forma $n\tilde{}$, para o usuário saber que essa variável tem uma determinada propriedade atribuída pelo comando *assume*. É possível inibir o aparecimento do til nessas variáveis com o comando

```
> interface(showassumed=0);
```

Outro detalhe que podemos observar do último exemplo, é que o número π é escrito com P maiúsculo. A variável pi com p minúsculo não tem nenhuma relação com a constante matemática π. Vamos retirar a declaração antes de continuar:

```
> n := 'n';
```

$$n := n$$

Agora, vamos ver um exemplo com números complexos. Queremos encontrar as raízes cúbicas de:

```
> z := (-8)^(1/3);
```

$$z := (-8)^{1/3}$$

Podemos ver que o Maple nada fez, ou melhor, não houve nenhuma simplificação do resultado. Assim, vamos pedir explicitamente:

```
> simplify(z);
```

$$1 + I\sqrt{3}$$

A variável I representa $\sqrt{-1}$. O usuário pode escolher outra variável, por exemplo J, através da instrução `interface(imaginaryunit=J)`. Os números complexos são sempre simplificados para a forma $a + b\,I$. Por exemplo:

```
> (3+4*I)/(1+I);
```

$$\frac{7}{2} + \frac{1}{2}\,I$$

Isso não ocorre se a expressão contiver uma letra. Por exemplo:

```
> z := (3+4*I)/(1+a*I);
```

$$z := \frac{3 + 4\,I}{1 + I\,a}$$

Neste caso, devemos usar o comando *evalc* (*evaluate in complex context*). Vejamos:

```
> evalc(z);
```

$$\frac{3}{1+a^2} + 4\,\frac{a}{1+a^2} + I\,(\frac{4}{1+a^2} - 3\,\frac{a}{1+a^2})$$

Dentro do *evalc*, todas a letras são consideradas variáveis reais, exceto as variáveis que o usuário tenha declarado complexas através do comando *assume*. Podemos obter a parte real, imaginária e o módulo de z com a ajuda do *evalc*:

```
> normal(evalc(Re(z)));
```

$$\frac{3 + 4\,a}{1 + a^2}$$

```
> evalc(abs(z));
```

$$\frac{5}{\sqrt{1 + a^2}}$$

Lembre que a variável I é reservada e portanto o usuário não pode fazer uma atribuição do tipo

```
> I := 1;

Error, illegal use of an object as a name
```

Exitem várias outras variáveis reservadas no Maple que não podem sofrer atribuição. A maioria está protegida e a mensagem de erro indica claramente onde está o problema. No caso caso da variável I, a mensagem de erro é enigmática.

2.2 Atribuição de um Nome a uma Expressão

A princípio, uma expressão não precisa ter um nome. Podemos nos referir a ela com o sinal %. Nas situações em que vamos nos referir a uma expressão diversas vezes, é melhor dar um nome a ela. Isto é feito com o comando de atribuição ":=". Por exemplo:

```
> equacao := x^2+3*x+1=0;
```

$$equacao := x^2 + 3\,x + 1 = 0$$

```
> solve(equacao);
```

$$-\frac{3}{2} + \frac{1}{2}\sqrt{5},\ -\frac{3}{2} - \frac{1}{2}\sqrt{5}$$

O resultado também pode ser atribuído a uma variável:

```
> solucao1 := %[1];
```

$$solucao1 := -\frac{3}{2} + \frac{1}{2}\sqrt{5}$$

```
> solucao2 := %%[2];
```

$$solucao2 := -\frac{3}{2} - \frac{1}{2}\sqrt{5}$$

Podemos confirmar que *solucao1* é de fato uma solução da equação:

```
> expand(subs(x=solucao1, equacao));
```

$$0 = 0$$

Note que o sinal de igualdade "=" tem um papel bem diferente do sinal de atribuição ":=". O sinal de igualdade não modifica o valor de uma variável. Vejamos:

```
> y = x + 1;
```

$$y = x + 1$$

O valor de y continua sendo ele próprio:

```
> y;
```

$$y$$

Vejamos agora:

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

$$y := x + 1$$

```
> y;
```

$$x + 1$$

Para retirar a atribuição ("limpar a variável"), usa-se

```
> y:='y';
```

$$y := y$$

De modo equivalente:

```
> unassign('y');
```

2.3 Avaliação Completa

Vamos fazer uma série de atribuições em cadeia:

```
> A := B;
```

$$A := B$$

```
> B := C;
```

$$B := C$$

```
> C := 3;
```

$$C := 3$$

Agora, observe o valor que o Maple retorna para a variável A:

```
> A;
```

$$3$$

O que aconteceu foi que A foi avaliado como sendo B que por sua vez foi avaliado como sendo C que foi avaliado como sendo 3. Isso se chama "avaliação completa" de uma variável. Existem várias excessões a essa regra como veremos ao longo deste texto. Estas excessões são muito importantes. Podemos adiantar uma, a saber, no comando abaixo a variável A não será avaliada:

```
> A := 10;
```

$$A := 10$$

Caso A tivesse sido considerado com o valor 3, teríamos o comando 3 :=10, o que não é admitido. Existe uma maneira de retardar a avaliação de uma variável. Sabemos que o comando *A;* vai retornar 10. Porém, podemos retardar esta avaliação:

```
> 'A';
```

$$A$$

```
> %;
```

$$10$$

Isso explica por que o comando *A:='A';* "limpa" a variável A. O recurso de retardar a avaliação, seja de uma variável, seja de um comando, é utilizado com muita frequência.

2.4 Tipos de Objetos

Para usar o Maple de maneira eficiente, é necessário conhecer a forma de alguns objetos desta linguagem. Outro nome para *objetos* do Maple comumente usado é *estrutura de dados*. Os principais são as listas, conjuntos, *arrays*, sequências e tabelas. Vários comandos do Maple têm estes objetos como resultado. Podemos precisar selecionar um elemento do resultado e, para isto, é necessário compreender a estruturas destes objetos. O resultado do comando *solve* pode ser uma sequência de raízes de uma equação, como vimos acima. Vejamos outro exemplo:

```
> x^8+2*x^7-13*x^6-24*x^5+43*x^4+58*x^3-67*x^2- 36*x+36;
```

$$x^8 + 2\,x^7 - 13\,x^6 - 24\,x^5 + 43\,x^4 + 58\,x^3 - 67\,x^2 - 36\,x + 36$$

```
> sequencia_de_solucoes := solve(%);
```

$$\mathit{sequencia\_de\_solucoes} := -3,\ 3,\ -1,\ -2,\ -2,\ 1,\ 1,\ 1$$

O resultado do comando *solve* foi uma sequência de raízes, sendo que as raízes repetidas aparecem tantas vezes quanto for a multiplicidade. No caso acima, a raíz 1 tem multiplicidade 3 e a raíz -2 tem multiplicidade 2. Podemos, agora, colocar esta sequência de raízes na forma de outros objetos. Por exemplo, na forma de uma lista:

```
> lista_de_solucoes := [ sequencia_de_solucoes ];
```

$$\mathit{lista\_de\_solucoes} := [-3,\ 3,\ -1,\ -2,\ -2,\ 1,\ 1,\ 1]$$

Outra possibilidade é armazenar as raízes na forma de um conjunto:

```
> conjunto_de_solucoes := { sequencia_de_solucoes };
```

$$\mathit{conjunto\_de\_solucoes} := \{-1,\ 1,\ -2,\ -3,\ 3\}$$

Cada objeto tem sua característica própria. Podemos ver que a lista mantêm o ordenamento das raízes e não elimina as repetições. Por sua vez, o conjunto não respeita o ordenamento e elimina as repetições. O conjunto acima tem 5 elementos enquanto que a lista tem 8 elementos, número este que deve coincidir com o grau do polinômio.

Para selecionar um elemento de um objeto do Maple, deve-se usar a seguinte notação:

***OBJETO*[*posição do elemento*]**

Por exemplo, vamos selecionar o quarto elemento da sequência de soluções:

```
> sequencia_de_solucoes[4];
```

$$-2$$

O último elemento da lista de soluções:

```
> lista_de_solucoes[-1];
```

$$1$$

Neste último exemplo, podemos sempre selecionar o último elemento mesmo sem ter contado à mão (ou por inspeção visual) quantos elementos o objeto tem. Um número negativo indica que a contagem é feita a partir do final da lista: -1 indica o último elemento, -2 o penúltimo, e assim por diante. O comando *nops* fornece o tamanho da lista ou do conjunto. Para memorizar o nome do comando é bom saber que *nops* é o acrônimo de *number of operands*. Vamos usar com frequência o nome "operando" para nos referir a "elemento".

Vamos nos colocar o seguinte problema: queremos saber qual é o polinômio que tem as mesmas raízes que o polinômio acima, porém todas com multiplicidade 1. Ou seja, sabemos que as raízes do polinômio que queremos achar são:

```
> conjunto_de_solucoes;
```

$$\{-1, 1, -2, -3, 3\}$$

Temos então que construir o polinômio (x+1)*(x-1)... . A maneira mais simples de resolver este problema é escrever explicitamente o seguinte produto:

```
> expand((x+1)*(x-1)*(x+2)*(x+3)*(x-3));
```

$$x^5 + 2\,x^4 - 10\,x^3 - 20\,x^2 + 9\,x + 18$$

Caso o número de raízes seja grande, não é conveniente usar este método, pois ele será trabalhoso e podemos cometer erros. Vejamos uma outra forma de resolver o problema:

```
> map(elem -> x-elem, conjunto_de_solucoes);
```

$$\{x + 1,\, x - 3,\, x - 1,\, x + 2,\, x + 3\}$$

```
> convert(%,`*`);
```

$$(x + 1)\,(x - 1)\,(x + 2)\,(x + 3)\,(x - 3)$$

```
> expand(%);
```

$$x^5 + 2\,x^4 - 10\,x^3 - 20\,x^2 + 9\,x + 18$$

Este método é o mesmo não importa o número de raízes. Um erro comum de ser cometido no primeiro método é usar $x-2$ no lugar de $x+2$. No segundo não se corre este risco. Aqui foi usado o comando *map* (*mapping*). Ele está associado às estruturas de dados. Todos os comandos que usamos até agora atuavam em um único elemento. No entanto, quando estamos lidando com estrutura de dados ou objetos com muitos elementos, precisamos aplicar comandos ou funções a todos os elementos da estrutura. O comando que faz isso é o *map*. No exemplo acima, queremos somar x a cada elemento do conjunto de soluções. A notação é $elem \to x + elem$. Esta operação deve ser realizada em todos os elementos do conjunto de soluções. O primeiro argumento do comando *map* deve ser a lei transformação. O segundo argumento tem qqe ser o conjunto ou qualquer objeto com vários elementos. O resultado foi o desejado. Cada elemento foi somado a x. Falta mais um passo para obter o polinômio, que é converter o conjunto em produto. O produto é especificado no Maple como '*'. As crases são necessárias, pois o asterisco é um caracter não alfanumérico. O comando *convert* espera que o seu segundo argumento seja um nome. O asterisco não é um nome, mas sim o operador de multiplicação. As crases fazem com que ele seja apenas um nome, ou um caracter como qualquer outra letra.

Podemos converter uma lista em um conjunto e vice-versa:

```
> convert( conjunto_de_solucoes, list);
```

$$[-1, 1, -2, -3, 3]$$

Os conjuntos são objetos inspirados nos conjuntos usados em Matemática. Podemos fazer a união, interseção e subtração de conjuntos com os comandos *union*, *intersect* e *minus*. Por exemplo:

```
> {1,2,3} union {a,b,c,d};
```

$$\{1, 2, 3, a, b, c, d\}$$

```
> % intersect {3,a,c,w,z};
```

$$\{3, a, c\}$$

```
> %% minus %;
```

$$\{1, 2, b, d\}$$

Os conjuntos e as listas que têm uma certa regra de formação podem ser gerados com o comando *seq* (*sequence*). Por exemplo, o conjunto dos 10 primeiros números primos:

```
> { seq(ithprime(i), i=1..10) };
```

$$\{2, 3, 5, 7, 11, 13, 17, 19, 23, 29\}$$

A lista dos dez primeiros números da forma $2^p - 1$:

```
> [ seq(2^i-1, i=1..10) ];
```

$$[1, 3, 7, 15, 31, 63, 127, 255, 511, 1023]$$

A lista das derivadas de $x \ln(x)$ em relação a x até ordem 5:

```
> [ seq(diff(x*ln(x),x$i), i=1..5) ];
```

$$[\ln(x) + 1, \frac{1}{x}, -\frac{1}{x^2}, \frac{2}{x^3}, -\frac{6}{x^4}]$$

Vimos como selecionar um único elemento de uma estrutura de dados. Agora, como podemos obter um sub-conjunto de um conjunto ou uma sub-lista de uma lista? A notação para obter sub-listas ou sub-conjuntos é:

$$\boldsymbol{OBJETO}[\ \boldsymbol{a}\ ..\ \boldsymbol{b}\ ]$$

onde a é a posição do primeiro elemento e b é a posição do último elemento da parte que queremos selecionar. Por exemplo:

```
> lista1 := [a, b, c, d, e];
```

$$lista1 := [a, b, c, d, e]$$

```
> sublista := lista1[2..4];
```

$$sublista := [b, c, d]$$

Chapter 3

Simplificação de Expressões

3.1 Introdução

O problema de simplificação de expressões é um problema clássico da computação algébrica. Todo mundo tem uma idéia intuitiva do que seja a forma mais simples de uma expressão. No entanto, na computação não é possível dar um comando intuitivo. É necessário fornecer um algoritmo, de forma que, uma vez seguida uma sequência de passos, o computador sempre chegue a forma mais simples. Uma regra poderia ser: fatores não nulos comuns ao numerador e denominador de uma expressão devem ser cancelados. O comando *normal* (*normalize*) executa essa tarefa. Por exemplo:

```
> (x^2-1)/(x^2-x-2);
```

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

```
> normal(%);
```

$$\frac{x-1}{x-2}$$

O resultado ficou mais simples que o original. Agora, vamos aplicar a mesma regra à seguinte expressão:

```
> (x^20-1)/(x-1);
```

$$\frac{x^{20}-1}{x-1}$$

```
> normal(%);
```

$$x^{19}+x^{18}+x^{17}+x^{16}+x^{15}+x^{14}+x^{13}+x^{12}+x^{11}+x^{10}$$
$$+x^9+x^8+x^7+x^6+x^5+x^4+x^3+x^2+x+1$$

Neste exemplo o numerador tem grau 20. Imagine o cancelamento no caso de numerador de grau 1000. A expressão sem fatores comuns ocuparia várias páginas. Qual das formas é a mais simples? Existem situações onde é possivel decidir qual é a forma mais simples. Isto é o que tenta fazer o comando *simplify*. Em outras situações o próprio usuário deve interferir para obter a forma de sua preferência. Para isso, o Maple possui uma série de comandos de simplificação. São comandos que, quando aplicados a uma expressão, mudam sua forma, mas não mudam seu conteúdo. As diferentes formas são matematicamente equivalentes, ou melhor, quase equivalentes. As duas últimas expressões são equivalentes em todos os pontos diferentes de 1, porém não são equivalentes em $x=1$.

Os comandos mais usados para simplificar expressões algébricas são: *expand*, *normal*, *simplify*, *collect*, *combine* e *factor*.

3.2 *Expand*

Quando um usuário entra uma expressão, o Maple a mantém na mesma forma por via de regra. A expressão não tem a sua forma alterada até que o usuário peça. Não há um padrão predeterminado de apresentação das expressões algébricas. Por isto, elas são armazenadas internamente, na forma em que foram fornecidas. Por exemplo, uma expressão dada na forma fatorada, permanece nesta forma até que sua expansão seja pedida, e vice-versa:

```
> (x-1)^5;
```

$$(x-1)^5$$

```
> expand(%);
```

$$x^5 - 5\,x^4 + 10\,x^3 - 10\,x^2 + 5\,x - 1$$

```
> factor(%);
```

$$(x-1)^5$$

O comando *expand* serve para expandir as expressões no sentido de tirar os parênteses, e serve também para expandir funções trigonométricas, logarítmicas, etc. Por exemplo:

```
> sin(2*alpha+beta) = expand(sin(2*alpha+beta));
```

$$\sin(2\,\alpha+\beta) = 2\cos(\beta)\sin(\alpha)\cos(\alpha) + 2\sin(\beta)\cos(\alpha)^2 - \sin(\beta)$$

```
> ln(x^2*y^2) = expand(ln(x^2*y^2));
```

$$\ln(x^2\,y^2) = 2\ln(x) + 2\ln(y)$$

```
> Psi(2*x) = expand(Psi(2*x)); # Psi e' a funcao digamma
```

$$\Psi(2\,x) = \ln(2) + \frac{1}{2}\,\Psi(x) + \frac{1}{2}\,\Psi(x+\frac{1}{2})$$

Em certos casos, queremos expandir uma expressão sem expandir um certo pedaço. Para isso, devemos colocar a parte que queremos congelar como segundo argumento do comando *expand*:

```
> (x + sin(gamma + delta))^2;
```

$$(x + \sin(\gamma+\delta))^2$$

```
> expand(%,sin);
```

$$x^2 + 2\sin(\gamma+\delta)\,x + \sin(\gamma+\delta)^2$$

```
> sin(omega*(t+t0)+delta);
```

$$\sin(\omega\,(t+t0)+\delta)$$

```
> expand(%,t+t0);
```

$$\sin(\omega\,(t+t0))\cos(\delta) + \cos(\omega\,(t+t0))\sin(\delta)$$

Nestes últimos exemplos, os resultados seriam bem diferentes se não tivéssemos colocado o segundo argumento. No caso da expressão $\sin(\gamma+\theta)$ basta colocar *sin* no segundo argumento do comando *expand* para obter o efeito desejado. Evidentemente, se tivéssemos colocado $\sin(\gamma+\theta)$ em vez de *sin,* obteríamos o mesmo resultado.

Um terceiro efeito do comando *expand* se refere a expressões com denominador. Ele coloca o denominador embaixo de cada numerador, sem expandir o denominador:

```
>   expr := (x+y)^2/(a+b)^2;
```

$$expr := \frac{(x+y)^2}{(a+b)^2}$$

```
>   expand(expr);
```

$$\frac{x^2}{(a+b)^2} + 2\,\frac{x\,y}{(a+b)^2} + \frac{y^2}{(a+b)^2}$$

Para expandir o denominador, é necessário primeiro usar o comando *normal* com a opção *expanded* e depois expandir com *expand*:

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

$$\frac{x^2+2\,x\,y+y^2}{a^2+2\,a\,b+b^2}$$

```
>   expand(%);
```

$$\frac{x^2}{a^2+2\,a\,b+b^2} + 2\,\frac{x\,y}{a^2+2\,a\,b+b^2} + \frac{y^2}{a^2+2\,a\,b+b^2}$$

3.3 *Combine*

Vimos na seção anterior como expandir expressões. A expansão é um procedimento simples computacionalmente. O mesmo não podemos dizer do procedimento inverso. Ao contrário, podemos afirmar que a fatoração é um processo complexo, baseado em um longo algoritmo desenvolvido com conceitos da Álgebra Moderna. A fatoração é feita pelo comando *factor*. Além do comando *factor*, os comandos *normal* e *combine* fazem o contrário do que faz o comando *expand*. Destes três, o comando *combine* requer maior atenção, pois para usá-lo com eficiência é necessário conhecer as opções que devem ser fornecidas como segundo argumento. A sintaxe é:

***combine*(*expressão, opção*)**

A ***opção*** pode ser: *exp, ln, power, trig, Psi, polylog, radical, abs, signum, plus, atatsign, conjugate, plot, product* ou *range* entre outras. A opção *trig* engloba todas as funções trigonométricas e a opção *power* expressões que envolvem potenciação. Vejamos alguns exemplos:

```
>   (x^a)^2*x^b = combine((x^a)^2*x^b, power);
```

$$(x^a)^2\,x^b = x^{(2\,a+b)}$$

```
>   4*sin(x)^3 = combine(4*sin(x)^3, trig);
```

$$4\sin(x)^3 = -\sin(3\,x) + 3\sin(x)$$

```
>   exp(x)^2*exp(y) = combine(exp(x)^2*exp(y), exp);
```

$$(e^x)^2\,e^y = e^{(2\,x+y)}$$

```
> exp(sin(a)*cos(b))*exp(cos(a)*sin(b)) =
> combine(exp(sin(a)*cos(b))*
> exp(cos(a)*sin(b)), [trig,exp]);
```

$$e^{(\sin(a)\cos(b))}\, e^{(\cos(a)\sin(b))} = e^{\sin(a+b)}$$

Neste último exemplo, usamos o comando *combine* com duas opções. Neste caso, temos que colocar as opções entre colchetes. A opção *ln* pode não funcionar como desejado se não usarmos a opção adicional *symbolic*, por exemplo:

```
> combine(5*ln(x)-ln(y), ln); # nada acontece
```

$$5\ln(x) - \ln(y)$$

```
> 5*ln(x)-ln(y) = combine(5*ln(x)-ln(y), ln,
> symbolic); # agora sim!
```

$$5\ln(x) - \ln(y) = \ln(\frac{x^5}{y})$$

Uma outra forma de obter esse resultado é declarar as variáveis envolvidas como sendo reais e positivas com o comando *assume*. Vejamos mais alguns exemplos:

```
> Int(x,x=a..b)-Int(x^2,x=a..b)=
> combine(Int(x,x=a..b)-Int(x^2,x=a..b));
```

$$\int_a^b x\,dx - \int_a^b x^2\,dx = \int_a^b x - x^2\,dx$$

```
> log(x)+Limit(f(x)^3,x=a)*Limit(f(x)^2,x=a) =
> combine(log(x)+Limit(f(x)^3, x=a)*Limit(f(x)^2, x=a));
```

$$\ln(x) + \lim_{x\to a} (f(x))^3 \lim_{x\to a} (f(x))^2 = \ln(x) + \lim_{x\to a} (f(x))^5$$

Nos dois últimos exemplos, usamos os comandos *int* e *limit* nas suas formas inertes, isto é, com iniciais maiúsculas. Neste caso, a integração e o limite não são executados até que seja pedido através do comando *value*. No entanto, as propriedades da integral e do limite são conhecidas nesta forma inerte.

