INTRODUÇÃO AO MAPLE

por Renato Portugal (LAFEX/CBPF e FIS/PUC-Rio)

Índice

Índice i

1 Introdução **1**
1 Aspectos do Sistema Maple 1
2 Help on line 2
3 A worksheet 2
4 Comentários e Agradecimentos 3

2 Noções Básicas **5**
1 Números 5
2 Atribuição de um nome a uma expressão 8
3 Avaliação Completa 9
4 Tipos de Objetos 10

3 Simplificação de Expressões **14**
1 Introdução 14
2 *Expand* 15
3 *Combine* 16
4 *Convert* 18
5 *Simplify* 19

4 Álgebra Linear **21**
1 Introdução 21
2 Definindo uma Matriz 22
3 Matrizes Especiais 23
4 Autovalores e Autovetores 25
5 Manipulação Estrutural de Matrizes 26
6 Cálculo Vetorial 27

5 Gráficos **30**
1 Introdução 30
2 Gráficos em 2 Dimensões 33
2.1 Introdução 33
2.2 Gráficos de Funções Parametrizadas 38
2.3 Gráficos em Coordenadas Polares 39
2.4 Gráficos de Funções Contínuas por Partes 40
2.5 Gráficos de Pontos Isolados 42

2.6 Gráficos de Funções Definidas Implicitamente . . . 47
3 Gráficos em 3 Dimensões . . . 48
3.1 Introdução . . . 48
3.2 Gráficos de Funções Parametrizadas . . . 50
3.3 Gráficos em Coordenadas Esféricas . . . 50
4 Exibindo vários Gráficos Simultaneamente . . . 51
5 Animando Gráficos em duas ou três Dimensões . . . 53
6 Colocando Textos em Gráficos . . . 56
7 Imprimindo Gráficos . . . 57
8 Manipulando Gráficos . . . 57

6 Cálculo Diferencial e Integral **60**
1 Funções Matemáticas . . . 60
1.1 Definindo uma Função . . . 60
1.2 Álgebra e Composição de Funções (@ e @@) . . . 63
2 Integral . . . 64

7 Equações Diferenciais **65**
1 Equações Diferenciais Ordinárias . . . 65
1.1 Introdução . . . 65
1.2 Método Numérico . . . 67
1.3 Método de Séries . . . 68
1.4 Método de Séries de Potência . . . 69
2 Equações Diferenciais Parciais . . . 70

8 Introdução à Programação **72**
1 Introdução . . . 72
2 Completando o Quadrado . . . 72
3 Diagonalizando uma Matriz . . . 76

9 Aplicação: Oscilação de Membranas **81**
1 Equação de Movimento da Membrana . . . 81
2 Membrana Retangular . . . 81
2.1 Resolução da equação de onda . . . 81
2.2 Animação dos Modos Normais . . . 84
3 Membrana Circular . . . 87

Exercícios **92**

Referências **95**

Capítulo 1

Introdução

1 Aspectos do Sistema Maple

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

- computação algébrica
- computação numérica
- computação gráfica
- programação

Todas 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 ter que 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 notação da linguagem C ou para a linguagem Fortran.

Os aspectos novos trazidos por esse *software* juntamente com outros sistemas algébricos são a computação algébrica e a programação simbólica. A computação algébrica é uma área que teve um forte desenvolvimento nas décadas de 60 e 70, onde foram desenvolvidos 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. No início do desenvolvimento desta área, uma série de tentativas foram feitas de usar recursos heurísticos provenientes da Inteligência Artificial, no entanto os resultados foram ineficientes. Os programas para integração analítica eram lentos, resolviam algumas integrais difícies, porém falhavam em integrais simples que qualquer aluno de Cálculo consegue determinar.

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. Noventa e cinco por cento dos algoritmos estão escritos na linguagem Maple e está acessível 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 todos estes aspectos e analisaremos como eles se inter-relacionam. A versão utilizada é ***Maple V Release 4***.