3.4 *Convert*

A princípio, a função *convert* não é um comando de simplificação pois ele tem um propósito mais geral. No entanto, algumas conversões servem como simplificação. A sintaxe deste comando é:

***convert*(*expressão*, *tipo*)**

onde ***tipo*** pode ser um dos seguintes nomes: *trig, tan, ln, exp, expln, expsincos, rational, parfrac, radians, degree, GAMMA, factorial*, entre outros, no caso de ser tratar de conversão de uma expressão algébrica em outra expressão algébrica. Vejamos alguns exemplos:

```
> expr := (1+I)*(exp(-I*x)-I*exp(I*x))/2;
```

$$expr := (\frac{1}{2} + \frac{1}{2} I)\,(e^{(-I\,x)} - I\,e^{(I\,x)})$$

```
> expand(convert(%,trig));
```

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

```
> convert(cosh(x),exp);
```

$$\frac{1}{2}\,e^x + \frac{1}{2}\,\frac{1}{e^x}$$

```
> convert(arcsinh(x),ln);
```

$$\ln(x + \sqrt{x^2 + 1})$$

Vejamos alguns exemplos relacionados com a função *factorial*:

```
> n!  = convert(n!, GAMMA);
```

$$n! = \Gamma(n + 1)$$

```
> convert(%,factorial);
```

$$n! = \frac{(n + 1)!}{n + 1}$$

```
> expand(%);
```

$$n! = n!$$

```
> binomial(n,k) = convert(binomial(n,k), factorial);
```

$$\mathrm{binomial}(n,\, k) = \frac{n!}{k!\,(n - k)!}$$

3.5 *Simplify*

O comando *simplify* é um comando geral de simplificação. Uma das primeiras coisas que ele faz é procurar dentro da expressão a ocorrência de funções matemáticas, como as funções trigonométricas. Caso encontre, ele usa as propriedades de simplificação destas funções. O mesmo ocorre com as raízes quadradas, os radicais e as potências. No entanto, é possível aplicar as regras de simplificação de determinadas funções de maneira selecionada. Para isso deve-se dar o nome da função em questão no segundo argumento do comando *simplify*, que pode ser um dos seguintes nomes: *trig, hypergeom, radical, power, exp, ln, sqrt, BesselI, BesselJ, BesselK, BesselY, Ei, GAMMA, RootOf, LambertW, dilog, polylog, pg, pochhammer* e *atsign* ("@"). Por exemplo, no primeiro comando *simplify* abaixo, tanto as regras de simplificação das funções trigonométricas como da função exponencial foram usadas, enquanto que no segundo somente as regras das funções trigonométricas foram usadas:

```
> expr := (sin(x)^3 + cos(x)^3)*exp(a)/exp(a+b);
```

$$expr := \frac{(\sin(x)^3 + \cos(x)^3)\,e^a}{e^{(a+b)}}$$

```
> simplify(expr);
```

$$-(-\sin(x) + \sin(x)\cos(x)^2 - \cos(x)^3)\,e^{(-b)}$$

```
> simplify(expr, trig);
```

$$-\frac{e^a\,(-\sin(x) + \sin(x)\cos(x)^2 - \cos(x)^3)}{e^{(a+b)}}$$

O comando *simplify* tem uma opção bastante útil para simplificação de uma expressão com vínculos adicionais. Estes vínculos devem ser colocados em um conjunto, mesmo que haja um único vínculo, e repassados como segundo argumento do comando *simplify*. Por exemplo, queremos calcular o valor de $a^4 + b^4 + c^4$ com a, b e c satisfazendo as seguintes equações: $a + b + c = 3$, $a^2 + b^2 + c^2 = 9$ e $a^3 + b^3 + c^3 = 24$:

```
> vinculos := {a+b+c=3, a^2+b^2+c^2=9,a^3+b^3+c^3=24};
```

$$vinculos := \{a + b + c = 3,\ a^2 + b^2 + c^2 = 9,\ a^3 + b^3 + c^3 = 24\}$$

```
> simplify(a^4+b^4+c^4, vinculos);
```

$$69$$

Outro exemplo de aplicação do comando *simplify* com vículos é na substituição de dois termos de um produto por um número. Por exemplo, queremos queremos substituir $a\,b$ por 10 na expressão $a\,b\,c$:

```
> expr := a*b*c;
```

$$expr := a\,b\,c$$

```
> subs( a*b=10, expr);
```

$$a\,b\,c$$

Podemos ver que não funcionou. Agora, usando o comando *simplify* com vínculos:

```
> simplify( expr, {a*b=10});
```

$$10\,c$$

É possível também dizer ao comando *simplify* que as variáveis obedecem as certas restrições, que são as mesmas usadas no comando *assume*. Por exemplo, a raíz quadrada do produto de vários termos só é o produto das raízes quadradas se os termos forem reais e positivos. Vejamos:

```
> expr := simplify(sqrt(x^2*y^2));
```

$$expr := \sqrt{x^2\,y^2}$$

```
> simplify(expr, assume=nonneg);
```

$$x\,y$$

A opção *assume=nonneg* faz com que todas as variáveis sejam declaradas temporariamente como variáveis não negativas. No caso da expressão acima, temos outra forma de obter o mesmo resultado:

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

$$x\,y$$

Chapter 4

Álgebra Linear

4.1 Introdução

Os comandos de Álgebra Linear formam um pacote chamado *LinearAlgebra*, que deve ser carregado com o comando *with*:

```
> restart;
> with(LinearAlgebra):
```

Normalmente, terminamos o comando de carregar pacotes com dois pontos para que suas funções não sejam mostradas, especialmente com relação ao pacote *LinearAlgebra* pois ele possui um quantidade bem grandes de funções. Para ter acesso à lista das funções do pacote e suas respectivas páginas de ajuda, devemos dar o comando *?LinearAlgebra*. Vamos começar vendo como definir matrizes.

4.2 Definindo uma Matriz

O principal comando para definir matrizes é *Matrix. Por exemplo*

```
> A := Matrix( [ [1,2], [3,4], [5,6] ] );
```

$$A := \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

```
> B := Matrix( [ [a,b], [d,e] ] );
```

$$B := \begin{bmatrix} a & b \\ d & e \end{bmatrix}$$

Podemos fazer agora operações algébricas com as matrizes A e B:

```
> A + A;
```

$$\begin{bmatrix} 2 & 4 \\ 6 & 8 \\ 10 & 12 \end{bmatrix}$$

```
> 3*A;
```

$$\begin{bmatrix} 3 & 6 \\ 9 & 12 \\ 15 & 18 \end{bmatrix}$$

```
> A . B;
```

$$\begin{bmatrix} a+2\,d & b+2\,e \\ 3\,a+4\,d & 3\,b+4\,e \\ 5\,a+6\,d & 5\,b+6\,e \end{bmatrix}$$

Note que usamos o produto usual "*" para multiplicação por escalar, e usamos a multiplicação não-comutativa "." para produto de matrizes. O produto comutativo não pode ser usado para matrizes

```
> A * B;
Error, (in rtable/Product) invalid arguments
```

Podemos também elevar uma matriz a um número inteiro. A potenciação por um número negativo quer dizer a inversão da matriz, e subsequente potenciação pelo módulo do número.

```
> inv_B := B^(-1);
```

$$inv\_B := \begin{bmatrix} \dfrac{e}{a\,e-b\,d} & -\dfrac{b}{a\,e-b\,d} \\ -\dfrac{d}{a\,e-b\,d} & \dfrac{a}{a\,e-b\,d} \end{bmatrix}$$

Vamos testar se *inv_B* é o inverso de B

```
> inv_B . B;
```

$$\begin{bmatrix} \dfrac{e\,a}{a\,e-b\,d} - \dfrac{b\,d}{a\,e-b\,d} & 0 \\ 0 & \dfrac{e\,a}{a\,e-b\,d} - \dfrac{b\,d}{a\,e-b\,d} \end{bmatrix}$$

Para simplificar cada elemento da matriz é necessário usar o comando *Map*

```
> Map(simplify, %);
```

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Podemos modificar um elemento da matriz da seguinte forma

```
> A[2,1] := alpha;
```

$$A_{2,1} := \alpha$$

```
> A;
```

$$\begin{bmatrix} 1 & 2 \\ \alpha & 4 \\ 5 & 6 \end{bmatrix}$$

Quando uma matriz tem uma regra de formação, é possível repassar esta regra como terceiro argumento do comando *Matrix*. Os dois primeiros argumentos devem ser as dimensões da matriz. Vamos definir uma matriz de dimensão *3x4*, onde o elemento *<i,j>* é dado por $\frac{i}{j}$:

```
> Matrix(3, 4, (i,j) -> i/j);
```

$$\begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ 2 & 1 & \frac{2}{3} & \frac{1}{2} \\ 3 & \frac{3}{2} & 1 & \frac{3}{4} \end{bmatrix}$$

Uma notação mais econômica para definir matrizes ou vetores é

```
> C := < <alpha, beta> | <gamma,delta> >;
```

$$C := \begin{bmatrix} \alpha & \gamma \\ \beta & \delta \end{bmatrix}$$

As linhas são separadas pela barra vertical —, e cada linha deve ficar dentro de < ... >. Veja mais detalhes em *?MVshortcuts*. Duas operações usuais com matrizes são o cálculo do traço, matriz transposta e posto.

```
> Trace(C);
```

$$\alpha + \delta$$

```
> Transpose(C);
```

$$\begin{bmatrix} \alpha & \beta \\ \gamma & \delta \end{bmatrix}$$

```
> Rank(C);
```

$$2$$

4.3 Matrizes Especiais

Existem várias matrizes especiais que são usadas com frequência em Álgebra Linear. Muitas delas têm comandos específicos para gerá-las. Os comandos para definir matrizes especiais são

> *BandMatrix*, *BezoutMatrix*, *CompanionMatrix*, *ConstantMatrix*, *DiagonalMatrix*, *GenerateMatrix*, *GivensRotationMatrix*, *HankelMatrix*, *HilbertMatrix*, *HouseholderMatrix*, *IdentityMatrix*, *JordanBlockMatrix*, *RandomMatrix*, *ScalarMatrix*, *SylvesterMatrix*, *ToeplitzMatrix*, *VandermondeMatrix*, *ZeroMatrix*.

Na maioria dos casos, podemos prever o que o comando faz pelo nome. Por exemplo, para definir uma matriz nula 4x4

```
> ZeroMatrix(4);
```

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

O comando *DiagonalMatrix* cria matrizes diagonal em blocos. A entrada deve ser uma lista de matrizes ou escalares.

```
> DiagonalMatrix([A,B,1]);
```

$$\begin{bmatrix} 1 & 2 & 0 & 0 & 0 \\ \alpha & 4 & 0 & 0 & 0 \\ 5 & 6 & 0 & 0 & 0 \\ 0 & 0 & a & b & 0 \\ 0 & 0 & d & e & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

O comando *GenerateMatrix* é usado para construir uma matriz a partir dos coeficientes de um sistema de equações lineares. Considere o seguinte sistema

```
> eqs := [ 3*a + 2*b + 3*c - 2*d = 1,
> a + b + c = 3,
> a + 2*b + c - d = 2 ]:
> vars := [ a, b, c, d ]:
```

Vamos fazer a seguinte atribuição dupla

```
> A,v := GenerateMatrix( eqs, vars );
```

$$A,\, v := \begin{bmatrix} 3 & 2 & 3 & -2 \\ 1 & 1 & 1 & 0 \\ 1 & 2 & 1 & -1 \end{bmatrix},\ \begin{bmatrix} 1 \\ 3 \\ 2 \end{bmatrix}$$

Note que tanto A

```
> A;
```

$$\begin{bmatrix} 3 & 2 & 3 & -2 \\ 1 & 1 & 1 & 0 \\ 1 & 2 & 1 & -1 \end{bmatrix}$$

quanto v

```
> v;
```

$$\begin{bmatrix} 1 \\ 3 \\ 2 \end{bmatrix}$$

foram atribuidos pelas repectivas matrizes. A equação matricial correspondente ao sistema de equações pode ser obtida da seguinte forma usando o comando *Vector*

```
> A . Vector(vars) = v;
```

$$\begin{bmatrix} 3\,a + 2\,b + 3\,c - 2\,d \\ a + b + c \\ a + 2\,b + c - d \end{bmatrix} = \begin{bmatrix} 1 \\ 3 \\ 2 \end{bmatrix}$$

A opção *augmented=true* é uma forma alternativa de usar o comando GenerateMatrix

```
> B := GenerateMatrix( eqs, vars, augmented=true );
```

$$B := \begin{bmatrix} 3 & 2 & 3 & -2 & 1 \\ 1 & 1 & 1 & 0 & 3 \\ 1 & 2 & 1 & -1 & 2 \end{bmatrix}$$

Nesse caso podermos repassar o resultado diretamente para o comando *LinearSolve*

```
> LinearSolve(B);
```

$$\begin{bmatrix} 1 - \_t3_3 \\ 2 \\ \_t3_3 \\ 3 \end{bmatrix}$$

A variável $\_t3_3$ é um parâmetro livre da solução.

O comando *RandomMatrix* serve para gerar matrizes randômicas. Vamos gerar uma matriz randômica triangular com elementos inteiros no intervalo 1..10.

```
> randomize():
> RandomMatrix(4,4,generator=1..10,outputoptions=
> [shape=triangular[upper]]);
```

$$\begin{bmatrix} 10 & 3 & 10 & 4 \\ 0 & 7 & 9 & 5 \\ 0 & 0 & 9 & 1 \\ 0 & 0 & 0 & 2 \end{bmatrix}$$

O comando *randomize*( ) usa a hora corrente como semente para o algoritmo de randomização. Vamos passar agora para importantes comandos do pacote.

4.4 Autovalores e Autovetores

Os comandos relativos a cálculo de autovetores e autovalores são

> *CharacteristicMatrix*, *CharacteristicPolynomial*, *Eigenvalues*, *Eigenvectors*, *MinimalPolynomial*.

Vamos definir uma matriz

```
> A := Matrix( [[0,1], [epsilon,0]] );
```

$$A := \begin{bmatrix} 0 & 1 \\ \varepsilon & 0 \end{bmatrix}$$

O comando *Eigenvectors* fornece uma sequência de 2 elementos, o primeiro corresponde aos autovalores, e o segundo corresponde aos autovetores

```
> val,vec := Eigenvectors(A);
```

$$val,\ vec := \begin{bmatrix} \sqrt{\varepsilon} \\ -\sqrt{\varepsilon} \end{bmatrix},\ \begin{bmatrix} \frac{1}{\sqrt{\varepsilon}} & -\frac{1}{\sqrt{\varepsilon}} \\ 1 & 1 \end{bmatrix}$$

Vamos selecionar o primeiro autovetor:

```
> v1 := vec[1..2,1];
```

$$v1 := \begin{bmatrix} \frac{1}{\sqrt{\varepsilon}} \\ 1 \end{bmatrix}$$

e vamos confirmar que *v1* de fato é o autovetor correspondente ao primeiro autovalor

```
> simplify(A . v1 - val[1]*v1, symbolic);
```

$$\begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

O comando *simplify*(..., *symbolic*) deve ser usado pois multiplicamos uma expressão não numérica ao autovetor *v1*. Observe que o comando *simplify* sem a opção *symbolic* não simplifica como desejado

```
> simplify(val[1]*v1);
```

$$\sqrt{\varepsilon}\begin{bmatrix} \frac{1}{\sqrt{\varepsilon}} \\ 1 \end{bmatrix}$$

no entanto com a opção *symbolic*

```
> simplify(val[1]*v1, symbolic);
```

$$\begin{bmatrix} 1 \\ \sqrt{\varepsilon} \end{bmatrix}$$

Uma maneira alternativa de separar os diversos autovalores é usar a opção *output=list*

```
> vec := Eigenvectors(A, output=list);
```

$$vec := \left[\left[\sqrt{\varepsilon},\, 1,\, \left\{\begin{bmatrix} \frac{1}{\sqrt{\varepsilon}} \\ 1 \end{bmatrix}\right\}\right],\, \left[-\sqrt{\varepsilon},\, 1,\, \left\{\begin{bmatrix} -\frac{1}{\sqrt{\varepsilon}} \\ 1 \end{bmatrix}\right\}\right]\right]$$

Neste caso a saída é uma lista de listas com o autovalor, sua multiplicidade e o conjunto de autovetores correspondentes. O conjunto tem mais de um elemento quando o autovalor é degenerado. O primeiro autovetor é selecionado da seguinte forma

```
> vec[1][3][1];
```

$$\begin{bmatrix} \frac{1}{\sqrt{\varepsilon}} \\ 1 \end{bmatrix}$$

onde *vec*[1][3][1] deve ser lido da seguinte maneira. *vec*[1] é a primeira componente da lista *vec*. *vec*[1][3] é a terceira componente da lista *vec*[1]. *vec*[1][3][1] é o primeiro elemento do conjunto *vec*[1][3] . O polinômio característico é obtido com o comando *CharacteristicPolynomial:*

```
> CharacteristicPolynomial(A,x);
```

$$x^2 - \varepsilon$$

O polinômio mínimo é obtido com o comando *MinimalPolynomial:*

```
> MinimalPolynomial(A,x);
```

$$x^2 - \varepsilon$$

O polinômio mínimo é igual ao polinômio característico se todos os autovalores forem diferentes (multiplicidade 1), como é o caso acima.

4.5 Manipulação Estrutural de Matrizes

Os principais comandos para manipulação estrutural com matrizes são

> *DeleteRow, DeleteColumn, Dimension, RowDimension, ColumnDimension, Map, Row, Column, RowOperation, ColumnOperation, SubMatrix, Zip.*

A maioria dos nomes dos comandos acima falam por si só. As terminações ou prefixos *Row* e *Column* se referem a linha e coluna, respectivamente. Todos os comandos com prefixo *Row* têm seu equivalente com prefixo *Column*. O comando *RowDimension*, por exemplo, fornece o número de linhas da matriz enquanto que *ColumnDimension* fornece o número de colunas da matriz. Vejamos alguns exemplos dos comandos desta seção.

```
> A := Matrix(2,3, (i,j) -> Lambda||i||j);
```

$$A := \begin{bmatrix} \Lambda 11 & \Lambda 12 & \Lambda 13 \\ \Lambda 21 & \Lambda 22 & \Lambda 23 \end{bmatrix}$$

```
> B := Matrix(2,3, (i,j) -> lambda||i||j);
```

$$B := \begin{bmatrix} \lambda 11 & \lambda 12 & \lambda 13 \\ \lambda 21 & \lambda 22 & \lambda 23 \end{bmatrix}$$

Podemos juntar as matrizes A e B lateralmente

```
> M1 := Matrix([A,B]);
```

$$M1 := \begin{bmatrix} \Lambda 11 & \Lambda 12 & \Lambda 13 & \lambda 11 & \lambda 12 & \lambda 13 \\ \Lambda 21 & \Lambda 22 & \Lambda 23 & \lambda 21 & \lambda 22 & \lambda 23 \end{bmatrix}$$

ou verticalmente

```
> M2 := Matrix([[A],[B]]);
```

$$M2 := \begin{bmatrix} \Lambda 11 & \Lambda 12 & \Lambda 13 \\ \Lambda 21 & \Lambda 22 & \Lambda 23 \\ \lambda 11 & \lambda 12 & \lambda 13 \\ \lambda 21 & \lambda 22 & \lambda 23 \end{bmatrix}$$

Podemos extrair uma sub-matriz de uma matriz com o comando *SubMatrix*

```
> M3 := SubMatrix(M2,2..3,1..3);
```

$$M3 := \begin{bmatrix} \Lambda 21 & \Lambda 22 & \Lambda 23 \\ \lambda 11 & \lambda 12 & \lambda 13 \end{bmatrix}$$

Podemos multiplicar uma determinada linha de uma matriz por um escalar:

```
> RowOperation(M3,2,10);
```

$$\begin{bmatrix} \Lambda 21 & \Lambda 22 & \Lambda 23 \\ 10\,\lambda 11 & 10\,\lambda 12 & 10\,\lambda 13 \end{bmatrix}$$

Podemos apagar uma ou mais linhas com o comando *DeleteRow.*

```
> DeleteRow(%,1..1);
```

$$\begin{bmatrix} 10\,\lambda 11 & 10\,\lambda 12 & 10\,\lambda 13 \end{bmatrix}$$

O comando *Zip* é equivalente ao comando *zip* . O comando *zip* com *z* minúsculo atua basicamente em listas enquanto que *Zip* com *Z* maiúsculo atua em matrizes. Vejamos um exemplo.

```
> Zip( (x,y) -> {x,y}, A, B);
```

$$\begin{bmatrix} \{\Lambda 11, \lambda 11\} & \{\Lambda 12, \lambda 12\} & \{\Lambda 13, \lambda 13\} \\ \{\Lambda 21, \lambda 21\} & \{\Lambda 22, \lambda 22\} & \{\Lambda 23, \lambda 23\} \end{bmatrix}$$

O primeiro argumento do comando *Zip* deve ser uma função com 2 argumentos. Os valores de x (primeiro argumento) são as componentes da matriz A, enquanto que os valores de y (segundo argumento) são as componentes da matriz B. Se A e B têm a mesma dimensão, o resultado será uma nova matriz com a mesma dimensão. Se elas tem dimensões diferentes, a dimensão da matriz resultante vai depender de parâmetros adicionais do comando *Zip.* Veja o *help on line* neste caso.