2 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
```

Dentro de um texto é possível criar *hyperlinks* com as *worksheets* de ajuda do próprio Maple. Por exemplo, clicando na palavra sublinhada a seguir, teremos acesso à *worksheet* de ajuda: maple.

A página de ajuda contêm várias partes incluindo uma descrição detalhada sobre a função, e no final, contêm palavras chaves correlacionadas com o assunto. A partir dessas 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 *Contents*. Podemos selecionar o tópico *Mathematics* clicando no sinal +. A partir daí podemos entrar na ajuda para *Algebra* ou *Calculus* etc.

3 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 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*, um usuário pode desenvolver a solução de um problema usando o Maple, 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* ou pode ser automaticamente convertida em um arquivo *Latex*. Um exemplo de uso das *worksheets* é esse curso. Ele foi apresentado e as notas impressas a partir de *worksheets*.

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. O resto do trabalho se ajusta automaticamente às mudanças. Essa idéia é a mesma dos processadores de textos que vêm gradativamente substituindo 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 no *Latex*, pois a *worksheet* tem interface com este processador. 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* tem quatro tipos de linhas que são: as linhas de entrada de comando, geralmente precedidas pelo sinal de pronto "`>`", as linhas de saída dos comandos, as linhas de texto e as linhas de gráfico. Algumas dessas linhas podem ser convertidas umas nas outras. Em geral, as linhas de entrada, de saída e de texto podem ser convertidas entre si. As versões de *worksheet* para estações de trabalho e para microcomputadores não são iguais entre si, porém na grande maioria dos casos, tudo o que um usuário faz na *worksheet* de uma estação de trabalho pode ser feita na *worksheet* de um micro e vice-versa.

As linhas de saída usam os 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 saídas 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* é gravada com a terminação *.mws* (ou *.ms* na versão *V Release 3*) e ela própria é um arquivo *ASCII*. Isso significa que ela pode ser enviada por correio eletrônico. É claro que a pessoa que recebe tem que editá-la e retirar o cabeçalho do correio para que ela seja lida corretamente pelo Maple.

A versão *V Release 4* possui diversos recursos para escrever textos. É possível criar seções e sub-seções. As letras podem ter diversos tamanhos e estilos, podem ser em 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*.

4 Comentários e Agradecimentos

Esta apostila foi usada em mini-curso proferido no CBPF[1] e nos cursos de Métodos Matemáticos da Física e Engenharia I e II na PUC-Rio em 1996. Ela foi escrita tendo em vista tantos os alunos dos cursos de ciências exatas como profissionais que não tenham tido contato com esta linguagem no curso universitário. Não é necessário nenhum conhecimento prévio de outras linguagens de computação. É claro que é necessário um conhecimento mínimo do sistema operacional ou do ambiente de janelas do computador que está sendo usado para processar o Maple.

Para se ter uma base nesta linguagem, não é necessário uma leitura de todos os capítulos deste trabalho. O autor recomenda uma leitura cuidadosa dos três primeiros. Os outros capítulos podem ser lidos conforme a necessidade do usuário.

O autor agradace a professora Sarah de Castro Barbosa pelos comentários e aos professores do departamento de Física da PUC-Rio que deram suporte para a introdução da computação algébrica nos cursos de Métodos Matemáticos da PUC-Rio.

[1]Centro Brasileiro de Pesquisas Física, Rua Dr. Xavier Sigaud 150, Urca, 22290-180.

O autor está aberto para comentários e críticas sobre este trabalho que podem ser enviados por *e-mail* para PORTUGAL@CAT.CBPF.BR. As *worksheets* usadas para escrever este trabalho estarão a disposição por mídia eletrônica através do *ftp* anônimo ANONYMOUS@LCA1.DRP.CBPF.BR no diretório *pub/apostila*. A referência [4] complementa este trabalho na parte de programação.

Capítulo 2

Noções Básicas

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

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(");
```

$$10534.22314$$

As aspas duplas " têm o valor do último resultado calculado. O seu valor 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. Este último resultado tem 10 dígitos (valor *default*). Podemos aproximar o resultado com quantos dígitos queiramos, por exemplo, com 50 dígitos:

```
> evalf("",50);
```

$$10534.223140495867685950413223140495867685950413 2$$

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, já que 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;
```

$$10$$

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

$$Digits := 20$$

Vejamos agora o número de Euler com vinte dígitos:

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

$$2.71828182845904523 54$$

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 as simplificações mais elementares são feitas, como 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. Pois 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 no mesmo 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*. Outro detalhe que podemos observar deste 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);
```

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

Obtivemos somente a raíz principal. Como se obtêm as outras duas raízes? Temos que fazer em duas etapas. Primeiro converter o número complexo na forma de *RootOf* e depois pedir todas as raízes com o comando *allvalues:*

```
> convert(z,RootOf);
```

$$RootOf(\_Z^3 + 8)$$

```
> allvalues(");
```

$$-2,\ 1 - I\sqrt{3},\ 1 + I\sqrt{3}$$

A variável I é um apelido para $\sqrt{-1}$. No momento em que o Maple é disparado, o comando $alias(I = \sqrt{-1})$ é dado automaticamente estabelecendo que o apelido para $\sqrt{-1}$ é I. O usuário pode dar outro apelido, por exemplo $alias(J = \sqrt{-1})$, e pode cancelar o apelido anterior através do comando $alias(I = I)$. 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));
```

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

2 Atribuição de um nome a uma expressão

Em princípio, uma expressão não precisa ter um nome. Podemos nos referir a ela com as aspas duplas. 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"):

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

$$y := y$$

ou equivalentemente:

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

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 em B que por sua vez foi avaliado em C que foi avaliado em 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: no comando abaixo a variável A não será avaliada:

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

$$A := 10$$

Caso A tivesse sido avaliado em 3, teríamos o comando 3 :=10, 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 ou de um comando, é utilizado com muita frequência.

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(");
```

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

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 ];
```

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

Ou na forma de um conjunto:

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

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

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[nops(lista_de_solucoes)];
```

$$1$$

Neste último exemplo, podemos sempre selecionar o último elemento mesmo sem ter contado a mão (ou no olho) quantos elementos o objeto tem. 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 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 $elem \rightarrow x + elem$. 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 que 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:

OBJETO[a .. 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];
```

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

Capítulo 3

Simplificação de Expressões

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 a sequência de passos, o computador sempre chega na 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 último exemplo temos x elevado a potência 20, porém poderia ser x elevado a 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 decidir qual é a forma que ele quer que a expressão seja colocada. 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*.

2 *Expand*

Quando um usuário entra com uma expressão, o Maple a mantêm na mesma forma por via de regra. A expressão não tem a sua forma alterada até que o usuário peça. Não há um pradrão predeterminado de apresentação das expressões algébricas, e por isso elas são armazenadas internamente na forma 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 que congela uma sub-expressão. No caso da expressão $sin(\gamma+\theta)$ basta colocar *sin* como segundo argumento do comando *expand* para congelar essa função. 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 *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 os conceitos de 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*. 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, combinamos a expressão com relação a 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. 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));
```

$$ln(x) + (\lim_{x\to a} f(x,\, y,\, \frac{3+4\, I}{1+I\, a})(x)^3)\, (\lim_{x\to a} f(x,\, y,\, \frac{3+4\, I}{1+I\, a})(x)^2) =$$
$$(\lim_{x\to a} f(x,\, y,\, \frac{3+4\, I}{1+I\, a})(x)^5) + ln(x)$$

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.

4 *Convert*

A princípio, a função *convert* não é um comando de simplificação pois ele tem uma utilidade 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*, etc, 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);
```

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

5 *Simplify*

O comando *simplify* é um comando geral de simplificação. A primeira coisa que ele faz é procurar dentro da expressão a ocorrência de funções matemáticas, como as funções trigonométricas, e 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);
```

$$x\,y$$

Capítulo 4

Álgebra Linear

1 Introdução

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

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

```
Warning, new definition for norm

Warning, new definition for trace
```