4.6 Cálculo Vetorial

Os vetores linha são criados pelo comando *Vector.*

```
> v1 := Vector([a,b,c]);
```

$$v1 := \begin{bmatrix} a \\ b \\ c \end{bmatrix}$$

Por *default*, o vetor coluna é criado, o equivalente vetor linha é obtido com o comando

```
> v1a := Vector[row]([a,b,c]);
```

$$v1a := [a, b, c]$$

Pode-se usar uma função para calcular cada elemento do vetor.

```
> v2 := Vector(3, n->x^n);
```

$$v2 := \begin{bmatrix} x \\ x^2 \\ x^3 \end{bmatrix}$$

As operações aritméticas são realizadas usando os operadores usuais de soma e multiplicação.

```
> v1 + v2;
```

$$\begin{bmatrix} a+x \\ b+x^2 \\ c+x^3 \end{bmatrix}$$

```
> 2*v1;
```

$$\begin{bmatrix} 2a \\ 2b \\ 2c \end{bmatrix}$$

A multiplicação por um coeficiente simbólico não é executado automaticamente.

```
> alpha*v1;
```

$$\alpha \begin{bmatrix} a \\ b \\ c \end{bmatrix}$$

O comando *simplify*(..., *symbolic*) é necessário para especificar que o coeficente é um escalar.

```
> simplify(alpha*v1,symbolic);
```

$$\begin{bmatrix} \alpha\, a \\ \alpha\, b \\ \alpha\, c \end{bmatrix}$$

Uma forma alternativa é

```
> simplify(alpha*v1,assume=scalar);
```

$$\begin{bmatrix} \alpha\, a \\ \alpha\, b \\ \alpha\, c \end{bmatrix}$$

Os principais comandos do pacote LinearAlgebra relacionados com vetores são:

> *CrossProduct*, *Dimension*, *DotProduct*, *IsOrthogonal*, *MatrixVectorMultiply*, *Norm*, *Normalize*, *UnitVector*, *VectorAngle*, *Zip*.

Vamos ver exemplos de alguns deles. O produto escalar é calculado com o comando *DotProduct*.

```
> DotProduct(v1,v2);
```

$$\overline{(a)}\, x + \overline{(b)}\, x^2 + \overline{(c)}\, x^3$$

Se *v1* for um vetor coluna, o produto escalar será calculado usando o complexo conjugado de *v1*, como é o caso acima. Se *v1* for um vetor linha, o produto escalar será calculado usando o complexo conjugado de *v2*.

```
> DotProduct(v1a,v2);
```

$$\overline{(x)}\, a + \overline{(x)}^2\, b + \overline{(x)}^3\, c$$

Note que em qualquer caso não importa se *v2* seja vetor linha ou vetor coluna. Para evitar o uso de complexo conjugado deve-se usar a opção *conjugate=false*.

```
> DotProduct(v1,v2,conjugate=false);
```

$$x\, a + x^2\, b + x^3\, c$$

O produto exterior ou produto vetorial é calculado com o comando *CrossProduct*.

```
> v3 := CrossProduct(v1,v2);
```

$$v3 := \begin{bmatrix} b\, x^3 - c\, x^2 \\ c\, x - a\, x^3 \\ a\, x^2 - b\, x \end{bmatrix}$$

Por definição, o vetor *v3* é ortogonal a *v1* e *v2*, como podemos facilmente verificar.

```
> simplify(DotProduct(v1,v3,conjugate=false));
```

$$0$$

```
> simplify(DotProduct(v2,v3,conjugate=false));
```

$$0$$

Ou de forma análoga, podemos verificar que o ângulo entre *v1* e *v3* é $\frac{\pi}{2}$.

```
>  simplify(VectorAngle(v1,v3,conjugate=false));
```

$$\frac{1}{2}\pi$$

O módulo de um vetor é calculado pelo comando *Norm* com a opção *Euclidean*.

```
>  Norm(v1,Euclidean);
```

$$\sqrt{|a|^2+|b|^2+|c|^2}$$

O valor *default* da segunda opção é *infinity*. Neste caso a norma é definida da seguinte maneira:

```
>  Norm(v1,infinity);
```

$$\max(|a|\,,\,|c|\,,\,|b|)$$

ou seja, o resultado é o valor em módulo da maior componente.

Um vetor pode ser normalizado com o comando *Normalize*. Como exemplo, vamos definir um vetor não normalizado e em seguida normalizá-lo.

```
>  v4 := < 1, 2, 3 >;
```

$$v4 := \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

```
>  Normalize(v4,Euclidean);
```

$$\begin{bmatrix} \frac{1}{14}\sqrt{14} \\ \frac{1}{7}\sqrt{14} \\ \frac{3}{14}\sqrt{14} \end{bmatrix}$$

Naturalmente, o módulo do último resultado deve ser 1.

```
>  Norm(%,Euclidean);
```

$$1$$

Note que a opção *Euclidean* deve aparecer explicitamente em todos os cálculos envolvendo módulos de vetores.

Os comandos para calcular gradiente, divergente, rotacional e laplaciano infelizmente não foram transpostos para o pacote *LinearAlgebra* (pelo menos não na versão 6). Nesse caso é necessário usar o antigo pacote *linalg.* A primeiro tarefa é carregar o pacote.

```
> with(linalg):
```

Os comandos em questão são *grad, diverge, curl* e *laplacian.* Estes comandos devem ter no mínimo dois argumentos, onde o primeiro é uma função, ou melhor, uma expressão que depende de certas variáveis, e o segundo uma lista de variáveis que representam as coordenadas. O sistema de coordenadas *default* é o sistema cartesiano. Vamos dar uma apelido para a expressão f(x, y, z), e calcular o gradiente, divergente o laplaciano desta função.

```
> alias(f=f(x,y,z));
```

$$f$$

```
> v:=[x,y,z]; # lista das coordenadas
```

$$v := [x,\, y,\, z]$$

```
> grad(f,v);
```

$$\left[\frac{\partial}{\partial x}\,f,\ \frac{\partial}{\partial y}\,f,\ \frac{\partial}{\partial z}\,f\right]$$

```
> diverge(%,v);
```

$$(\frac{\partial^2}{\partial x^2}\,f) + (\frac{\partial^2}{\partial y^2}\,f) + (\frac{\partial^2}{\partial z^2}\,f)$$

```
> laplacian(f,v);
```

$$(\frac{\partial^2}{\partial x^2}\,f) + (\frac{\partial^2}{\partial y^2}\,f) + (\frac{\partial^2}{\partial z^2}\,f)$$

O rotacional deve ser aplicado a uma função vetorial. Assim, vamos dar apelidos para g(x, y, z) e h(x, y, g).

```
> alias(g=g(x,y,z),h=h(x,y,z));
```

$$f,\, g,\, h$$

```
> curl([f,g,h],v);
```

$$\left[(\frac{\partial}{\partial y}\,h) - (\frac{\partial}{\partial z}\,g),\ (\frac{\partial}{\partial z}\,f) - (\frac{\partial}{\partial x}\,h),\ (\frac{\partial}{\partial x}\,g) - (\frac{\partial}{\partial y}\,f)\right]$$

Podemos confirmar que o divergente do rotacional é zero:

```
> diverge(%,v);
```

$$0$$

e, da mesma forma, confirmar que o rotacional do gradiente é o vetor nulo:

```
> curl(grad(f,v), v);
```

$$[0, 0, 0]$$

Todas estas operações podem ser feitas em sistemas de coordenadas não-cartesianos. Vamos ver um exemplo de cálculo de gradiente no sistema de coordenadas esféricas:

```
> f1 := r^2*sin(theta)*cos(phi);
```

$$f1 := r^2 \sin(\theta) \cos(\phi)$$

```
> v:= [r, theta, phi];
```

$$v := [r, \theta, \phi]$$

```
> grad(f1, v, coords=spherical);
```

$$[2\, r \sin(\theta) \cos(\phi),\ r \cos(\theta) \cos(\phi),\ -r \sin(\phi)]$$

Além de coordenadas cartesianas, esféricas e cilíndricas que são as mais utilizadas, o Maple conhece mais de 40 sistemas de coordenadas em 2 e 3 dimensões. Para ter acesso a lista completa destes sistemas, pedimos ajuda da forma *?coords*.

Chapter 5

Gráficos

5.1 Introdução

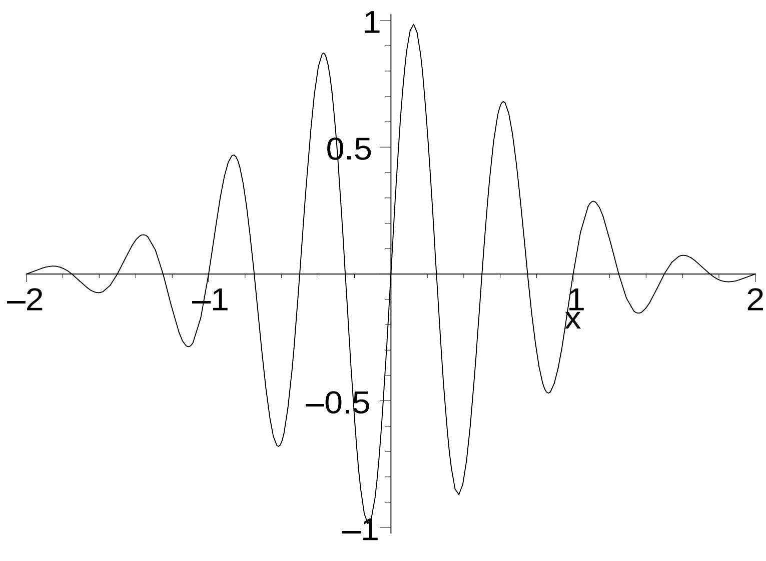
Vamos ver alguns exemplos de gráficos. Considere a seguinte expressão como uma função de x.

```
> F := exp(-x^2)*sin(4*Pi*x);
```

$$F := e^{(-x^2)} \sin(4\,\pi\,x)$$

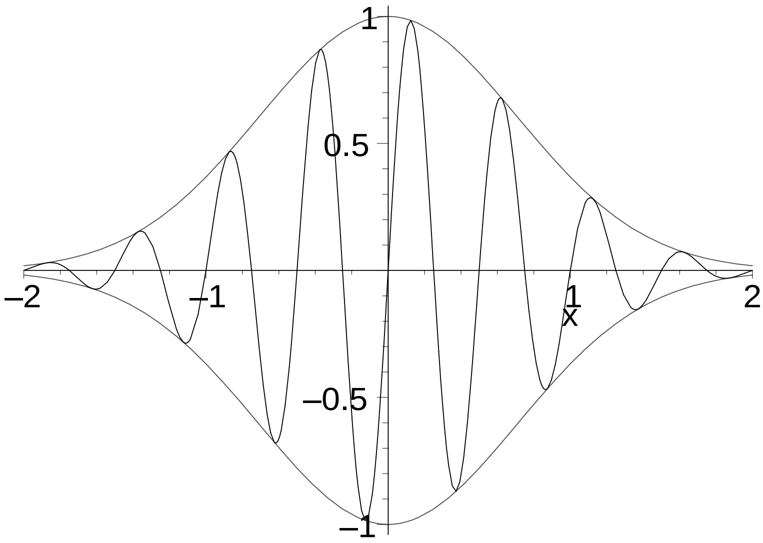
O seu gráfico no intervalo $[-2\,..\,2]$ é

```
> plot(F, x=-2..2);
```



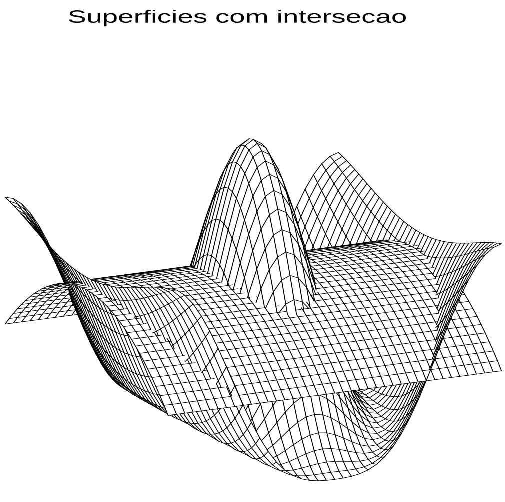
Vamos agora fazer o gráfico de F junto com as envoltórias. A curva F em preto e as envoltórias em vermelho. Devemos colocar as funções em uma lista (colchetes quadrados) com as cores correspondentes na opção *color*.

```
> plot( [F, exp(-x^2), -exp(-x^2)], x=-2..2, color=[black,red,red]);
```



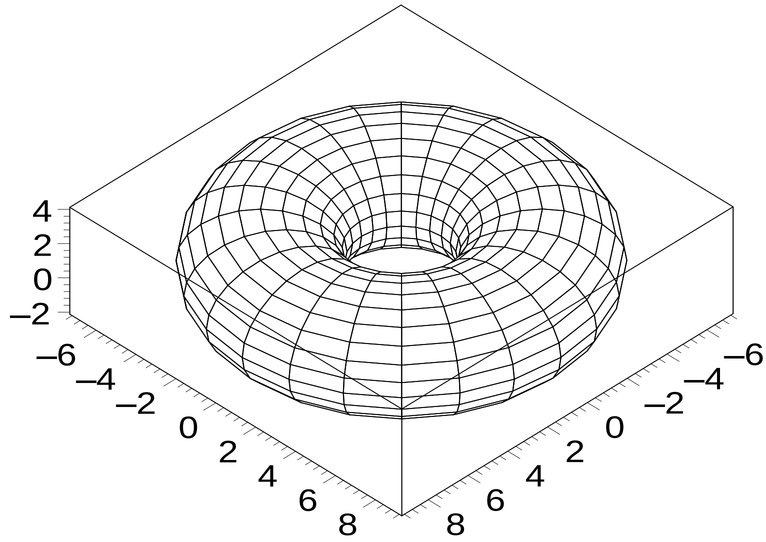
Vamos fazer o gráfico de duas superfícies que têm interseção. Note que usamos o comando *plot3d* no caso de gráficos em 3 dimensões e *plot* no caso de 2 dimensões.

```
> plot3d({cos(sqrt(x^2+3*y^2))/(1+y^2/8), 2/15-(2*x^2+x^2)/50},
> x=-3..3, y=-3..3, grid=[41,41], orientation=[-26,71],
> title=`Superficies com intersecao`);
```



Os comandos seguintes fazem o gráfico de um toro cujo centro está na posição [1, 1, 1] com raio 5 e com raio de meridiano 3. Nesse caso vamos usar um comando especial do pacote *plottools*.

```
> T := plottools[torus]([1,1,1], 3, 5):
> plots[display](T, scaling=constrained, style=HIDDEN,
> axes=boxed, color=black);
```



Além do pacote *plottools*, vamos explorar o pacote *plots*, que contem os principais comandos para fazer gráficos em 2 ou 3 dimensões..

5.2 Gráficos em 2 Dimensões

5.2.1 Introdução

A sintaxe para gerar gráficos em 2 dimensões é:

plot(f(x), x = a .. b, opções)

onde ***f(x)*** é uma função de uma variável e ***a..b*** é o intervalo de variação da variável ***x***. As ***opções*** são da forma:

nome da opção = tipo da opção

A lista completa dos nomes das opções e seus tipo pode ser obtida através do *help on line* com o comando *?*plot,options.

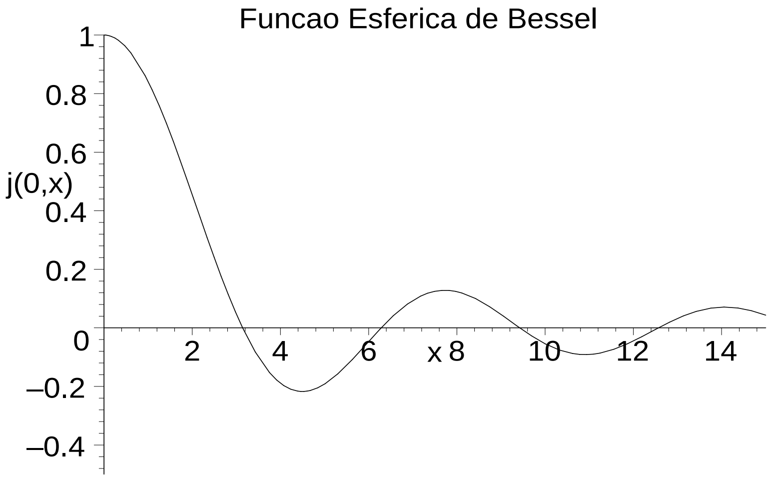
As opções servem para se obter os efeitos desejados em cada gráfico. Vamos ver alguns exemplos com vários tipos de opções. Primeiro, vamos definir a função esférica de Bessel a partir da função cilíndrica de Bessel:

```
> j := (n,x) -> sqrt(Pi/(2*x))*BesselJ(n+1/2,x);
```

$$j := (n,\, x) \rightarrow \sqrt{\frac{1}{2}\,\frac{\pi}{x}}\,\mathrm{BesselJ}(n + \frac{1}{2},\, x)$$

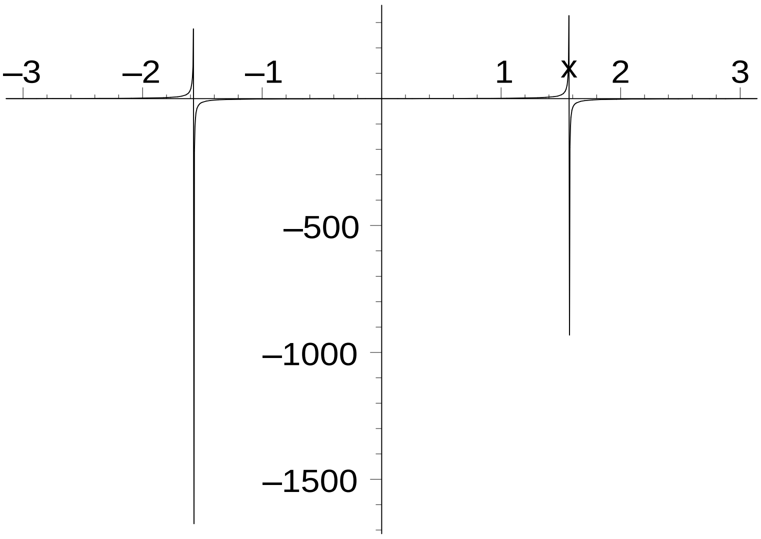
Agora, vamos fazer o gráfico da função de ordem zero com o título "Função Esférica de Bessel", com o nome "j(0,x)" no eixo vertical e controlando o número de marcações no eixo x e no eixo y:

```
> plot(j(0,x), x=0..15, `j(0,x)`=-0.5..1, xtickmarks=4,
> ytickmarks=6, title=`Funcao Esferica de Bessel`);
```



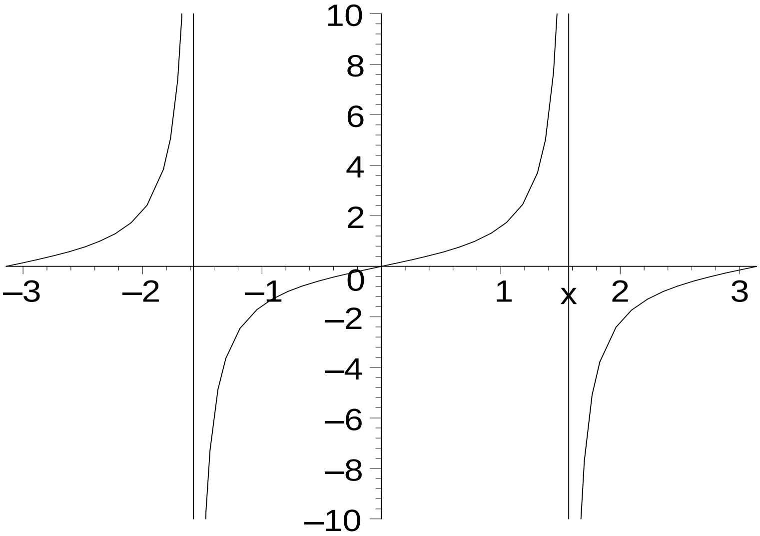
Em várias situações, as opções são necessárias, pois sem elas, o gr áfico pode ficar ilegível. Por exemplo, o gráfico da função tangente sem opções:

```
> plot( tan(x), x=-Pi..Pi);
```



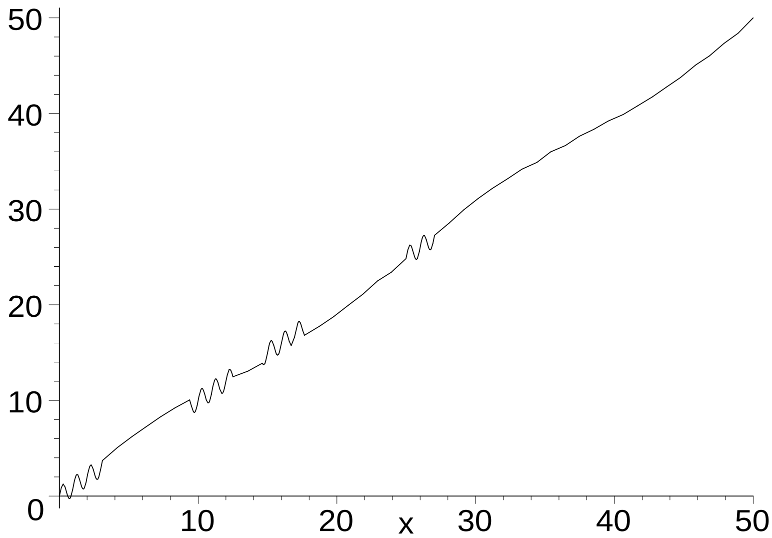
Aqui, precisamos da opção que limita o intevalo do eixo vertical:

```
> plot( tan(x), x=-Pi..Pi, -10..10);
```



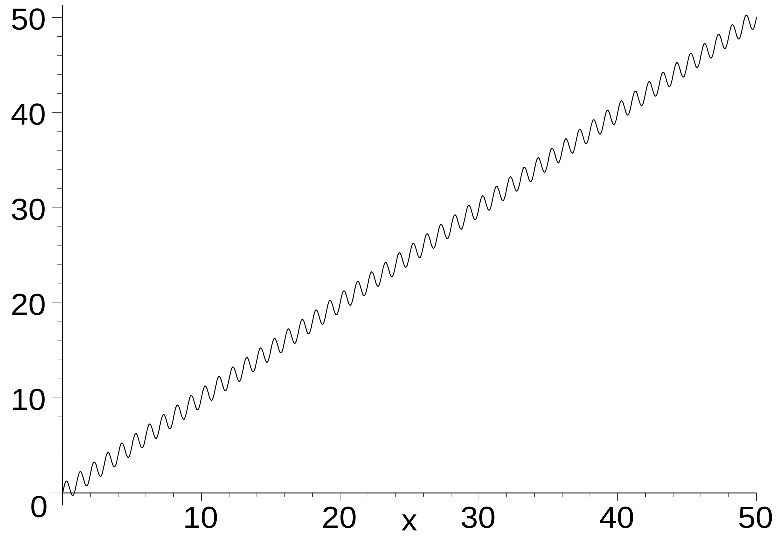
Vejamos outro exemplo. O gráfico da função a função $x + \sin(2\,\pi\,x)$ deve apresentar 50 máximos no intervalor $x = 0$ .. 50. Porém, vejamos o resultado:

```
> plot(x+sin(2*Pi*x),x=0..50);
```



Este é um caso em que a opção *numpoints* é necessária. O número de pontos *default* (50) não foi suficiente para este caso:

```
> plot(x+sin(2*Pi*x),x=0..50,numpoints=1000);
```



Agora, vamos ver outros tipos de gráficos em 2 dimensões.