Normalmente, terminarmos o comando de carregar pacotes com dois pontos para que as funções do pacote não sejam mostradas. Somente as mensagens de aviso de redefinição de comandos serão mostradas. Isto é o que acontece com os comandos *norm* e *trace*, que servem primeiramente para calcular norma de polinômios e para correção de procedimentos, respectivamente. Após o pacolte *linalg* ser carregado, eles passam a calcular norma de vetores e traço de matrizes. No presente contexto, queremos saber quais são estas funções:

```
> with(linalg);
```

[*BlockDiagonal*, *GramSchmidt*, *JordanBlock*, *LUdecomp*, *QRdecomp*, *Wronskian*, *addcol*, *addrow*, *adj*, *adjoint*, *angle*, *augment*, *backsub*, *band*, *basis*, *bezout*, *blockmatrix*, *charmat*, *charpoly*, *cholesky*, *col*, *coldim*, *colspace*, *colspan*, *companion*, *concat*, *cond*, *copyinto*, *crossprod*, *curl*, *definite*, *delcols*, *delrows*, *det*, *diag*, *diverge*, *dotprod*, *eigenvals*, *eigenvalues*, *eigenvectors*, *eigenvects*, *entermatrix*, *equal*, *exponential*, *extend*, *ffgausselim*, *fibonacci*, *forwardsub*, *frobenius*, *gausselim*, *gaussjord*, *geneqns*, *genmatrix*, *grad*, *hadamard*, *hermite*, *hessian*, *hilbert*, *htranspose*, *ihermite*, *indexfunc*, *innerprod*, *intbasis*, *inverse*, *ismith*, *issimilar*, *iszero*, *jacobian*, *jordan*, *kernel*, , *leastsqrs*, *linsolve*, *matadd*, *matrix*, *minor*, *minpoly*, *mulcol*, *mulrow*, *multiply*, *norm*, *normalize*, *nullspace*, *orthog*, *permanent*, *pivot*, *potential*, *randmatrix*, *randvector*, *rank*, *ratform*, *row*, *rowdim*, *rowspace*, *rowspan*, *rref*, *scalarmul*, *singularvals*, *smith*, *stack*, *submatrix*, *subvector*, *sumbasis*, *swapcol*, *swaprow*, *sylvester*, *toeplitz*, *trace*, *transpose*, *vandermonde*, *vecpotent*, *vectdim*, *vector*, *wronskian*]

Vamos começar vendo como definir matrizes.

2 Definindo uma Matriz

Os comandos do pacote *linalg* para definir matrizes são: *matrix, entermatrix, genmatrix, randmatrix, band* e *diag*. A título de exemplo, vamos definir duas matrizes e gravá-las nas variáveis A e B:

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

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

```
> B := matrix(3, 2, [a,1,1,d,e,1] );
```

$$B := \begin{bmatrix} a & 1 \\ 1 & d \\ e & 1 \end{bmatrix}$$

Na primeira matriz, entramos os elementos fornecendo cada linha na forma de uma lista. Neste caso, não é necessário especificar as dimensões da matriz. Na segunda, primeiro estipulamos as dimensões da matriz como sendo *3x2*, depois fornecemos todos os elementos numa única lista. O próprio Maple separa as linhas de acordo com as dimensões da matriz.

Podemos também fornecer os elementos da matriz interativamente. Primeiro, gravamos uma matriz com dimensão especificada em uma determinada variável, e depois usamos o comando *entermatrix*:

```
> C:=matrix(2,2);
```

$$C := array(1 \,..\, 2,\ 1 \,..\, 2,\ [])$$

```
> entermatrix(C);
> 1/2;
> 1/3;
> 1/5;
> 1/6;
```

$$\begin{bmatrix} \frac{1}{2} & \frac{1}{3} \\ \frac{1}{5} & \frac{1}{6} \end{bmatrix}$$

Após a matriz ter sido definida, é possível trocar um elemento. Temos que atribuir o novo valor ao elemento correspondente. Por exemplo, vamos trocar 1/6 por 1/7 na posição **<2,2>** da matriz *C:*

```
> C[2,2] := 1/7;
```

$$C_{2,2} := \frac{1}{7}$$

Vamos verificar que a mudança foi feita com sucesso. Para ver os elementos de uma matriz temos que usar algum comando de avaliação, por exemplo *evalm*:

```
> evalm(C);
```

$$\begin{bmatrix} \frac{1}{2} & \frac{1}{3} \\ \frac{1}{5} & \frac{1}{7} \end{bmatrix}$$

As operações de soma e potenciação de matrizes são feitas com os operadores "+" e "^" usuais de soma e potenciação de números. A multiplicação de matrizes, não sendo comutativa, é feita pelo operador "&*" (ampersand vezes). As expressões matriciais devem ser envolvidas pelo comando *evalm*, acrônimo de *evaluate in matrix context*. Por exemplo:

```
> evalm((A&*B + C)^(-1));
```

$$\begin{bmatrix} -30\,\frac{71+35\,d}{\%1} & 70\,\frac{13+6\,d}{\%1} \\ 84\,\frac{10\,a+13+15\,e}{\%1} & -105\,\frac{2\,a+5+6\,e}{\%1} \end{bmatrix}$$

$$\%1 := 1510\,a + 630\,a\,d - 593 - 441\,d - 930\,e - 630\,e\,d$$

A potenciação por um número negativo, quer dizer a inversão da matriz, e subsequente potenciação pelo módulo do número. A inversa também pode ser encontrada através do comando *inverse*:

```
> inverse(matrix([[a,b],[c,d]]));
```

$$\begin{bmatrix} \frac{d}{a\,d-b\,c} & -\frac{b}{a\,d-b\,c} \\ -\frac{c}{a\,d-b\,c} & \frac{a}{a\,d-b\,c} \end{bmatrix}$$

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. Suponha que queiramos definir uma matriz de dimensão *3x4*, onde o elemento *<i,j>* é dado por $\frac{i}{j}$:

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

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

3 Matrizes Especiais

Existem várias matrizes especiais que são usadas com frequência em Álgebra Linear. Muitas delas têm comandos específicos para gerá-las. Por exemplo, as matrizes diagonais quadradas podem ser

geradas através do comando *diag*. Neste caso, é bem mais econômico entrar os elementos através deste comando do que com o comando *matrix*, pois neste último, teríamos que fornecer os zeros fora da diagoanal. Vejamos alguns exemplos:

```
> diag(1,2,3,4);
```

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 4 \end{bmatrix}$$

```
> diag(a$3);
```

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

O comando $a\,\$\,3$ gera uma sequência de três elementos a, de forma que o último comando dado acima é equivalente a *diag*(a,a,a). Podemos também criar matrizes diagonais em bloco. Vamos usar a matriz C, definida acima com o comando *entermatrix,* para criar a seguinte matriz:

```
> diag(C,C);
```

$$\begin{bmatrix} \frac{1}{2} & \frac{1}{3} & 0 & 0 \\ \frac{1}{5} & \frac{1}{7} & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{3} \\ 0 & 0 & \frac{1}{5} & \frac{1}{7} \end{bmatrix}$$

Um caso particular de matriz diagonal é a matriz identidade. Ela pode ser criada com o comando *diag*, da seguinte forma: *diag(1$n)*, onde n é a dimensão da matriz identidade. Existem outras formas não equivalentes de definir a matriz identidade. Podemos definí-la com o comando *array* com a funç o de indexação *identity*. Por exemplo:

```
> ID := array(identity, 1..3, 1..3);
```

$$ID := array(identity, 1 \,..\, 3, 1 \,..\, 3, [])$$

```
> evalm(ID);
```

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

Existe uma forma mais abstrata da matriz identidade no Maple que é "&*( )". Esta forma assume a repesentação usual dependendo do contexto. Por exemplo:

```
> evalm(C - &*()*lambda);
```

$$\begin{bmatrix} \frac{1}{2} - \lambda & \frac{1}{3} \\ \frac{1}{5} & \frac{1}{7} - \lambda \end{bmatrix}$$

Neste exemplo, a matriz "&*( )" assumiu a dimensão 2 porque ela está somada a matriz C que tem dimensão 2. Na maioria dos casos, não é necessário usar a matriz identidade, pois o Maple assume que, quando um número está somando a uma matriz, este número está multiplicado pela matriz identidade de dimensão conveniente. De forma que o comando acima é equivalente ao comando *evalm(C-lambda)*.