5.2.2 Gráficos de Funções Parametrizadas

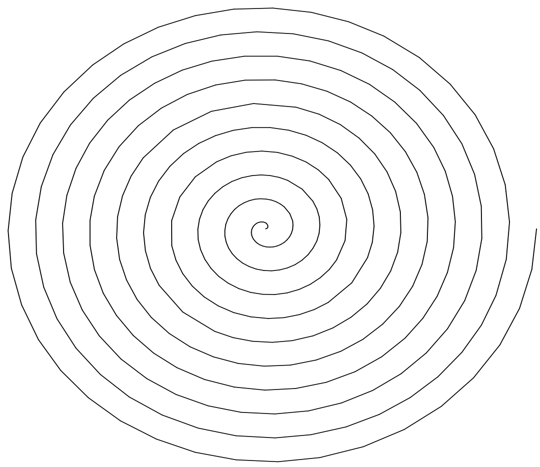
A sintaxe dos gráficos de funções parametrizadas é:

plot([x(t), y(t), t = a .. b], opções)

onde as opções são da mesma forma do comando *plot* usual. A lista completa pode ser obtida com o comando *?plot,options*.

Como exemplo, vamos fazer um espiral:

```
> plot([t*cos(2*Pi*t), t*sin(2*Pi*t), t=0..10],
> scaling=constrained, axes=none);
```



Note que sem a opção *scaling=constrained* figura teria um formato elítico.

5.2.3 Gráficos em Coordenadas Polares

A sintaxe dos gráficos de funções em coordenadas polares é:

polarplot(r(theta), theta = a .. b, opções)

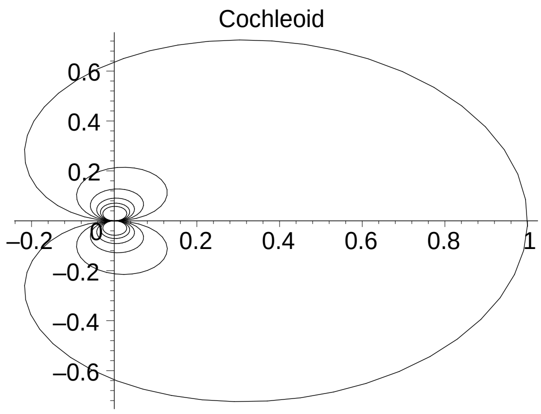
onde as opções são da mesma forma do comando *plot* usual. A lista completa pode ser obtida com o comando *?plot,options*.

É possível também fazer gráficos em coordenadas polares parametrizados. A sintaxe é:

polarplot([r(x), theta(x), x = a .. b], opções)

Como exemplo, vamor fazer o gráfico da figura conhecida por *Cocheloid*.

```
> plot( [sin(t)/t, t, t=-6*Pi..6*Pi], coords=polar,
> title=‘Cochleoid‘);
```



5.2.4 Gráficos de Funções Contínuas por Partes

As funções contínuas por partes são definidas com o comando *piecewise*. As funções definidas desta forma podem ser diferenciadas e integradas. Por exemplo, vamos definir uma função cujo gráfico tem a forma de uma barreira. A função é descrita por:

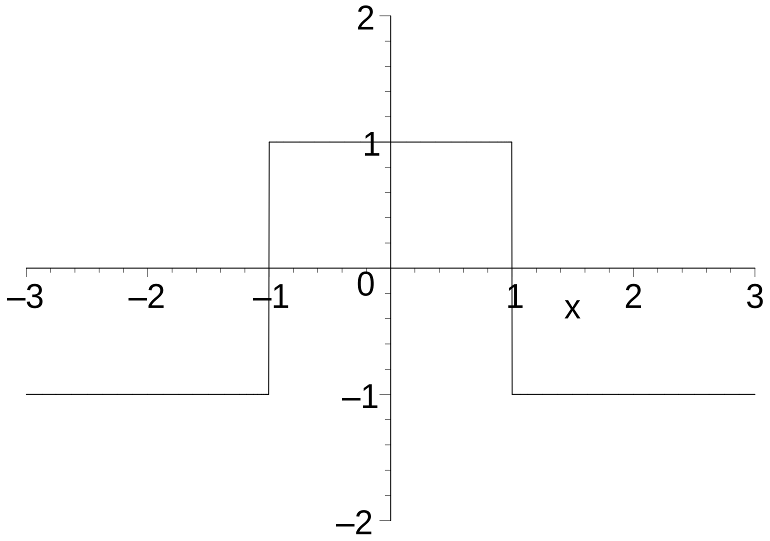
$$\mathrm{f}(x) = \begin{cases} -1 & x < -1 \\ 1 & x \leq 1 \\ -1 & 1 < x \end{cases}$$

Assim, podemos usar o comando *piecewise* da seguinte forma:

```
> F := piecewise( x<-1,-1,x<=1,1,-1);
```

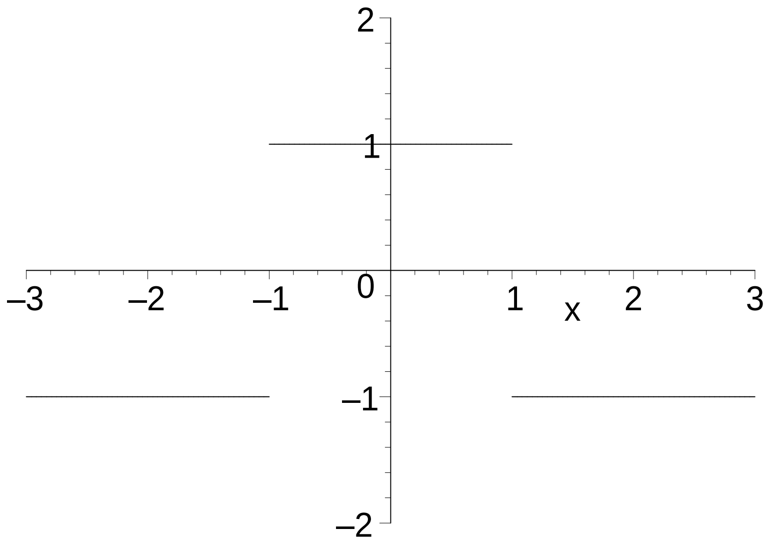
$$F := \begin{cases} -1 & x < -1 \\ 1 & x \leq 1 \\ -1 & \mathit{otherwise} \end{cases}$$

```
> plot( F, x=-3..3, -2..2, scaling=constrained);
```



Observe que as descontinuidades do gráfico foram unidas por retas verticais. Estas retas podem ser eliminadas com a opção *discont=true*:

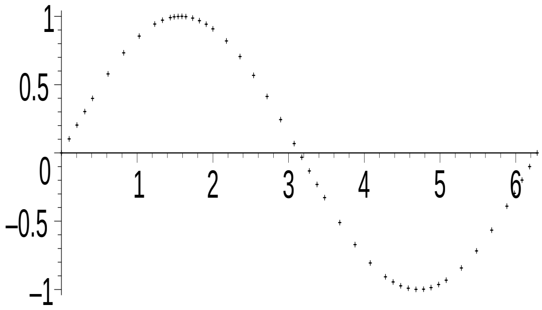
```
> plot( F, x=-3..3, -2..2, scaling=constrained,
> discont=true, color=black);
```



5.2.5 Gráficos de Pontos Isolados

Podemos fazer gráficos de pontos isolados com a opção *style=point*:

```
> plot( sin, 0..2*Pi, scaling=constrained, style=point,
> numpoints=5);
```



Quando atribuimos um gráfico a uma variável, podemos ver quais pontos o Maple escolheu para gerar o gráfico em questão.

```
> sin_plot := %;
```

$$
\begin{aligned}
&\mathit{sin\_plot} := \mathit{INTERFACE\_PLOT}(\mathit{CURVES}([[0., 0.],\\
&[.102716668893423791, .102536141786977023],\\
&[.205433337786847582, .203991404898633294],\\
&[.308150006680271372, .303296304678287720],\\
&[.410866675573695161, .399404024397164936],\\
&[.616300013360542742, .578019850839916250],\\
&\qquad\qquad\vdots\\
&[5.88278669634446150, -.389785456253492146],\\
&[5.98288634591334656, -.295805802057080624],\\
&[6.08298599548223074, -.198864665580834348],\\
&[6.18308564505111490, -.999325803838139876e-1],\\
&[6.28318529461999998, -.125595865048264204e-7]]),\\
&\mathit{STYLE}(\mathit{POINT}), \mathit{SCALING}(\mathit{CONSTRAINED}), \mathit{COLOUR}(\mathit{RGB}, 0, 0, 0),\\
&\mathit{AXESLABELS}("", ""), \mathit{VIEW}(0. .. 6.283185308, \mathit{DEFAULT}))
\end{aligned}
$$

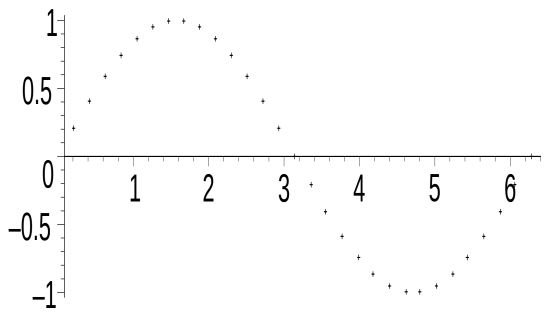
Estes pontos não são igualmente espaçados, pois nas regiões onde a curvatura do gráfico é maior, o Maple escolhe mais pontos, como podemos verificar no gráfico anterior. Vamos fazer os mesmo gráfico, porém vamos nós mesmos escolher os pontos de forma que fiquem igualmente espaçados. Vamos gerar estes pontos com o comando *seq*:

```
> sin_points := seq(evalf([2*Pi*i/30,sin(2*Pi*i/30)],3),
> i=1..30);
```

$$
\begin{aligned}
&\mathit{sin\_points} := [.209, .207], [.418, .406], [.628, .588], [.838, .743], [1.05, .865], [1.26, .952],\\
&[1.47, .995], [1.67, .995], [1.88, .952], [2.09, .865], [2.30, .743], [2.51, .588], [2.72, .406],\\
&[2.93, .207], [3.14, 0.], [3.36, -.207], [3.55, -.406], [3.77, -.588], [3.99, -.743],\\
&[4.18, -.865], [4.40, -.952], [4.62, -.995], [4.80, -.995], [5.02, -.952], [5.24, -.865],\\
&[5.43, -.743], [5.65, -.588], [5.87, -.406], [6.06, -.207], [6.28, 0.]
\end{aligned}
$$

Vamos colocar a sequência de pontos em uma lista, e fazer o gráfico com a opção *style=points*:

```
> plot([sin_points], scaling=constrained, style=point);
```

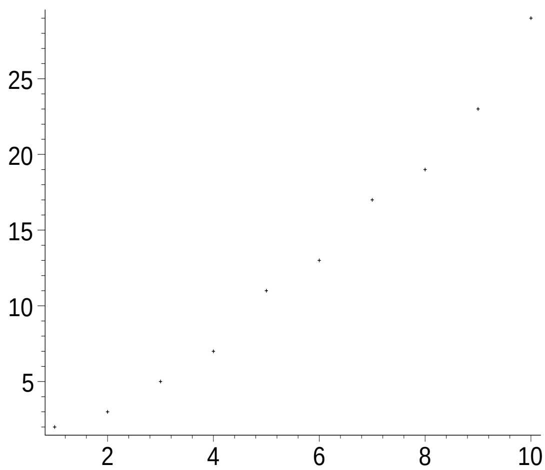


Vejamos outro exemplo:

```
> pontos := seq([ i, ithprime(i) ],i =1..10);
```

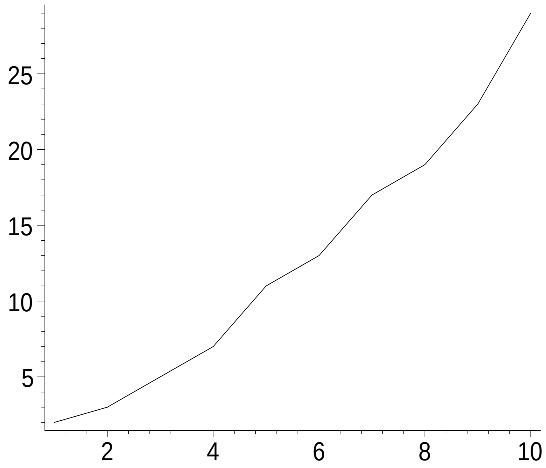
pontos := [1, 2], [2, 3], [3, 5], [4, 7], [5, 11], [6, 13], [7, 17], [8, 19], [9, 23], [10, 29]

```
> plot( [pontos], style=point);
```



Sem a opção *style=point*, os pontos são unidos por retas:

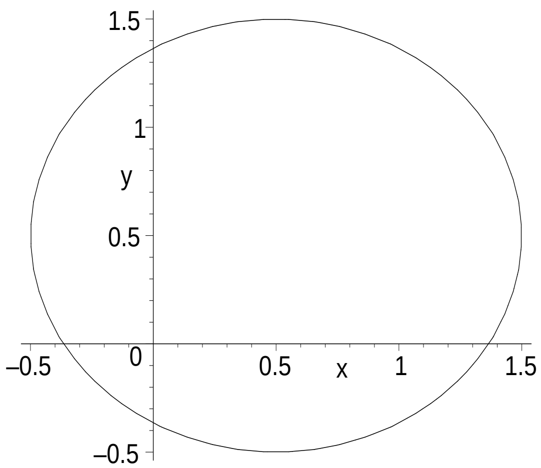
```
> plot( [pontos]);
```



5.2.6 Gráficos de Funções Definidas Implicitamente

Vejamos um exemplo:

```
> with(plots):
> implicitplot(x^2-x+1/4+y^2-y+1/4=1,x=-1..2,y=-1..2,
> grid=[30,30], color=black);
```



5.3 Gráficos em 3 Dimensões

5.3.1 Introdução

A sintaxe dos gráficos em 3 dimensões é:

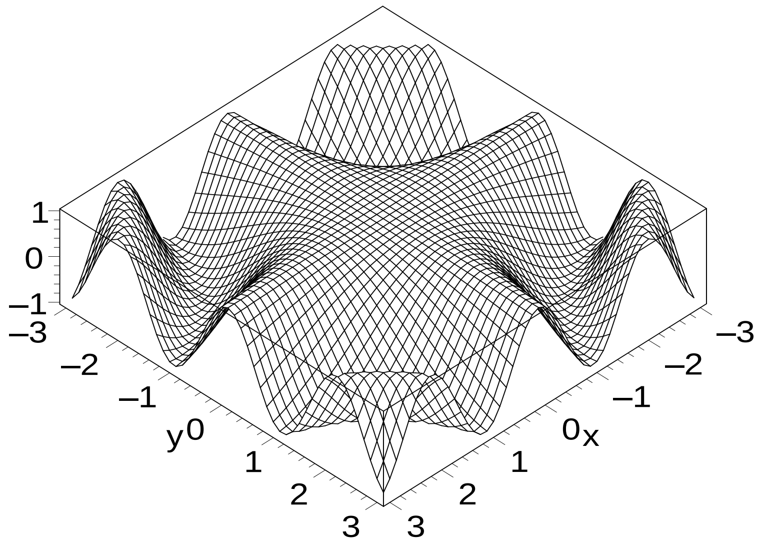
$$\boldsymbol{plot3d(\ f(x,y),\ x = a\ ..\ b,\ y = c\ ..\ d,\ opções)}$$

onde ***f(x,y)*** é uma função de duas variáveis, ***a .. b*** é o intervalo para o eixo ***x*** e ***c .. d*** é o intervalo para o eixo ***y***. As ***opções*** são da forma

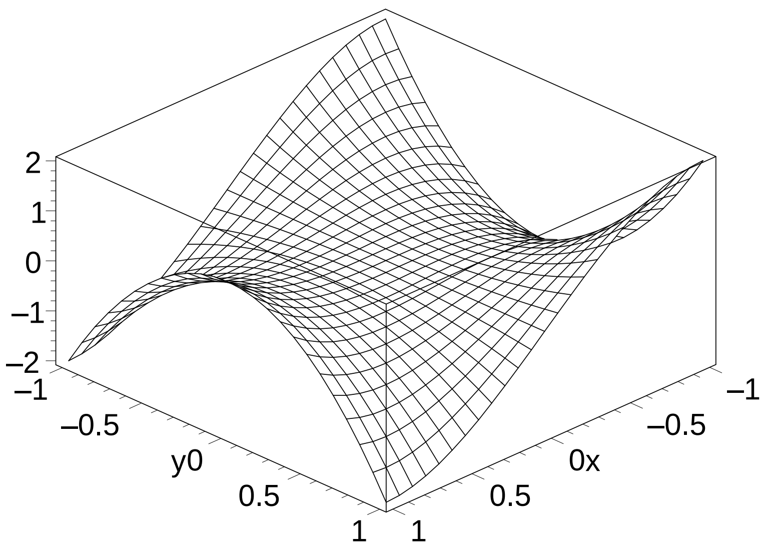
$$\boldsymbol{nome\ da\ opção = tipo\ da\ opção}$$

A lista completa dos nomes das opções e seus tipo pode ser obtida através do *help on line* com o comando *?plot3d,options*. Vejamos alguns exemplos:

```
> plot3d( cos(x*y), x=-3..3, y=-3..3, grid=[49,49],
> axes=boxed, scaling=constrained, shading=zhue);
```




```
> plot3d( x^3-3*x*y^2, x=-1..1, y=-1..1, axes=boxed);
```



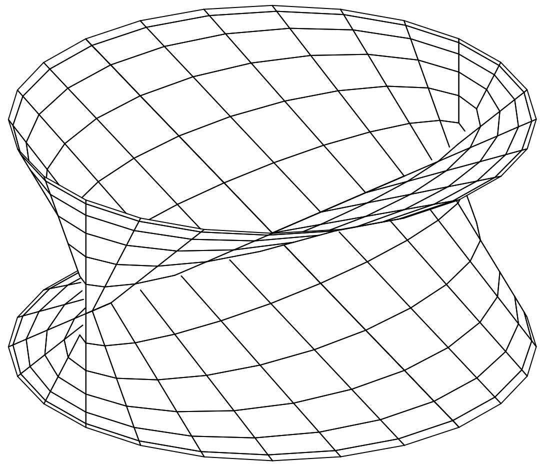
5.3.2 Gráficos de Funções Parametrizadas

A sintaxe dos gráficos de funções parametrizadas em 3 dimensões é:

plot3d([x(t,s), y(t,s), z(t,s)], t = a .. b, s = c .. d, opções)

onde as opções são da mesma forma do comando *plot3d* usual. A lista completa pode ser obtida com o comando *?plot3d,options*. Note que a lista não inclui o intervalo das variáveis t e s, como ocorre no comando equivalente em 2 dimensões. Vamos ver um exemplo:

```
>  plot3d([ sin(t), cos(t)*sin(s), sin(s) ], t=-Pi..Pi,
>  s=-Pi..Pi);
```



5.3.3 Gráficos em Coordenadas Esféricas

A sintaxe dos gráficos de funções em coordenadas esféricas é:

sphereplot(r(theta,phi), theta = a .. b, phi = c .. d, opções)

É possível também fazer gráficos de funções em coordenadas esféricas parametrizadas. A sintaxe é:

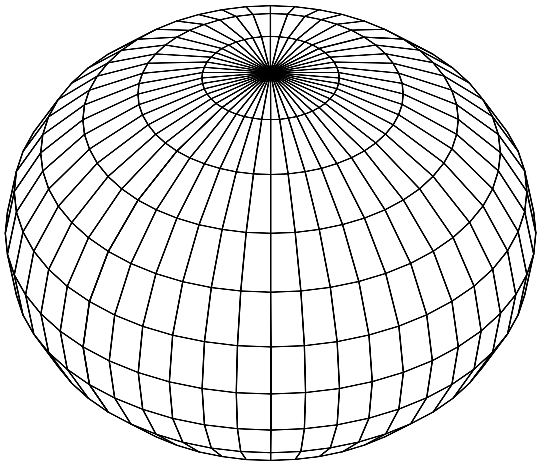
sphereplot([r(t,s), theta(t,s), phi(t,s)], t = a .. b, s = c .. d, opções)

O comando *sphereplot* está dentro do pacote *plots*. Vamos ver um exemplo do primeiro caso:

```
>  with(plots):
>  sphereplot(1, theta=0..Pi, phi=0..2*Pi,
>  scaling=constrained);
```

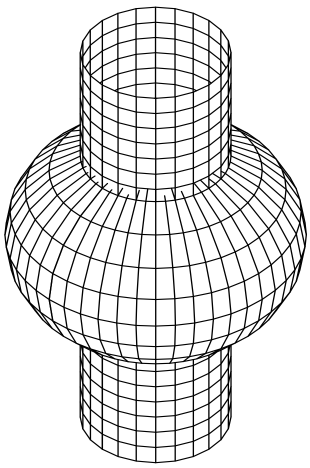
5.4 Exibindo vários Gráficos Simultaneamente

Para exibir dois ou mais gráficos simultaneamente, podemos usar o comando *plot* ou *plot3d* colocando as expressões de cada gráfico dentro de uma lista, isto é, entre colchetes, ou em um conjunto, isto é, entre chaves. Vimos um exemplo no início deste capítulo. Desta forma, todas a opções usadas serão comuns a todos os gráficos. Para fazer gráficos com opções diferentes e mostrá-los na mesma tela, temos que tomar outro caminho. Cada gráfico deve ser feito independentemente e atribuído a uma variável. Este comando de atribuição deve ser terminado com dois pontos ":". Para mostrar os



gráficos juntos, devemos usar o comando *display* ou *display3d*, dependendo se forem gráficos em 2 ou 3 dimensões. As variáveis devem ser colocadas dentro de uma lista ou um conjunto. Estes comandos estão dentro do pacote *plots*. Vejamos um exemplo:

```
> with(plots):
> G1 := sphereplot(1, theta=0..Pi, phi=0..2*Pi):
> G2 := cylinderplot(1/2, theta=0..2*Pi, z=-2..2):
> display([G1,G2], scaling=constrained);
```

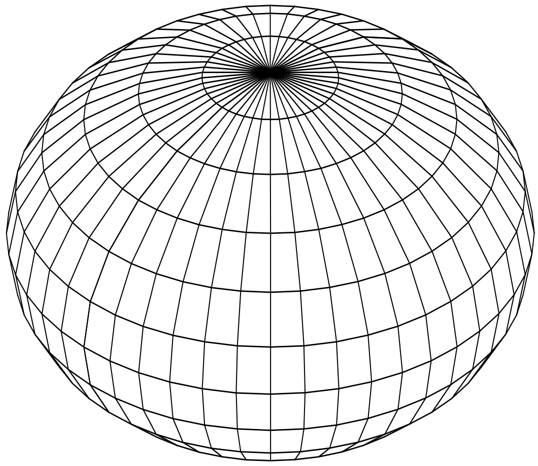


Para ver um dos gráficos isoladamente, basta avaliar a variável onde o gráfico foi guardado. Por exemplo, para ver a esfera:

```
> G1;
```

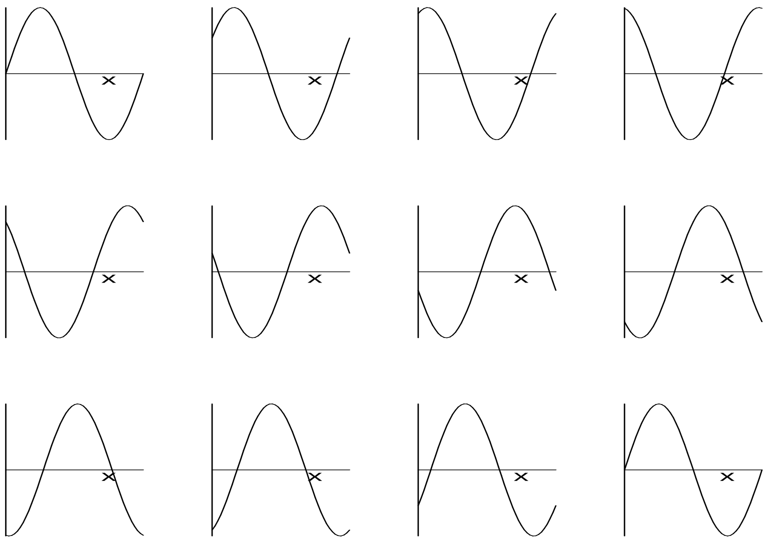
5.5 Animando Gráficos em duas ou três Dimensões

Além de permitir fazer gráficos numa mesma tela simultaneamente, o Maple permite que vários gráficos sejam exibidos em sequência, gerando o efeito de animação de gráficos. Para fazer animação de gráficos em coordenadas cartesianas com todas as



opções em comum, podemos usar os comandos *animate* e *animate3d*. Vejamos um exemplo:

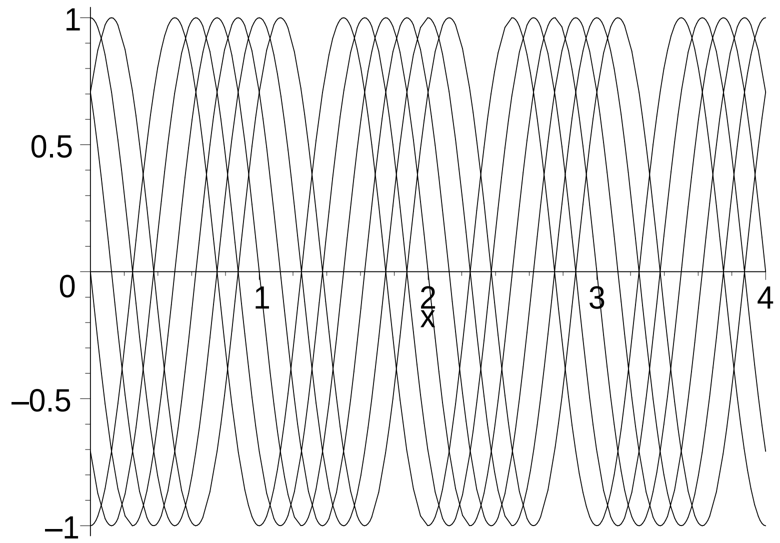
```
> with(plots):
> animate( sin(2*Pi*(x+t)),x=0..1,t=0..1,frames=12);
```



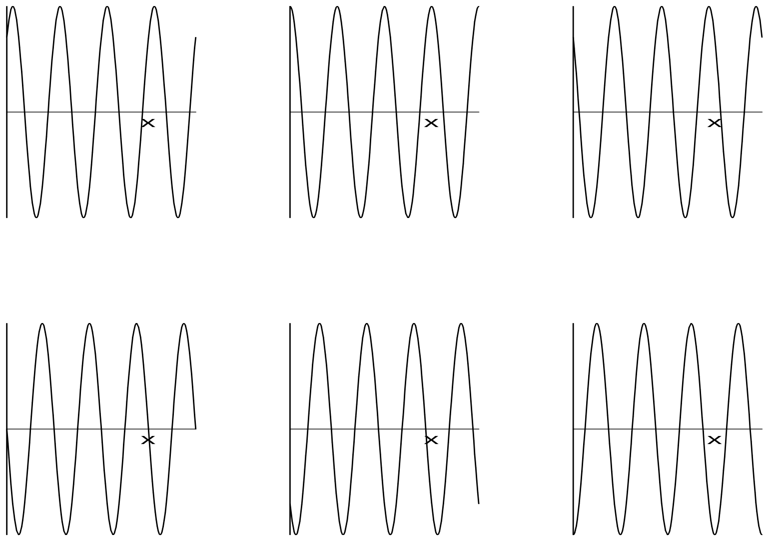
Para disparar a animação devemos clicar no gráfico e depois na tecla *play*. O comando *animate* gera tantos gráficos quanto for o valor da opção *frames*. Os valores que a variável t assume, depende do valor de *frames*. No caso acima, t assumiu os valores $t = 0$, $t = \frac{1}{11}$ e $t = \frac{10}{11}$ e $t = 1$. Observe que t assumiu 12 valores, sendo que o valor $t = 0$ e $t = 1$ geram o mesmo gráfico. Isso faz com que a animação tenha sempre uma rápida parada quando um ciclo se completa. A maneira de evitar isto, é tomar o intervalo para t de 0 até $\frac{11}{12}$.

Para fazer a animação de outros tipos de gráficos sem ser em coordenadas cartesianas, temos que usar um método mais elaborado. Cada gráfico deve ser feito independentemente e atribuído a uma variável. Cada comando de atribuição deve ser terminado com dois pontos ":". Para mostrar os gráficos em sequência, devemos usar o comando *display* ou *display3d*, dependendo se forem gráficos em 2 ou 3 dimensões, com a opção *insequence=true*. O comando *for* é muito útil neste contexto, pois podemos gerar vários gr áficos com um único comando. A título de exemplo, vamos fazer a mesma animação anterior sem usar o comando *animate*:

```
> for i from 1 to 6 do
> P[i] := plot(sin(2*Pi*(x+i/8)),x=-0..4)
> end do:
> display({seq(P[i],i=1..6)});
> # para ver os gráficos simultaneamente
```



```
> display( [seq(P[i],i=1..6)], insequence=true);
> # para ver em sequência
```



5.6 Colocando Textos em Gráficos

Os comandos usuais de gerar gráficos colocam uma série de textos no gr áfico. Este é o caso do nome das coordenadas, do título do gráfico entre outros. Podemos escolher as fontes através das opções *font, titlefont, axesfont* e *labelfont*. Porém, para modificar a posição destes textos, é necessário proceder de outra forma. Devemos usar os comandos textplot e textplot3d, que têm as seguintes sintaxe:

textplot([coord-x, coord-y, ‘texto‘])

e

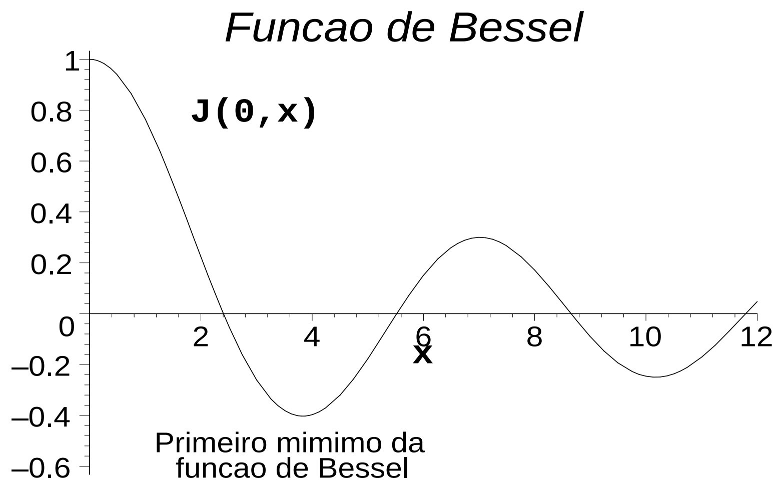
textplot3d([coord-x, coord-y, coord-z, ‘texto‘])

onde *coord-x* é um número especificando a posição x do centro do texto. O mesmo com relação a *coord-y* e *coord-z*. Estes comandos geram um gráfico com o texto desejado na posição especificada. Com o comando *display* ou *display3d* juntamos estes gráficos-textos com o gráfico das funções em questão.

Vejamos um exemplo de composição de vários gráficos-textos. Cada gr áfico será atribuido a uma variável e os comandos serão terminados com dois-pontos para evitar que informações desnecessárias sejam mostradas na tela. Somente o comando *display* será terminado com ponto-e-vírgula:

```
> G1 := textplot([3.83,-0.5,‘Primeiro mimimo da‘]):
> G2 := textplot([3.83,-0.6,‘funcao de Bessel‘]):
> G3 := textplot([3.0,0.8,‘J(0,x)‘],font=[COURIER,BOLD,12]):
```

```
> G4 := plot(BesselJ(0,x), x=0..12, labelfont=[COURIER,BOLD,12],
> title=`Funcao de Bessel`, titlefont=[HELVETICA,OBLIQUE,15]):
> display({G1,G2,G3,G4});
```



Para exibir letras gregas é necessário usar a fonte *symbol*.

5.7 Imprimindo Gráficos

Para imprimir gráficos no formato *PostScript*, temos que modificar o valor da variável *plotdevice* para *ps*, e direcionar a saida para um arquivo. Isto deve ser feito com o comando *interface*:

```
> interface( plotdevice = ps, plotoutput = "c:/tmp/graf1.ps");
> plot( sin, -2..2);
```

O gráfico do último comando foi enviado para o arquivo *graf1.ps*, e pode ser impresso numa impressora *PostScript*. Uma maneira alternativa de enviar gráficos para um arquivo é usando o comando *plotsetup*. Vamos mostrar mais um exemplo, mas nesse caso vamos tirar a caixa retangular que o Maple coloca no gráfico enviado para o arquivo, e vamos apresentá-lo na orientação usual (*portrait* em vez de *landscape* que é o *default*)

```
> plotsetup(ps, plotoutput = "c:/tmp/graf1.ps",
> plotoptions = `portrait,noborder`);
```

Para voltar a mostrar o gráfico na *worksheet* novamente, o valor de *plotdevice* deve ser modificado para *inline* ou *win*:

```
> interface( plotdevice = inline);
```

Para imprimir gráficos em outros formatos, veja o *help on line* de plot,device e interface.

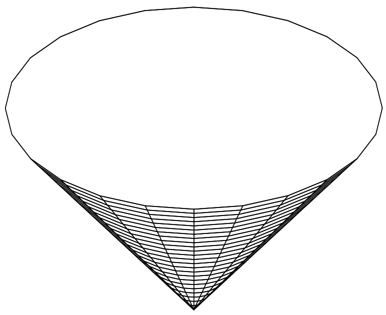
5.8 Manipulando Gráficos

O pacote *plottools* tem vários comandos para manipulação com gráficos além de diversos comandos para criar objetos gráficos. Estes comandos não mostram o gráfico. O comando *display* do pacote *plots* deve ser usado em conjunto. Por exemplo, podemos criar um cone com a ponta na origem, de raio 1/2 e altura 2 com o comando *cone*:

```
> with(plottools);
```

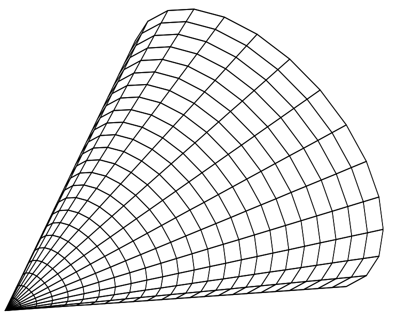
[*arc, arrow, circle, cone, cuboid, curve, cutin, cutout, cylinder, disk, dodecahedron, ellipse, ellipticArc, hemisphere, hexahedron, homothety, hyperbola, icosahedron, line, octahedron, pieslice, point, polygon, project, rectangle, reflect, rotate, scale, semitorus, sphere, stellate, tetrahedron, torus, transform, translate, vrml*]

```
> C := cone([0,0,0], 1/2, 2):
> with(plots):
> display( C );
```



Agora, vamos girar este cone de $\frac{\pi}{2}$ em relação ao eixo y:

```
> display(rotate( C, 0, Pi/2, 0));
```



Chapter 6

Cálculo Diferencial e Integral

6.1 Funções Matemáticas

6.1.1 Definindo uma Função

O Maple tem uma grande variedade de funções definidas internamente, incluindo as funções trigonométricas, hiperbólicas, elíticas, hipergeométricas, de Bessel e muitas outras. A lista completa pode ser obtida através do *help on line* com o comando *?inifcn*. Todos os comandos do Maple, como o comando de integração, de diferenciação, de limite, etc., estão aptos a reconhecer estas funções e fornecer o resultado conveniente.

Por maior que seja a lista das funções definidas internamente pelo Maple, um usuário necessita definir suas próprias funções, por exemplo, generalizando funções já existentes. Estas funções definidas pelo usuário serão incluidas na lista das funções do Maple no sentido de que elas também serão reconhecidas pelos comandos de diferenciação, integração, etc., como funções válidas, mas não serão reinicializadas automaticamente caso o usuário saia do Maple ou dê o comando *restart*.

Para definir uma função no Maple, devemos usar o operador seta (->, isto é, sinal de menos seguido do sinal de maior). Por exemplo, a função f(x) = $\frac{x^2-1}{x^2+1}$ é definida da seguinte maneira.

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

$$f := x \to \frac{x^2-1}{x^2+1}$$

Podemos avaliar essa função em um ponto ou encontrar sua derivada.

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

$$0$$

```
> simplify( diff(f(x), x) );
```

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

A título de precaução, vamos avisar o seguinte. Não defina uma função da forma

```
> g(x) := x^2;
```

$$\mathrm{g}(x) := x^2$$

Na forma acima, o máximo que o Maple sabe é que o valor da função g no ponto x é x^2. Nada mais. O valor de g no ponto 2? O Maple não sabe!

```
> g(2);
```

$$\mathrm{g}(2)$$

Para a maior parte dos usuários, o que foi dito acima é suficiente para se trabalhar com funções no Maple, e portanto o leitor pode pular para a próxima seção. Caso o usuário enfrente algum problema, ele deve ler o resto dessa seção que tem um grau de dificuldade acima da média desta apostila. Primeiramente, é importante distinguir função de expressão algébrica. Vejamos primeiro uma expressão algébrica que não é uma função.

```
> expr := sin(a*x);
```

$$expr := \sin(a\,x)$$

No comando acima, $\sin(a\,x)$ é uma expressão que foi atribuida a variável *f*. Não podemos considerar f como uma função de x, pois não obteríamos o resultado desejado com o comando $f(\pi)$:

```
> expr(pi);
```

$$\sin(a\,x)(\pi)$$

O resultado desejado era $\sin(a\,\pi)$. Para obter o valor da função $\sin(a\,x)$ em $x = \pi$, temos que usar o comando *subs*:

```
> subs(x=Pi, expr);
```

$$\sin(a\,\pi)$$

Para definir uma verdadeira função (e não uma expressão algébrica) podemos usar o comando unapply:

```
> F := unapply(sin(a*x),x);
```

$$F := x \to \sin(a\,x)$$

Neste caso especificamos que a expressão $\sin(a\,x)$ é uma função de x e a é um parâmetro. Até agora, o Maple não tinha condições de saber se queríamos uma função na variável x tendo a como parâmetro ou vice-versa. Vejamos o que ocorre com esta nova definição:

```
> F(Pi);
```

$$\sin(a\,\pi)$$

```
> F(y);
```

$$\sin(a\,y)$$

```
> F(x);
```

$$\sin(a\,x)$$

Assim podemos obter o valor da função em um ponto da forma $F(ponto)$. Podemos dizer que F é o nome da função $x \to \sin(a\,x)$ enquanto que $\sin(a\,x)$ é uma expressão algébrica. Vejamos:

```
> eval(F);
```

$$x \to \sin(a\,x)$$

Exitem alguns comandos no Maple que têm funções como argumento. Eles não aceitam uma expressão algébrica como argumento. Este é o caso do operador D, que só atua em funções dando como resultado outras funções (a derivada). Por exemplo:

```
> D(F);
```

$$x \to \cos(a\,x)\,a$$

O comando acima está com a sintaxe correta. O comando abaixo está com a sintaxe errada.

```
> D(expr);
```

$$\mathrm{D}(\sin(a\,x))$$

$\mathrm{D}(\sin(a\,x))$ tem a sintaxe errada, pois o operador D está sendo aplicado a uma expressão. Vamos comparar com o operador *diff*. Este atua em expressões algébricas e também retorna uma expressão algébrica. Por exemplo:

```
> diff(F(x), x); # Note que usamos F(x) e nao F.
```

$$\cos(a\,x)\,a$$

Outro comando perfeitamente válido.

```
> diff(expr, x);
```

$$\cos(a\,x)\,a$$

Como vimos acima, uma outra maneira de definir uma função é usar diretamente o operador seta (->). Por exemplo:

```
> g := x -> x^2 + 1;
```

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

```
> g(2);
```

$$5$$

Esta forma não é necessariamente equivalente ao comando *unapply*.

ATENÇÃO: Quando usar o operador seta (->), a expressão da função deve ser dada explicitamente.

Como consequência da chamada de atenção acima podemos afirmar que: "não podemos usar % dentro do operador seta". Veja a seguinte sequência de comandos problemática.

```
> x^2;
```

$$x^2$$

```
> f := x -> % + 1;
```

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

```
> f(2);

Error, (in f) invalid terms in sum
```

Como aplicação destes conceitos, vamos contruir a função que dá a soma do MDC com o MMC de dois números. Por exemplo, consideremos os números 25 e 15. O MDC é 5 e o MMC é 75, de forma que a função que queremos definir deve retornar 80, pois

```
> gcd(25,15) + lcm(25,15);
```

$$80$$

Vamos construir duas versões, uma usando o comando *unapply* e outra usando o operador seta. Neste exemplo, veremos que é mais conveniente usar o operador seta. Primeiro, vamos ver as dificuldades que aparecem no uso do *unapply*. A primeira tentativa será:

```
> F := unapply(gcd(n1,n2)+lcm(n1,n2), n1,n2);
```

$$F := (n1,\, n2) \to 1 + n1\; n2$$

```
> F(25,15);
```

$$376$$

Obtivemos o resultado errado. A função foi definida de maneira incorreta e o motivo foi que o comando *unapply* avaliou os seus argumentos antes de criar a função. A avaliação de $\gcd(n1,\, n2) + \mathrm{lcm}(n1,\, n2)$ é $1 + n1\; n2$, como podemos verificar:.

```
> gcd(n1,n2) + lcm(n1,n2);
```

$$1 + n1\; n2$$

A maneira correta de definir esta função é:

```
> F := unapply('gcd(n1,n2)+lcm(n1,n2)', n1,n2);
```

$$F := (n1,\, n2) \to \gcd(n1,\, n2) + \mathrm{lcm}(n1,\, n2)$$

```
> F(25,15);
```

$$80$$

A versão com o operador seta não sofre deste problema, pois seus argumentos não são avaliados:

```
> F := (n1,n2) -> gcd(n1,n2) + lcm(n1,n2);
```

$$F := (n1,\ n2) \rightarrow \mathrm{gcd}(n1,\ n2) + \mathrm{lcm}(n1,\ n2)$$

```
> F(25,15);
```

$$80$$

Um erro muito comum é tentar definir uma uma função da seguinte forma $\mathrm{g}(x) := x^2$, como alertamos no início da seção. O comando

```
> g(x) := x^2;
```

$$\mathrm{g}(x) := x^2$$

faz com o Maple guarde na memória (através da *remember table*) que o valor da função no pont x é x^2. Porém, a função $\mathrm{g}(x) = x^2$ para um x genérico não foi criada, em vez disso, o Maple criou o seguinte procedimento extremamente rebuscado que é

```
> eval(g);
```

$$x \rightarrow x^2 + 1$$

Para os curiosos o que podemos dizer é: a opção *remember* associa uma tabela de recordação à função g. A variável especial *procname* se refere ao nome da função, que no caso é g. *args* é uma variável especial que se refere aos argumento da função numa chamada real. Note que *procname*(*args*) está entre *plics*, isso evita uma recursão infinita. Finalmente, traduzindo para o Português, a função g, que a rigor é um procedimento, não faz nada, porém tem um valor guadado na sua tabela de recordação. Para imprimir a tabela de recordação

```
> print(op(4,eval(g)));
```

$$\mathrm{table}([x = x^2])$$

6.1.2 Álgebra e Composição de Funções (@ e @@)

Podemos somar, multiplicar e compor duas ou mais funções. Por exemplo, se temos uma equação e queremos subtrair o lado direito do lado esquerdo podemos usar a soma das funções *lhs* e *rhs*:

```
> equacao := 2*y*x + x - 1 = 2*x-5;
```

$$\mathit{equacao} := 2\,y\,x + x - 1 = 2\,x - 5$$

```
> (lhs - rhs)(equacao);
```

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

O último comando é equivalente a

```
> lhs(equacao) - rhs(equacao);
```

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

porém tem um sentido diferente. No primeiro caso, fizemos uma subtração de funções antes de aplicar ao argumento. No segundo caso fizemos uma subtração usual de expressões algébricas.

Vejamos outros exemplos. Vamos elevar a função seno ao quadrado e depois aplicá-lo ao ponto π.

```
> (sin^2)(Pi/4); # potenciação
```

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

O próximo exemplo parece misterioso.

```
> (1/sin + (x->x^2)*cos)(x); # adição e produto
```

$$\frac{1}{\sin(x)} + x^2\cos(x)$$

Note que usamos uma função anônima no meio. Outra forma equivalente, porém mais clara é fazer em 2 etapas.

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

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

```
> (1/sin + F*cos)(x);
```

$$\frac{1}{\sin(x)} + x^2\cos(x)$$

Note que *cos* e *sin* são nomes de funções pré-definidas do Maple assim como F é o nome da função $x \to x^2$. Vejamos agora como fazer composição de funções dentro das álgebra de funções.

```
> (g1@g2)(x); # composição
```

$$\mathrm{g1}(\mathrm{g2}(x))$$

```
> (g3@g3-g3@@2)(x);
```

$$0$$

O símbolo @ é o operador de composição de funções enquanto que $F@@n$ quer dizer $F@F@F...$ n vezes. Isso explica porque o último resultado é zero. O primeiro resultado, por sua vez, deu o que é esperado, pois composição de funcões é equivalente a aplicação sucessivas das funções. Vejamos outros exemplos:

```
> F := x -> a^x;
```

$$F := x \to a^x$$

```
> (F@@5)(x);
```

$$a^{\left(a^{\left(a^{\left(a^{(a^x)}\right)}\right)}\right)}$$

```
> G := x -> 1/(1+x);
```

$$G := x \to \frac{1}{1+x}$$

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

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

```
> evalf(subs(x=1,%));
```

$$.6250000000$$

```
> evalf((G@@100)(1)); # razão aurea (1+sqrt(5))/2
```

$$.6180339887$$

Se o operador @@ for usado com um número negativo, o Maple entende que a função inversa deve ser usada:

```
> (cos@@(-1))(0) = arccos(0);
```

$$\frac{1}{2}\,\pi = \frac{1}{2}\,\pi$$

6.2 Integral

A integral indefinida de uma expressão pode ser obtida da seguinte forma:

```
> Int(x/(x^3+1), x); # Forma inerte
```

$$\int \frac{x}{x^3+1}\,dx$$

```
> value(%);
```

$$-\frac{1}{3}\ln(1+x)+\frac{1}{6}\ln(x^2-x+1)+\frac{1}{3}\sqrt{3}\arctan(\frac{1}{3}\,(2\,x-1)\,\sqrt{3})$$

Podemos confirmar o resultado da seguinte forma:

```
> normal(diff(%,x), expanded);
```

$$\frac{x}{x^3+1}$$

Na integral definida, os limites de integração devem ser especificados da seguinte forma:

```
> int(1/x^2, x=-1..1);
```

$$\infty$$

Podemos ver que o Maple reconhece que há uma descontinuidade na função $\frac{1}{x^2}$ dentro dos limites de integração e ele calcula corretamente a integral.

O método usado para calcular uma integral definida é exato. Para realizar uma integração numérica aproximada, usamos a forma inerte do comando *int* e depois *evalf*:

```
> Int(exp(-2*t)*t*ln(t), t=0..infinity);
```

$$\int_0^\infty e^{(-2\,t)}\,t\ln(t)\,dt$$

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

$$-.06759071137$$

O Maple não resolve automaticamente a seguinte integral.

```
> Int1 := int(1/(3/2*(z^2-1)^(1/2)-3/2*arccos(1/z))^2*(z^2-1)^(1/2)/z,z);
```

$$Int1 := \int \frac{\sqrt{z^2-1}}{(\frac{3}{2}\sqrt{z^2-1}-\frac{3}{2}\arccos(\frac{1}{z}))^2\,z}\,dz$$

Isto não quer dizer que seja o fim da linha. Vamos fazer a substituição de variáveis $z \rightarrow \zeta$ usando a relação:

```
> new_var := (z^2-1)^(1/2)-arccos(1/z)=2/3*zeta^(3/2);
```

$$new\_var := \sqrt{z^2-1}-\arccos(\frac{1}{z}) = \frac{2}{3}\,\zeta^{(3/2)}$$

Vamos usar o comando *changevar* do pacote *student*.

```
> 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*]

```
> res := changevar(new_var, Int1, zeta);
```

$$res := -\frac{2}{3}\,\frac{1}{\zeta^{(3/2)}}$$

O resultado final na variável z é

```
> final_res := powsubs(3/2*rhs(new_var)=3/2*lhs(new_var),res);
```

$$\mathit{final\_res} := -4\,\frac{1}{9\,\sqrt{z^2-1}-9\arccos(\frac{1}{z})}$$

6.3 Séries

O comando para expandir em séries é *series*. Por exemplo

```
> series(exp(x), x);
```

$$1+x+\frac{1}{2}\,x^2+\frac{1}{6}\,x^3+\frac{1}{24}\,x^4+\frac{1}{120}\,x^5+\mathrm{O}(x^6)$$

O primeiro argumento deve ser a função a ser expandida, o segundo é variável. O terceiro argumento controla a ordem da expansão

```
> series(exp(x), x, 10);
```

$$1+x+\frac{1}{2}\,x^2+\frac{1}{6}\,x^3+\frac{1}{24}\,x^4+\frac{1}{120}\,x^5+\frac{1}{720}\,x^6+\frac{1}{5040}\,x^7+\frac{1}{40320}\,x^8+\frac{1}{362880}\,x^9+\mathrm{O}(x^{10})$$

Se o terceiro argumento estiver ausento, o Maple usa o valor da vari ável global *Digits*, que por *default* é 6. O resultado do comando *series* é uma estrutura especial que deve ser manipulada pelo próprio comando *series*. Por exemplo

```
> coseno := series(cos(x),x);
```

$$\mathit{coseno} := 1-\frac{1}{2}\,x^2+\frac{1}{24}\,x^4+\mathrm{O}(x^6)$$

```
> seno := series(sin(x),x);
```

$$\mathit{seno} := x-\frac{1}{6}\,x^3+\frac{1}{120}\,x^5+\mathrm{O}(x^6)$$

O comando *expand* nada pode fazer com essas expressões.

```
> expand(coseno^2 + seno^2);
```

$$(1-\frac{1}{2}\,x^2+\frac{1}{24}\,x^4+\mathrm{O}(x^6))^2+(x-\frac{1}{6}\,x^3+\frac{1}{120}\,x^5+\mathrm{O}(x^6))^2$$

A forma correta é usar o próprio comando *series*.

```
> series(coseno^2 + seno^2, x);
```

$$1+\mathrm{O}(x^6)$$

Outro caminho possível é transformar as expressões em polinômios da seguinte forma

```
> convert(coseno,polynom);
```

$$1-\frac{1}{2}\,x^2+\frac{1}{24}\,x^4$$

Note que o term O(x^6) foi eliminado. Agora os termos de ordem 6 ou superior não serão eliminados quando essa expressão for manipulada a menos que voltemos a usar o comando *series*.

A série de Taylor para funções de várias variáveis é obtida com o comando *mtaylor*.

```
> mtaylor(sin(x^2+y^2), [x,y], 8);
```

$$x^2 + y^2 - \frac{1}{6}x^6 - \frac{1}{2}y^2x^4 - \frac{1}{2}y^4x^2 - \frac{1}{6}y^6$$

Nesse caso o resultado já é do tipo polinomial.

6.4 Séries de Fourier e transformadas

A série de Fourier de uma função f(x) é definida como

$$\mathrm{f}(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty}(a_n\cos(n\,x) + b_n\sin(n\,x))$$

onde

$$a_n = \frac{1}{\pi}\int_{-\pi}^{\pi}\mathrm{f}(x)\cos(n\,x)\,dx$$

$$b_n = \frac{1}{\pi}\int_{-\pi}^{\pi}\mathrm{f}(x)\sin(n\,x)\,dx$$

O Maple não tem nenhum comando pré-definido para calcular séries de Fourier. Um programa para esse fim, baseado nas fórmulas acima, é

```
> FourierSeries := proc (expr, x)
> local a0, a, b, n, k;
> assume(n,integer);
> a0 := normal(1/Pi*int(expr,x = -Pi .. Pi));
> a := normal(1/Pi*int(expr*cos(n*x),x = -Pi .. Pi));
> b := normal(1/Pi*int(expr*sin(n*x),x = -Pi .. Pi));
> return eval(subs(n = k,1/2*a0+Sum(a*cos(k*x)+b*sin(k*x),k = 1 ..
> infinity)))
> end proc;
```

Os comandos *proc*, *local* e *return* são comandos especiais para procedimentos. O comando *proc* começa um bloco que se fecha no comando *end proc*. O comando *local* define variáveis locais ao procedimento de forma a não interferir com as variáveis de mesmo nome utilizadas pelo usuário. Finalmente, o comando *return* retorna o valor do procedimento. Vejamos um exemplo.

```
> FourierSeries(x^2, x);
```

$$\frac{1}{3}\pi^2 + \left(\sum_{k=1}^{\infty}(4\,\frac{(-1)^k\cos(k\,x)}{k^2})\right)$$

As transformadas de funções são calculada após carregar o pacote *inttrans* (*int*egral *trans*forms).

```
> with(inttrans);
```

[*addtable, fourier, fouriercos, fouriersin, hankel, hilbert, invfourier, invhilbert, invlaplace, invmellin, laplace, mellin, savetable*]

Por exemplo, a transformada de Laplace de f(t) = $\sin(k\,t)$ é

```
> laplace(sin(k*t), t, s);
```

$$\frac{k}{s^2+k^2}$$

e a transformada inversa é

```
> simplify(invlaplace(%,s,t));
```

$$\sin(k\,t)$$

Chapter 7

Equações diferenciais ordinárias

7.1 Introdução

O comando *dsolve* é o principal comando para resolver analiticamente equações diferenciais ordinárias. As páginas do *help on line* estão bem escritas e muito do que segue abaixo está diretamente baseado nelas. A sintaxe do comando *dsolve* é

dsolve(*EDO*)
dsolve(*EDO*, y(x), *extra_args*)

onde

EDO é uma equação diferencial ordinária
y(x) é uma função indeterminada de um única variável
extra_args é um argumento opcional que depende do tipo de problema a ser resolvido

Para resolver uma equação diferencial ordinária com condições iniciais a sintaxe é

dsolve({*EDO*, *CI's*}, y(x),

onde *CI's* são as condicões iniciais. Para resolver um sistema de equações diferenciais ordinárias

dsolve({*seq_de_EDOs, CI's*}, {y(x), z(x), ...}, *extra_args*)

onde

{*seq_de_EDOs*} é um conjunto de EDO's
{y(x), z(x), ...} é um conjunto de funções indeterminadas de uma única variável

Por exemplo, a equação diferencial ordinária não linear de segunda ordem

```
> EDO := x^4*diff(y(x),x,x) + (x*diff(y(x),x) - y(x))^3 = 0;
```

$$EDO := x^4\,(\frac{\partial^2}{\partial x^2}\,\mathrm{y}(x)) + (x\,(\frac{\partial}{\partial x}\,\mathrm{y}(x)) - \mathrm{y}(x))^3 = 0$$

é integrada explicitamente com o simples comando

```
> dsolve(EDO);
```

$$\mathrm{y}(x) = (-\arctan(\frac{1}{\sqrt{-1 + \_C1\, x^2}}) + \_C2)\, x,\ \mathrm{y}(x) = (\arctan(\frac{1}{\sqrt{-1 + \_C1\, x^2}}) + \_C2)\, x$$

Observe a presença das duas constantes arbitrárias *_C1* e *_C2* em cada solução. Essa constantes são determinadas pelas condições iniciais. Por exemplo, se as condições iniciais são

```
> CI := y(1) = 1.3, D(y)(1) = 3.4;
```

$$CI := \mathrm{y}(1) = 1.3,\ \mathrm{D}(y)(1) = 3.4$$

então a sintaxe nesse caso é

```
> dsolve( {EDO,CI}, y(x));
```

$$\mathrm{y}(x) = \left(-\arctan\left(\frac{1}{\sqrt{-1 + \frac{541}{441}\, x^2}}\right) + \arctan(\frac{21}{10}) + \frac{13}{10}\right) x$$

O sistema não linear de EDO's

```
> sistema := {diff(f(x),x) = g(x),
> diff(g(x),x) = -exp(f(x)),
> diff(h(x),x,x) = g(x)/f(x)};
```

$$sistema := \{\frac{\partial}{\partial x}\,\mathrm{f}(x) = \mathrm{g}(x),\ \frac{\partial}{\partial x}\,\mathrm{g}(x) = -e^{\mathrm{f}(x)},\ \frac{\partial^2}{\partial x^2}\,\mathrm{h}(x) = \frac{\mathrm{g}(x)}{\mathrm{f}(x)}\}$$

é resolvido da seguinte forma

```
> dsolve(sistema, {f(x),g(x),h(x)});
```

$$\left[\{\mathrm{f}(x) = \ln(\frac{1}{2}\_C3 - \frac{1}{2}\tanh(\frac{1}{2}\,x\,\sqrt{\_C3} + \frac{1}{2}\_C4\,\sqrt{\_C3})^2\,\_C3)\},\ \{\mathrm{g}(x) = \frac{\partial}{\partial x}\,\mathrm{f}(x)\},\right.$$
$$\left.\left\{\mathrm{h}(x) = \int\!\!\int \frac{\mathrm{g}(x)}{\mathrm{f}(x)}\,dx\,dx + \_C1\,x + \_C2\right\}\right]$$

Note que a solução explícita só foi dada para a função $\mathrm{f}(x)$, enquanto que $\mathrm{g}(x)$ é determinada a partir de $\mathrm{f}(x)$ ($\mathrm{h}(x)$ a partir de $\mathrm{f}(x)$ e $\mathrm{g}(x)$). Para obter a solução explícita de todas as funções, o parâmetro *explicit* deve ser repassado como terceiro argumendo do comando *dsolve*.

A solução de uma EDO de ordem n (ordem da derivada mais alta) é dita geral se ela possui n constantes arbitrárias. Se a EDO for linear, a solução geral fornece todas as soluções possíveis. Se a EDO for não-linear, podem existir soluções especiais (ditas singulares) que não são obtidas da solução geral para quaisquer valores finitos das constantes arbitrárias.

7.2 Método de Classificação

Uma das primeiras tentativas do comando *dsolve* é determinar se a EDO tem uma forma já classificada de acordo com os livros da área, em especial os livros Kamke [6], Murphy [7] e Zwillinger [8]. As EDO's de primeira ordem estão classificadas no Maple como (ver *?odeadvisor,types*)

Abel, *Abel2A*, *Abel2C*, *Bernoulli*, *Chini*, *Clairaut*, *dAlembert*, *exact*, *homogeneous*, *homogeneousB*, *homogeneousC*, *homogeneousD*, *homogeneousG*, *linear*, *patterns*, *quadrature*, *rational*, *Riccati*, *separable*

As EDO's de segunda ordem (ou de outra ordem qualquer) estão classificadas como

Bessel, *Duffing*, *ellipsoidal*, *elliptic*, *Emden*, *erf*, *exact_linear*, *exact_nonlinear*, *Gegenbauer*, *Halm*, *Hermite*, *Jacobi*, *Lagerstrom*, *Laguerre*, *Lienard*, *Liouville*, *linear_ODEs*, *missing*, *Painleve*, *quadrature*, *reducible*, *Titchmarsh*, *Van_der_Pol*

A classificação de uma EDO é feita com o comando *odeadvisor* do pacote *DEtools*.

```
> with(DEtools,odeadvisor);
```

$$[odeadvisor]$$

7.2.1 EDO's de ordem 1

Uma EDO de primeira ordem é dita *separable* se ela tem a forma

$$\frac{\partial}{\partial x}\,\mathrm{y}(x) = \mathrm{f}(x)\,\mathrm{g}(\mathrm{y}(x))$$

A solução geral é

$$\int \mathrm{f}(x)\,dx - \int^{\mathrm{y}(x)} \frac{1}{\mathrm{g}(\_a)}\,d\_a + \_C1 = 0$$

A EDO

```
> separable_EDO := exp(y(x)+sin(x))*diff(y(x),x)=1;
```

$$separable\_EDO := e^{(\mathrm{y}(x)+\sin(x))}\,(\frac{\partial}{\partial x}\,\mathrm{y}(x)) = 1$$

é separável. Podemos confirmar com o comando *odeadvisor*.

```
> odeadvisor(separable_EDO);
```

$$[\_separable]$$

A solução é

```
> dsolve(separable_EDO);
```

$$\mathrm{y}(x) = \ln(\int e^{(-\sin(x))}\,dx + \_C1)$$

A EDO de *Bernoulli* é da forma

$$\frac{\partial}{\partial x}\,\mathrm{y}(x) = A(x)\,\mathrm{y}(x) + B(x)\,\mathrm{y}(x)^c$$

onde $A(x)$ e $B(x)$ são funções de x e c é uma constante. Por exemplo, a EDO

```
> Bernoulli_EDO := diff(y(x),x)=x*y(x)+y(x)^(1/2);
```

$$\mathit{Bernoulli\_EDO} := \frac{\partial}{\partial x}\,\mathrm{y}(x) = x\,\mathrm{y}(x) + \sqrt{\mathrm{y}(x)}$$

é do tipo *Bernoulli*

```
> odeadvisor(Bernoulli_EDO);
```

$$[\_\mathit{Bernoulli}]$$

e tem a seguinte solução geral

```
> dsolve(Bernoulli_EDO);
```

$$\sqrt{\mathrm{y}(x)} - \frac{\frac{1}{2}\sqrt{\pi}\,\mathrm{erf}(\frac{1}{2}\,x) + \_C1}{e^{(-1/4\,x^2)}} = 0$$

7.2.2 EDO's de ordem 2 ou maior

Uma EDO é linear se ela é da forma,

$$A(x)\,\mathrm{y}(x) + B(x)\,\tfrac{\partial}{\partial x}\,\mathrm{y}(x) + C(x)\,\tfrac{\partial^2}{\partial x^2}\,\mathrm{y}(x) + \ D(x)\,\tfrac{\partial^3}{\partial x^3}\,\mathrm{y}(x) + ... \ = \ F(x)$$

onde $A(x)$, $B(x)$, $C(x)$, $D(x)$, ... são funções de x. Se $F(x) = 0$, a EDO é dita linear e homogênea. Se os coeficientes forem constantes, *dsolve* é capaz de achar a solução geral da equação homogênea. Por exemplo a EDO

```
> linear_EDO := diff(y(x),x$4) + a*diff(y(x),x$3) +
> b*diff(y(x),x$2) + c*diff(y(x),x)+d*y(x) = 0;
```

$$\mathit{linear\_EDO} := (\frac{\partial^4}{\partial x^4}\,\mathrm{y}(x)) + a\,(\frac{\partial^3}{\partial x^3}\,\mathrm{y}(x)) + b\,(\frac{\partial^2}{\partial x^2}\,\mathrm{y}(x)) + c\,(\frac{\partial}{\partial x}\,\mathrm{y}(x)) + d\,\mathrm{y}(x) = 0$$

tem a seguinte solução geral

```
> dsolve(linear_EDO);
```

$$\mathrm{y}(x) = \_C1\;e^{(\mathrm{RootOf}(\%1,\,\mathit{index}=1)\,x)} + \_C2\;e^{(\mathrm{RootOf}(\%1,\,\mathit{index}=2)\,x)} + \_C3\;e^{(\mathrm{RootOf}(\%1,\,\mathit{index}=3)\,x)}$$
$$+ \_C4\;e^{(\mathrm{RootOf}(\%1,\,\mathit{index}=4)\,x)}$$
$$\%1 := d + c\,\_Z + b\,\_Z^2 + a\,\_Z^3 + \_Z^4$$

Note que RootOf$(\cdots)$ representa as raízes de um polinômio de quarta ordem. Para se obter o resultado explícito, é necessário dar o comando *_EnvExplicit:=true* (veja *?solve*). Para EDO's de ordem maior que 4, em geral não é mais possível expandir as raízes de forma explícita. Se a equação não for homogênea, a solução geral é a solução da parte homogênea mais uma solução particular. Se os coeficientes dependem de x, a EDO não pode ser integrada de forma genérica (veja *?linear_ODEs* para mais detalhes).

Uma ODE é dita *missing* se ela tem uma das formas

$$\mathrm{F}\left(x,\ \tfrac{\partial}{\partial x}\,\mathrm{y}(x),\ \tfrac{\partial^2}{\partial x^2}\,\mathrm{y}(x),\ \tfrac{\partial^3}{\partial x^3}\,\mathrm{y}(x),\ \ldots\right) = 0$$

ou

$$\mathrm{F}\left(\mathrm{y}(x),\ \tfrac{\partial}{\partial x}\,\mathrm{y}(x),\ \tfrac{\partial^2}{\partial x^2}\,\mathrm{y}(x),\ \tfrac{\partial^3}{\partial x^3}\,\mathrm{y}(x),\ \ldots\right) = 0$$

No primeiro caso falta a função $\mathrm{y}(x)$ e na segunda a variável x. EDO's de ordem n do tipo *missing* podem sempre ser reduzidas para uma EDO de ordem $n-1$. Isso não garante a solução completa a menos que a EDO reduzida possa ser resolvida.

Por exemplo, considere a EDO de segunda ordem do tipo *missing* (sem $\mathrm{y}(x)$ e sem x) completamente geral.

```
> missing_x_and_y_EDO := F(diff(y(x),x),diff(y(x),x,x)) = 0;
```

$$\mathit{missing\_x\_and\_y\_EDO} := \mathrm{F}(\frac{\partial}{\partial x}\,\mathrm{y}(x),\ \frac{\partial^2}{\partial x^2}\,\mathrm{y}(x)) = 0$$

Observe que a classificação indica que a EDO náo é do tipo *missing*.

```
> odeadvisor(missing_x_and_y_EDO);
```

$$[[\_2nd\_order,\ \_missing\_x]]$$

Essa EDO pode ser completamente integrada.

```
> dsolve(missing_x_and_y_EDO);
```

$$\mathrm{y}(x) = \int \mathrm{RootOf}\left(x - \int^{\_Z} \frac{1}{\mathrm{RootOf}(\mathrm{F}(\_c,\ \_Z))}\,d\_c + \_C1\right)\,dx + \_C2$$

```
> EDO := diff(y(x),x,x) = y(x)^5;
```

$$\mathit{EDO} := \frac{\partial^2}{\partial x^2}\,\mathrm{y}(x) = \mathrm{y}(x)^5$$

A variável x não aparece explicitamente nessa EDO.

```
> odeadvisor(EDO);
```

$$[[\_2nd\_order,\ \_missing\_x],\ [\_2nd\_order,\ \_reducible,\ \_mu\_x\_y1]]$$

Para obter informações sobre o método de solução usamos o comando *infolevel*.

```
> infolevel[dsolve]:=2:
> res := dsolve(EDO);
```

```
Methods for second order ODEs:
Trying to isolate the derivative d^2y/dx^2...
Successful isolation of d^2y/dx^2
-> Trying classification methods
trying 2nd order Liouville
trying 2nd order, 2 integrating factors of the form mu(x,y)
trying differential order: 2; missing variables
   *** Sublevel 2 ***
   symmetry methods on request
   1st order, trying reduction of order with given symmetries:   [_a,3*_b]
```

$$\textit{trying way} = \textit{HINT}$$

```
   1st order, trying the canonical coordinates of the invariance group
      *** Sublevel 3 ***
      Methods for first order ODEs:
      Trying to isolate the derivative dY/dX...
      Successful isolation of dY/dX
      -> Trying classification methods
      trying a quadrature
      trying 1st order linear
      1st order linear successful
   1st order, canonical coordinates successful
differential order: 2; canonical coordinates successful
differential order 2; missing variables successful
```

$$res := \int^{y(x)} -3\,\frac{1}{\sqrt{3\,\_a^6 - 3\,\_C1}}\,d\_a - x - \_C2 = 0,$$

$$\int^{y(x)} 3\,\frac{1}{\sqrt{3\,\_a^6 - 3\,\_C1}}\,d\_a - x - \_C2 = 0$$

```
> infolevel[dsolve]:=1:
```

Observações:

1. O comando *infolevel*[*dsolve*] := 2 permite que o usuário acompanhe o método usado no processo de integração da EDO. No caso acima, o método *missing variables* reduz a EDO de segunda ordem para uma EDO de primeira ordem, que é resolvida com sucesso.
2. As soluções estão na forma implícita.
3. As integrais estão na forma inerte do comando *intat* (observe que a integral só tem o limite superior).

Podemos obter uma solução explícita se $\_C1 = 0$. O comando *value* é necessário para avaliar a integral inerte.

```
> sol := solve(value(subs(_C1=0,res[1])),y(x));
```

$$sol := \frac{\sqrt{2}\,\sqrt{(x + \_C2)\,\sqrt{3}}}{2\,x + 2\,\_C2},\ -\frac{\sqrt{2}\,\sqrt{(x + \_C2)\,\sqrt{3}}}{2\,x + 2\,\_C2}$$

Podemos confirmar as soluções.

```
> odetest(y(x)=sol[1],EDO), odetest(y(x)=sol[2],EDO);
```

$$0, 0$$

Muitas EDO's, que têm funções não elementares como solução, também estão classificadas. Por exemplo, as funções de *Bessel* $\mathrm{J}_\nu(x)$ e $\mathrm{Y}_\nu(x)$ obedecem a seguinte EDO.

```
> Bessel_EDO := x^2*diff(y(x),x,x) + x*diff(y(x),x) +
> (x^2-n^2)*y(x) = 0;
```

$$\mathit{Bessel\_EDO} := x^2\,(\frac{\partial^2}{\partial x^2}\,\mathrm{y}(x)) + x\,(\frac{\partial}{\partial x}\,\mathrm{y}(x)) + (x^2-n^2)\,\mathrm{y}(x) = 0$$

```
> dsolve(Bessel_EDO);
```

$$\mathrm{y}(x) = \_C1\,\mathrm{BesselJ}(n,\,x) + \_C2\,\mathrm{BesselY}(n,\,x)$$

A classificação é

```
> odeadvisor(Bessel_EDO);
```

$$[\_Bessel]$$

A solução de uma ODE não é necessarimente a função correspondente a classificação. A ODE

```
> erf_EDO := diff(y(x),x,x)+2*x*diff(y(x),x)-2*n*y(x) = 0;
```

$$\mathit{erf\_EDO} := (\frac{\partial^2}{\partial x^2}\,\mathrm{y}(x)) + 2\,x\,(\frac{\partial}{\partial x}\,\mathrm{y}(x)) - 2\,n\,\mathrm{y}(x) = 0$$

é classificada como

```
> odeadvisor(erf_EDO);
```

$$[\_\mathrm{erf}]$$

A função erro *erf* é definida pela integral

$$\mathrm{erf}(x) = \frac{2\int_0^x e^{(-t^2)}\,dt}{\sqrt{\pi}}$$

e a função erro complementar é definida por $\mathrm{erfc}(x) = 1 - \mathrm{erf}(x)$. As integrais iteradas da função erro é definida recursivamente por

$$\mathrm{erfc}(n,\,x) = \int_x^\infty \mathrm{erfc}(n-1,\,t)\,dt$$

onde $\mathrm{erfc}(0,\,x) = \mathrm{erfc}(x)$.

Podemos verificar que as integrais iteradas da função erro complementar é solução dessa ODE através do comando *odetest*.

```
> odetest(y(x)=erfc(n,x),erf_EDO);
```

$$0$$

A classificação nos diz que a função erro e correlatas obedecem a ODE *erf_ODE*, porém a solução é dada em termos das funções de *Whittaker*.

```
> dsolve(erf_EDO);
```

$$\mathrm{y}(x) = \frac{\_C1\ \mathrm{WhittakerM}(-\frac{1}{2}\,n - \frac{1}{4},\ \frac{1}{4},\ x^2)\, e^{(-1/2\,x^2)}}{\sqrt{x}} + \frac{\_C2\ \mathrm{WhittakerW}(-\frac{1}{2}\,n - \frac{1}{4},\ \frac{1}{4},\ x^2)\, e^{(-1/2\,x^2)}}{\sqrt{x}}$$

O comando *dsolve* não resolve algumas EDO's pois as funções que são soluções dessas equações não estão implementadas no Maple. Por exemplo

```
> Mathieu_EDO := diff(y(x),x,x) - cos(2*x)*y(x);
```

$$\mathit{Mathieu\_EDO} := (\frac{\partial^2}{\partial x^2}\,\mathrm{y}(x)) - \cos(2\,x)\,\mathrm{y}(x)$$

```
> sol_EDO := dsolve(Mathieu_EDO);
```

$$\mathit{trying}\ \ \mathit{way} = 3$$

$$[0,\ y]$$

$$\mathit{trying}\ \ \mathit{way} = 3$$

$$[0,\ y]$$

$$\mathit{trying}\ \ \mathit{way} = 5$$

$$\mathit{sol\_EDO} := \mathrm{y}(x) = \mathrm{DESol}(\{(\frac{\partial^2}{\partial x^2}\,\_\mathrm{Y}(x)) - \cos(2\,x)\,\_\mathrm{Y}(x)\},\ \{\_\mathrm{Y}(x)\})$$

porém essa EDO está classificada como um caso particular das EDO's elipsoidais.

```
> odeadvisor(Mathieu_EDO);
```

$$[\_\mathit{ellipsoidal}]$$

Note que a solução é dada pela função DESol(*EDO*, y(x)). Esse resultado pode ser usado em alguns cálculos posteriores como se fosse a própria solução explícita. Por exemplo, se

```
> f := rhs(sol_EDO);
```

$$f := \mathrm{DESol}(\{(\frac{\partial^2}{\partial x^2}\,\_\mathrm{Y}(x)) - \cos(2\,x)\,\_\mathrm{Y}(x)\},\ \{\_\mathrm{Y}(x)\})$$

então

```
> diff(f,x,x)/f;
```

$$\cos(2\,x)$$

Uma EDO é dita exata (*exact_linear*, *exact_nonlinear*, *exact order 1*) se

$$\tfrac{\partial}{\partial x}\,\mathrm{F}\left(x,\ \mathrm{y}(x),\ \tfrac{\partial}{\partial x}\,\mathrm{y}(x),\ \tfrac{\partial^2}{\partial x^2}\,\mathrm{y}(x),\ \tfrac{\partial^3}{\partial x^3}\,\mathrm{y}(x),\quad ...\quad\right) = 0.$$

Se $\mathrm{F}(x,\ y,\ z,\ ...)$ for uma função linear em todos os argumentos então a EDO também é linear. EDO's desse tipo podem sempre ser reduzidas de uma ordem. Por exemplo, a EDO de segunda ordem

```
> exact_EDO := diff(y(x),x)*(x+diff(y(x),x)^2*exp(y(x)) +
> 2*exp(y(x))*diff(y(x),x,x))=-y(x);
```

$$\mathit{exact\_EDO} := (\frac{\partial}{\partial x}\,\mathrm{y}(x))\,(x + (\frac{\partial}{\partial x}\,\mathrm{y}(x))^2\,e^{\mathrm{y}(x)} + 2\,e^{\mathrm{y}(x)}\,(\frac{\partial^2}{\partial x^2}\,\mathrm{y}(x))) = -\mathrm{y}(x)$$

é exata, como podemos confirmar com o comando *odeadvisor*.

```
> odeadvisor(exact_EDO);
```

$$[[\mathit{\_2nd\_order},\ \mathit{\_exact},\ \mathit{\_nonlinear}],\ [\mathit{\_2nd\_order},\ \mathit{\_reducible},\ \mathit{\_mu\_y\_y1}],\\ [\mathit{\_2nd\_order},\ \mathit{\_reducible},\ \mathit{\_mu\_poly\_yn}]]$$

Ela pode ser reduzida para uma EDO de primeira ordem

```
> sol := dsolve(exact_EDO);
```

$$\mathit{trying\ way} = 3$$

$$\mathit{trying\ way} = 2$$

$$\mathit{trying\ way} = 4$$

$$\mathit{trying\ way} = 5$$

$$\mathit{trying\ way} = 3$$

$$\mathit{trying\ way} = 5$$

$$sol := \mathrm{y}(x) = \_\mathrm{b}(\_a)\ \&\mathrm{where}[\{(\frac{\partial}{\partial \_a}\_\mathrm{b}(\_a))^2\, e^{\_\mathrm{b}(\_a)} + \_\mathrm{b}(\_a)\,\_a + 2\,\_C1 = 0\},$$
$$\{\_a = x,\ \_\mathrm{b}(\_a) = \mathrm{y}(x)\},\ \{x = \_a,\ \mathrm{y}(x) = \_\mathrm{b}(\_a)\}]$$

que pode ser selecionada com o seguinte comando

```
> EDO_reduzida := op([2,2,1,1],sol);
```

$$EDO\_reduzida := (\frac{\partial}{\partial \_a}\_\mathrm{b}(\_a))^2\, e^{\_\mathrm{b}(\_a)} + \_\mathrm{b}(\_a)\,\_a + 2\,\_C1 = 0$$

A EDO reduzida é do tipo *patterns*.

```
> odeadvisor(EDO_reduzida);
```

$$[y = \_G(x, y')]$$

porém não pode ser resolvida pelo comando *dsolve* com *_C1* arbitrário.

7.3 Pacote DEtools

7.3.1 Comandos para manipulação de EDO's

Os principais comando de manipulação são:

DEnormal, *autonomous*, *convertAlg*, *convertsys*, *indicialeq*, *reduceOrder*, *regularsp*, *translate*, *untranslate*, *varparam*.

O comando *reduceOrder* reduz a ordem da EDO uma vez conhecida uma ou mais soluções particulares para ela. A sintaxe é

reduceOrder(*EDO*, y(x), *solução_particular*, *opção*)

Por exemplo, a EDO de terceira ordem

```
> with(DEtools):
> EDO := cos(x)*diff(y(x),x$3)-diff(y(x),x$2) +
> Pi*diff(y(x),x) = y(x)-x;
```

$$EDO := \cos(x)\,(\frac{\partial^3}{\partial x^3}\mathrm{y}(x)) - (\frac{\partial^2}{\partial x^2}\mathrm{y}(x)) + \pi\,(\frac{\partial}{\partial x}\mathrm{y}(x)) = \mathrm{y}(x) - x$$

tem como solução particular a função

```
> f := x -> x + Pi;
```

$$f := x \to x + \pi$$

como podemos verificar com o comando *odetest*

```
> odetest(y(x)=f(x),EDO);
```

$$0$$

Podemos reduzir essa EDO para uma outra de segunda ordem

```
> EDO_reduzida := reduceOrder(EDO,y(x),x+Pi);
```

$$EDO\_reduzida := (\cos(x)\,x + \cos(x)\,\pi)\,(\frac{\partial^2}{\partial x^2}\,\mathrm{y}(x)) + (-x - \pi + 3\cos(x))\,(\frac{\partial}{\partial x}\,\mathrm{y}(x)) + (\pi\,x + \pi^2 - 2)\,\mathrm{y}(x)$$

cuja solução geral não inclui mais a função $\mathrm{f}(x) = x + \pi$.

O comando *convertAlg* converte um EDO linear em uma lista com 2 elementos. Se a EDO tem a forma

$$A_1\,\mathrm{y}(x) + A_2\,(\tfrac{\partial}{\partial x}\,\mathrm{y}(x)) + A_3\,(\tfrac{\partial^2}{\partial x^2}\,\mathrm{y}(x)) + ... = \mathrm{f}(x)$$

então o resultado da aplicação do comando é

$$[[A_1,\ A_2,\ A_3,\ ...],\ \mathrm{f}(x)].$$

O comando *convertsys* converte uma EDO ou um sistema de EDO's de qualquer ordem para um sistema de EDO's de primeira ordem.

7.3.2 Comandos para visualização

Os principais comandos para visualização das soluções são:

DEplot, *DEplot3d*, *dfieldplot*, *phaseportrait*.

O comando *DEplot* faz o gráfico da solução de uma EDO qualquer que seja a ordem (faz o gráfico de direcções de um sistema de duas EDO's de primeira ordem). A sintaxe para uma EDO é

DEplot(*EDO*, y(x), $x = a\,..\,b$, *CI's*, $y = c\,..\,d$, *opções*)

e para um sistema de duas EDO's é

DEplot({*EDO1*, *EDO2*}, { y(x), z(x)}, $x = a\,..\,b$, *CI's*, $y = c\,..\,d$, $z = e\,..\,f$, *opções*)

onde *CI's* são as condições iniciais nas forma de lista de listas (veja exemplos abaixo). Considere a seguinte EDO de segunda ordem que não pode ser resolvida com o comando *dsolve*.

```
> EDO := cos(x)*(x+Pi)*diff(y(x),x,x) - (x+Pi-3*cos(x))*
> diff(y(x),x)+(Pi*x+Pi^2-2)*y(x) = 0;
```

$$EDO := \cos(x)\,(x+\pi)\,(\frac{\partial^2}{\partial x^2}\,\mathrm{y}(x)) - (x+\pi-3\cos(x))\,(\frac{\partial}{\partial x}\,\mathrm{y}(x)) + (\pi\,x+\pi^2-2)\,\mathrm{y}(x) = 0$$

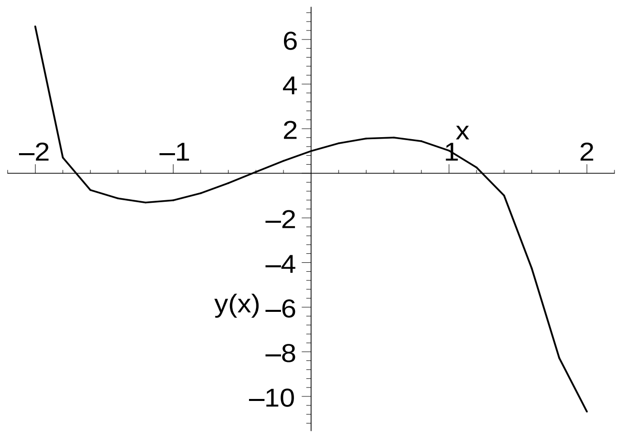
Considere as condições iniciais

```
> CI := [[y(0)=1,D(y)(0)=2]];
```

$$CI := [[\mathrm{y}(0) = 1, \mathrm{D}(y)(0) = 2]]$$

O gráfico da solução para x no intervalo $[-2, 2]$ é

```
> DEplot(EDO, y(x), x=-2..2, CI, linecolor=red);
```



O sistema de EDO's

```
> sistema_EDO := {diff(y(x),x)=y(x)-z(x),diff(z(x),x) =
> z(x)-2*y(x)};
```

$$sistema\_EDO := \{\frac{\partial}{\partial x}\,\mathrm{z}(x) = \mathrm{z}(x) - 2\,\mathrm{y}(x),\ \frac{\partial}{\partial x}\,\mathrm{y}(x) = \mathrm{y}(x) - \mathrm{z}(x)\}$$

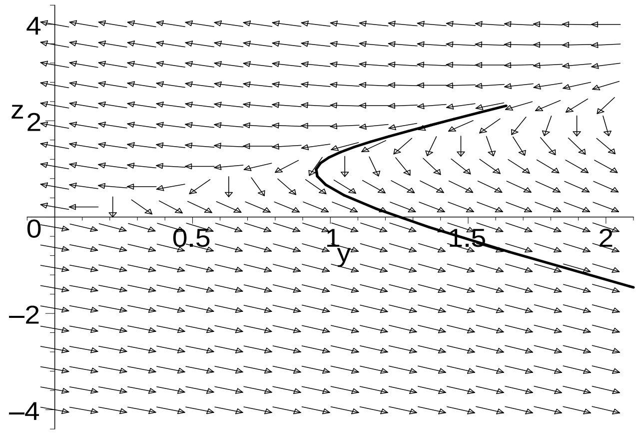
com as condições iniciais

```
> CI := [[y(0)=1.638,z(0)=2.31]];
```

$$CI := [[\mathrm{y}(0) = 1.638, \mathrm{z}(0) = 2.31]]$$

tem o seguinte gráfico de direções com a curva que obedece as condições iniciais em destaque.

```
> DEplot(sistema_EDO, {y(x),z(x)}, x=0..3, CI, y=0..2,
> z=-4..4, color=black, linecolor=red);
```

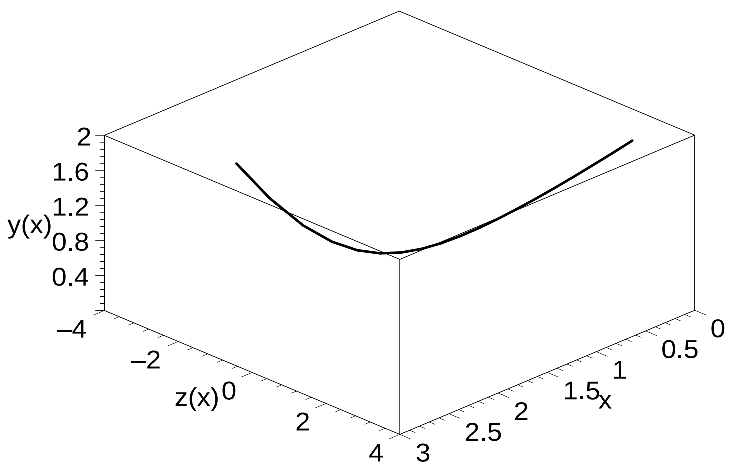


O mesmo gráfico pode ser obtido com o comando *phaseportrait* ou com o comando *dfieldplot*, porém este último não usa condições iniciais e portanto não destaca nenhuma curva especial. Os comandos equivalentes que geram o gráfico acima são

```
> dfieldplot(sistema_EDO, {y(x),z(x)}, x=0..3, y=0..2, z=-4..4);
> phaseportrait(sistema_EDO, {y(x),z(x)}, x=0..3, CI,y=0..2, z=-4..4);
```

O gráfico tridimensional da curva em destaque do último gráfico (bidimensional) pode ser feito com o comando *DEplot3d*. A opção *scene* = $[x, \mathrm{z}(x), \mathrm{y}(x)]$ especifica a ordem dos eixos no gráfico.

```
> DEplot3d(sistema_EDO, {y(x),z(x)}, x=0..3, CI, y=0..2,
> z=-4..4, scene=[x,z(x),y(x)], linecolor=black);
```



7.3.3 Outros comandos que retornam soluções de EDO's

Exite uma série de comandos que resolvem EDO's de tipos particulares. A maioria desses métodos está implementada no comando *dsolve* porém as soluções retornadas pelo *dsolve* não são totalmente equivalentes às soluções retornadas por esses comandos particulares, principalmente com relação a forma das soluções e com relação a soluções singulares de EDO's não lineares. A lista desses comando é

RiemannPsols, *abelsol*, *bernoullisol*, *chinisol*, *clairautsol*, *constcoeffsols*, *eulersols*, *exactsol*, *expsols*, *genhomosol*, *kovacicsols*, *liesol*, *linearsol*, *matrixDE*, *parametricsol*, *polysols*, *ratsols*, *riccatisol*, *separablesol*.

Vejamos um exemplo. O comando *matrixDE* acha a solução de um sistema de EDO's lineares da forma

$$\frac{\partial}{\partial t}\,\mathrm{X}(t) = \mathrm{A}(t)\,\mathrm{X}(t) + \mathrm{B}(t)$$

onde $\mathrm{A}(t)$ e $\mathrm{B}(t)$ são matrizes. Por exemplo, o sistema de EDO's pode ser resolvido da seguinte forma.

```
> A := matrix(2,2,[1,t,t,1]);
```

$$A := \begin{bmatrix} 1 & t \\ t & 1 \end{bmatrix}$$

```
> sol := matrixDE(A,t);
```

$$sol := \left[\begin{bmatrix} e^{(-1/2\,t\,(t-2))} & \sinh(\frac{1}{2}\,t^2)\,e^t \\ -e^{(-1/2\,t\,(t-2))} & \cosh(\frac{1}{2}\,t^2)\,e^t \end{bmatrix},\ [0,\,0]\right]$$

7.4 Método Numérico

Podemos resolver uma equação diferencial sem parâmetros livres usando métodos numéricos. A solução é dada na forma de um procedimento que pode ser usado para gerar o valor da solução em determinados pontos ou para gerar o gráfico da solução. Vimos que a equação de Mathieu não pode ser resolvida exatamente. Vamos fazer o gráfico da solução com condições iniciais.

```
> Mathieu_EDO := diff(y(x),x,x)-cos(2*x)*y(x);
```

$$Mathieu\_EDO := (\frac{\partial^2}{\partial x^2}\,\mathrm{y}(x)) - \cos(2\,x)\,\mathrm{y}(x)$$

```
> ci := y(0)=1, D(y)(0)=1;
```

$$ci := \mathrm{y}(0) = 1,\ \mathrm{D}(y)(0) = 1$$

```
> F := dsolve( { Mathieu_EDO, ci }, y(x), numeric):
```

F é um procedimento que fornece o valor da função e da derivada primeira uma vez dado o ponto x:

```
> F(0);
```

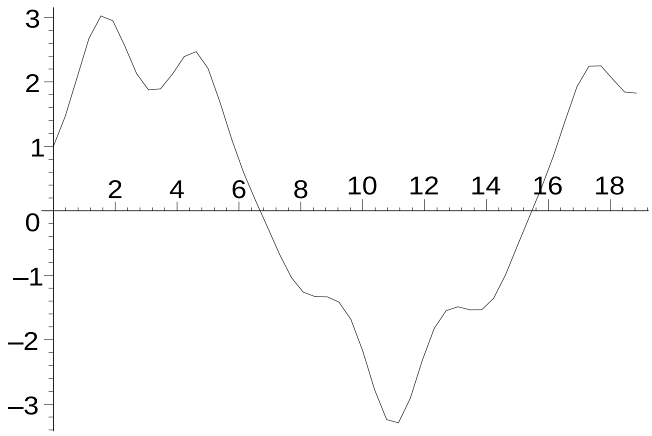
$$[x = 0, \mathrm{y}(x) = 1., \frac{\partial}{\partial x}\mathrm{y}(x) = 1.]$$

```
> F(1);
```

$$[x = 1, \mathrm{y}(x) = 2.45032122804124298, \frac{\partial}{\partial x}\mathrm{y}(x) = 1.56805429249998096]$$

Para fazer o gráfico, podemos usar o comando *odeplot* do pacote *plots*:

```
> plots[odeplot](F,[x,y(x)], 0..6*Pi);
```



Se o usuário não especificar o método de resolução numerica, o Maple usa o método de Euler. Existem vários outros métodos como o de Runge-Kutta de segunda, terceira ou quarta ordem, Adams-Bashford, séries de Taylor entre vários outros. Para mais informações veja o *help on line* de *dsolve,numeric*.

7.5 Método de Séries

É possível encontrar uma expansão em séries para a solução de uma EDO ou de um sistema de EDO's sem a resolução direta da equação. Os métodos usados são o método de iteração de Newton, método de Frobenius ou método de substituição por uma séria com coeficientes arbitrários a serem determinados. A sintaxe é

dsolve(*EDO*, y(x), *series*)
dsolve({*EDO*, *IC's*}, y(x), *series*)
dsolve({*seq_de_EDOs*, *CI's*}, {*funcs*}, *series*)

onde

EDO é uma equação diferencial ordinária
y(x) é uma função indeterminada de um única variável
CI's são as condicões iniciais
{*seq_de_EDOs*} é um conjunto de EDO's
{*funcs*} é um conjunto de funções indeterminadas de um única variável

Novamente vamos usar a equação de Mathieu como exemplo.

```
> Order:=12:
> dsolve({Mathieu_EDO, y(0)=0, D(y)(0)=1}, y(x), series);
```

$$\mathrm{y}(x) = x + \frac{1}{6}\,x^3 - \frac{11}{120}\,x^5 + \frac{29}{5040}\,x^7 + \frac{71}{24192}\,x^9 - \frac{3151}{4435200}\,x^{11} + \mathrm{O}(x^{12})$$

A variável *Order* controla a ordem da solução. Vejamos um exemplo de um sistema de EDO's.

```
> Order := 3:
> sistema := {diff(y(t),t)=-x(t),diff(x(t),t)=y(t)};
```

$$\mathit{sistema} := \{\frac{\partial}{\partial t}\,\mathrm{x}(t) = \mathrm{y}(t),\, \frac{\partial}{\partial t}\,\mathrm{y}(t) = -\mathrm{x}(t)\}$$

```
> dsolve(sistema union {x(0)=A,y(0)=B}, {x(t),y(t)},
> type=series);
```

$$\{\mathrm{y}(t) = B - A\,t - \frac{1}{2}\,B\,t^2 + \mathrm{O}(t^3),\, \mathrm{x}(t) = A + B\,t - \frac{1}{2}\,A\,t^2 + \mathrm{O}(t^3)\}$$

Compare o resultado com a expansão em série da solução exata.

```
> res := dsolve(sistema union {x(0)=A,y(0)=B},{x(t),y(t)});
```

$$\mathit{res} := \{\mathrm{x}(t) = A\cos(t) + B\sin(t),\, \mathrm{y}(t) = -A\sin(t) + B\cos(t)\}$$

```
> map(x->lhs(x)=series(rhs(x),t), res);
```

$$\{\mathrm{y}(t) = B - A\,t - \frac{1}{2}\,B\,t^2 + \mathrm{O}(t^3),\, \mathrm{x}(t) = A + B\,t - \frac{1}{2}\,A\,t^2 + \mathrm{O}(t^3)\}$$

Para mais informações veja o *help on line* de *dsolve,series*.

Chapter 8

Equações diferenciais parciais

8.1 Introdução

O novo comando *pdsolve* é o principal comando para resolver analiticamente EDP's. Ele substitui o antigo comando*pdesolve*. Apesar de ter muitas limitações, houve um avanço considerável com relação à implementação anterior. A sintaxe do novo comando é

pdsolve(*EDP*)
pdsolve(*EDP*, f(*x*,*y*,...), *HINT*=..., *INTEGRATE*, *build*)

onde

EDP é uma equação diferencial parcial
f(*x*,*y*,...) é função indeterminada de várias variáveis (opcional)
HINT=... é um parâmetro de sugestões dado pelo usuário (opcional)
INTEGRATE provoca a integração automática das ODE's encontradas pelo método de separação de variáveis da EDP (optional)
build tenta construir uma expressão explícita para a função indeterminada seja qual for a generalidade da solução encontrada (opcional)

Os parâmetros opcionais podem ser usados em qualquer ordem. Se o comando *pdsolve* não tiver sucesso, a opção *HINT* permite ao usuário sugerir uma estratégia para a resolução da EDP.

8.2 Solução Geral e Solução Quase-geral

O comando *pdsolve* tenta em primeiro lugar achar uma solução geral da EDP. Por exemplo, considere a seguinte EDP.

```
> restart;
> EDP := exp(y)*diff(F(x,y,z),x) + 2*sin(2*x)*
> diff(F(x,y,z),y) = 0;
```

$$EDP := e^y\,(\frac{\partial}{\partial x}\,\mathrm{F}(x,\,y,\,z)) + 2\sin(2\,x)\,(\frac{\partial}{\partial y}\,\mathrm{F}(x,\,y,\,z)) = 0$$

O comando *pdsolve* encontra a solução geral em termos da função arbitrária *_F1*.

```
> sol := pdsolve(EDP);
```

$$sol := \mathrm{F}(x,\, y,\, z) = \_\mathrm{F1}(\frac{1}{2}\, e^y + \frac{1}{2}\cos(2\, x),\, z)$$

Podemos testar o resultado através do comando *pdetest*.

```
> pdetest(sol,EDP);
```

$$0$$

A solução de uma EDP de ordem n dependendo de k variáveis é geral quando é dada em termos de n funções arbitrárias e $k-1$ variáveis. No exemplo acima, a ordem da EDP é 1 (derivada de ordem mais alta) e o número de variáveis é 3 (x, y, z). Podemos ver que a solução é dada em termos de 1 função arbitrária (*_F1*) que depende de 2 variáveis.

Se *pdsolve* não consegue achar a solução geral, a solução é dada usando a estrutura

(*solução em termos de _F1, _F2, ...*) *&where* [{*EDO's envolvendo _F1, _F2, ...*}]

Nesse caso, *_F1*, *_F2*, ... não são funções arbitrárias. Elas obedecem às EDO's descritas pela segunda componente do operador *&where*. Por exemplo, a equação de difusão (ou de propagação de calor) em 1 dimensão é

```
> EDP := a^2*diff(f(x,t),x,x)+0*diff(f(x,y),y,y) = diff(f(x,t),t);
```

$$EDP := a^2\,(\frac{\partial^2}{\partial x^2}\,\mathrm{f}(x,\, t)) = \frac{\partial}{\partial t}\,\mathrm{f}(x,\, t)$$

A solução é dada da forma

```
> sol1 := pdsolve(EDP);
```

$$sol1 := (\mathrm{f}(x,\, t) = \_\mathrm{F1}(x)\,\_\mathrm{F2}(t))\ \&\mathrm{where}$$
$$[\{\frac{\partial}{\partial t}\,\_\mathrm{F2}(t) = a^2\,\_c_1\,\_\mathrm{F2}(t),\ \frac{\partial^2}{\partial x^2}\,\_\mathrm{F1}(x) = \_c_1\,\_\mathrm{F1}(x)\}]$$

onde *_c1* é a constante de separação de variáveis. O argumento opcional *build* provoca a integração das EDO's dadas na segunda componente de *&where*.

```
> sol2 := pdsolve(EDP,build);
```

$$sol2 := \mathrm{f}(x,\, t) = e^{(\sqrt{\_c_1}\, x)}\,\_C3\, e^{(a^2\,\_c_1\, t)}\,\_C1 + \frac{\_C3\, e^{(a^2\,\_c_1\, t)}\,\_C2}{e^{(\sqrt{\_c_1}\, x)}}$$

onde as constantes *_C1*, *_C2*, *_C3* foram introduzidas na integração de uma ODE de segunda ordem e uma ODE de primeira orderm. A opção *INTEGRATE* produz um resultado similar mantendo o operador *&where*.

```
> sol3 := pdsolve(EDP,INTEGRATE);
```

$$sol3 := (\mathrm{f}(x,\, t) = \_\mathrm{F1}(x)\,\_\mathrm{F2}(t))\ \&\mathrm{where}$$
$$[\{\{\_\mathrm{F1}(x) = \_C1\, e^{(\sqrt{\_c_1}\, x)} + \_C2\, e^{(-\sqrt{\_c_1}\, x)}\},\ \{\_\mathrm{F2}(t) = \_C3\, e^{(a^2\,\_c_1\, t)}\}\}]$$

O comando *pdetest* funciona com qualquer dessas formas de solução.

```
> seq(pdetest(sol||i,EDP),i=1..3);
```

$$0, 0, 0$$

8.3 O Pacote PDEtools

O pacote *PDEtools* contem funções que complementam o comando *pdsolve* na tarefa de encontrar soluções para EDP's.

```
> with(PDEtools);
```

[*PDEplot*, *build*, *casesplit*, *charstrip*, *dchange*, *dcoeffs*, *declare*, *difforder*, *dsubs*, *mapde*, *separability*, *splitstrip*, *splitsys*, *undeclare*]

8.3.1 Comando build

O comando *build* constrói uma solução explícita para as funções indeterminadas que obedecem as EDO's dentro do operador *&where*. O argumento do comando *build* deve ser uma solução fornecido pelo comando *pdsolve*. Por exemplo, considere a seguinte EDP.

```
> EDP := diff(f(x,y),y)*exp(x)+diff(f(x,y),x)^2*log(y)=0;
```

$$Laplace\_EDP := (\frac{\partial}{\partial y}\,\mathrm{f}(x,\,y))\,e^x + (\frac{\partial}{\partial x}\,\mathrm{f}(x,\,y))^2\ln(y) = 0$$

A soluo é dada em termos de duas funções indeterminadas.

```
> sol := pdsolve(EDP);
```

$$sol := (\mathrm{f}(x,\,y) = \_\mathrm{F1}(x) + \_\mathrm{F2}(y))\ \&\mathrm{where}\,[\{(\frac{\partial}{\partial x}\,\_\mathrm{F1}(x))^2 = \_c_1\,e^x,\ \frac{\partial}{\partial y}\,\_\mathrm{F2}(y) = -\_c_1\ln(y)\}]$$

Para construir uma solução explícita:

```
> build(sol);
```

$$\mathrm{f}(x,\,y) = 2\sqrt{\_c_1\,e^x} + \_C1 - \_c_1\,y\ln(y) + \_c_1\,y + \_C2$$

8.3.2 Comando dchange

O comando *dchange* serve para fazer mudança de variáveis em EDP (e em outras expressões algébricas). A título de exemplo vamos resolver a EDP

```
> EDP := 3*diff(f(x,y),x) + 4*diff(f(x,y),y) - 2*f(x,y) =1;
```

$$EDP := 3\,(\frac{\partial}{\partial x}\,\mathrm{f}(x,\,y)) + 4\,(\frac{\partial}{\partial y}\,\mathrm{f}(x,\,y)) - 2\,\mathrm{f}(x,\,y) = 1$$

sem usar o comando *pdsolve*. Vamos considerar a seguinte tranformação de variáveis

```
> tr := {x = v*cos(alpha) - w*sin(alpha), y = v*sin(alpha) + w*cos(alpha)};
```

$$tr := \{x = v\cos(\alpha) - w\sin(\alpha),\ y = v\sin(\alpha) + w\cos(\alpha)\}$$

e a partir dela determinar o valor de α que transforma a EDP numa ODE. Uma vez que essa tranformação possui o parâmetro α, vamos adicionar o terceiro argumento $[v, w]$ no comando *dchange*.

```
> new_EDP := dchange(tr,EDP,[v,w]);
```

$$new\_EDP := 3\cos(\alpha)\left(\frac{\partial}{\partial v}\mathrm{f}(v, w, \alpha)\right) - 3\sin(\alpha)\left(\frac{\partial}{\partial w}\mathrm{f}(v, w, \alpha)\right) + 4\sin(\alpha)\left(\frac{\partial}{\partial v}\mathrm{f}(v, w, \alpha)\right) + 4\cos(\alpha)\left(\frac{\partial}{\partial w}\mathrm{f}(v, w, \alpha)\right) - 2\,\mathrm{f}(v, w, \alpha) = 1$$

Vamos fazer a seguinte substituição para eliminar o parâmetro α que infelizmente foi introduzido na função f.

```
> new_EDP := subs(f(v,w,alpha)=f(v,w),new_EDP);
```

$$new\_EDP := 3\cos(\alpha)\left(\frac{\partial}{\partial v}\mathrm{f}(v, w)\right) - 3\sin(\alpha)\left(\frac{\partial}{\partial w}\mathrm{f}(v, w)\right) + 4\sin(\alpha)\left(\frac{\partial}{\partial v}\mathrm{f}(v, w)\right) + 4\cos(\alpha)\left(\frac{\partial}{\partial w}\mathrm{f}(v, w)\right) - 2\,\mathrm{f}(v, w) = 1$$

Vamos determinar α de forma que o coeficiente do termo $\frac{\partial}{\partial w}\mathrm{f}(v, w)$ seja zero.

```
> coeff_w := coeff(lhs(new_EDP),diff(f(v,w),w));
```

$$coeff\_w := -3\sin(\alpha) + 4\cos(\alpha)$$

```
> alpha := solve(coeff_w=0, alpha);
```

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

Vejamos o formato da EDP agora.

```
> new_EDP;
```

$$5\left(\frac{\partial}{\partial v}\mathrm{f}(v, w)\right) - 2\,\mathrm{f}(v, w) = 1$$

Ela pode ser resolvida como o comando *dsolve* (em vez de *pdsolve*).

```
> sol := dsolve(new_EDP,f(v,w));
```

$$sol := \mathrm{f}(v, w) = -\frac{1}{2} + e^{(2/5\,v)}\,\_\mathrm{F1}(w)$$

O último passo é recuperar as variáveis originais.

```
> f(x,y) = subs(solve(tr,{v,w}), rhs(sol));
```

$$\mathrm{f}(x, y) = -\frac{1}{2} + e^{(6/25\,x + 8/25\,y)}\,\_\mathrm{F1}\left(\frac{3}{5}\,y - \frac{4}{5}\,x\right)$$

Podemos confirmar a solução.

```
> pdetest(%, EDP);
```

$$0$$

8.3.3 Comando PDEplot

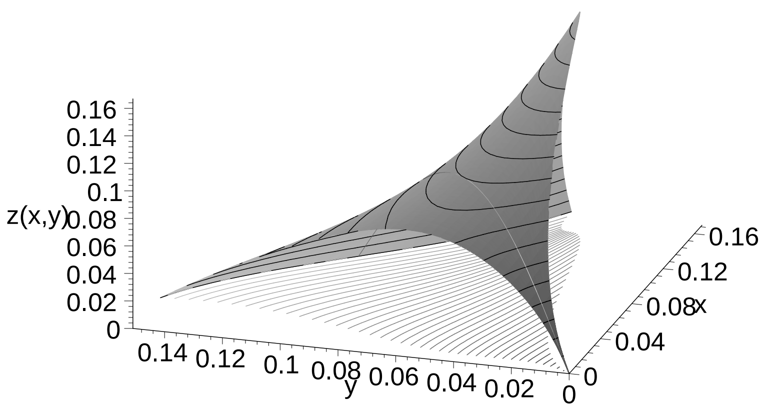
O comando *PDEplot* faz o gráfico da solução de uma EDP de primeira ordem uma vez dadas as condições iniciais. A EDP não pode ter parâmetro livres. Como exemplo, considere a seguinte EDP.

```
> MapleV4_logo_EDP := (y^2+z(x,y)^2+x^2)*diff(z(x,y),x) -
> 2*x*y*diff(z(x,y),y)- 2*z(x,y)*x = 0;
```

$$MapleV4\_logo\_EDP := (y^2+\mathrm{z}(x,\,y)^2+x^2)\,(\frac{\partial}{\partial x}\,\mathrm{z}(x,\,y))-2\,x\,y\,(\frac{\partial}{\partial y}\,\mathrm{z}(x,\,y))-2\,\mathrm{z}(x,\,y)\,x=0$$

cuja solução fornece o gráfico usado como logo da versão 4 do Maple.

```
> PDEplot(MapleV4_logo_EDP, z(x,y), [t,t,sin(Pi*t/0.1)/10],
> t=0..0.1, numchar=40, orientation=[-163,56],
> basechar=true, numsteps=[20,20], stepsize=.15,
> initcolour=cos(t)*t, animate=false, style=PATCHCONTOUR);
```



8.4 Limitações do Comando pdsolve

A implementação da versão 6 não é capaz de resolver

- sistemas de EDP's,
- EDP obedecendo condições de contorno,
- por métodos numéricos,
- por métodos de séries.

Bibliography

[1] Heal, K. M., Hansen, M., Rickard, K., Maple V Learning Guide, Springer-Verlag, New York, 1997.

[2] Monagan, M., Geddes, K., Heal, K., Labahn, G., Vorkoetter, S., Maple V Programming Guide, Springer-Verlag, New York, 1997.

[3] Heck, A., Introduction to Maple, Second edition, Springer-Verlag, New York, 1996.

[4] Wright, F., Computing with Maple, Chapman & Hall/CRC, 2001.

[5] Kofler, M., Maple: An Introduction and Reference, Addison-Wesley, 1997.

[6] Kamke, E., Differentialgleichungen: Losungsmethoden und Losungen, Chelsea Publishing Co, New York (1959).

[7] Murphy, G.M., Ordinary Differential Equations and their Solutions, Van Nostrand, Princeton, 1960.

[8] Zwillinger, D. Handbook of Differential Equations, 2nd edition, Academic Press (1992).