Vamos ver outras matrizes especiais. Se a matriz for uma faixa em torno da diagonal, podemos usar o comando *band:*

```
> band([-1,1,2],4);
```

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

O último argumento é a dimensão da matriz. O primeiro argumento é a lista dos elementos da faixa.

A matriz de Toeplitz é gerada pelo comando *toeplitz:*

```
> toeplitz([alpha,1,beta,2]);
```

$$\begin{bmatrix} \alpha & 1 & \beta & 2 \\ 1 & \alpha & 1 & \beta \\ \beta & 1 & \alpha & 1 \\ 2 & \beta & 1 & \alpha \end{bmatrix}$$

O comando *hilbert* cria a matriz de Hilbert. O comando *sylvester* cria a matriz de Sylvester a partir de dois polinômios, e o comando *frobenius* cria a matriz na forma canônica racional de outra matriz. Podemos citar ainda os comandos como *hessian, hermite* e *smith* entre outros.

4 Autovalores e Autovetores

Os autovetores e autovalores de uma matriz podem ser obtidos a partir dos comandos *eigenvetcs* e *eigenvalues*, respectivamente. Como exemplo, vamos definir uma matriz quadrada e gravá-la na variável *A:*

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

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

O polinômio característico é obtido com o comando *charpoly:*

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

$$x^2 - \varepsilon$$

O comando *eigenvects* fornece uma sequência de listas, onde os elementos destas listas são o autovalor, a sua multiplicidade e os autovetores dentro de um conjunto:

```
> autovetor := eigenvects(A);
```

$$autovetor := [\sqrt{\varepsilon},\ 1,\ \{\left[\frac{1}{\sqrt{\varepsilon}},\ 1\right]\}],\ [-\sqrt{\varepsilon},\ 1,\ \{\left[-\frac{1}{\sqrt{\varepsilon}},\ 1\right]\}]$$

Vamos selecionar os autovetores:

```
> av1 := autovetor[1][3][1];
```

$$av1 := \left[\frac{1}{\sqrt{\varepsilon}},\ 1\right]$$

```
> av2 := autovetor[2][3][1];
```

$$av2 := \left[-\frac{1}{\sqrt{\varepsilon}},\ 1\right]$$

5 Manipulação Estrutural de Matrizes

Os principais comandos para manipulação estrutural com matrizes são: *addcol, addrow, augment, col, row, coldim, rowdim, concat, copyinto, delcols, delrows, extend, mulrow, mulcol, stack, submatrix, swapcol* e *swaprow*. A maioria dos nomes dos comandos falam por si só. As terminações ou prefixos *row* e *col* se referem a linha e coluna, respectivamente. O comando *coldim*, por exemplo, fornece o número de colunas da matriz. O comando *swaprow* troca duas linha de uma matriz. Vejamos alguns exemplos. Primeiro, vamos criar duas matrizes genéricas A e B:

```
> A := matrix(2,3, (i,j) -> A.i.j);
```

$$A := \begin{bmatrix} A11 & A12 & A13 \\ A21 & A22 & A23 \end{bmatrix}$$

```
> B := matrix(2,3, (i,j) -> beta.i.j);
```

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

Podemos juntar as matrizes A e B lateralmente com o comando *augment*, e verticalmente com o comando *stack:*

```
> augment(A,B);
```

$$\begin{bmatrix} A11 & A12 & A13 & \beta 11 & \beta 12 & \beta 13 \\ A21 & A22 & A23 & \beta 21 & \beta 22 & \beta 23 \end{bmatrix}$$

```
> stack(A,B);
```

$$\begin{bmatrix} A11 & A12 & A13 \\ A21 & A22 & A23 \\ \beta 11 & \beta 12 & \beta 13 \\ \beta 21 & \beta 22 & \beta 23 \end{bmatrix}$$

Podemos extrair uma sub-matriz de uma matriz com o comando *submatrix:*

```
> submatrix(",2..3,1..2);
```

$$\begin{bmatrix} A21 & A22 \\ \beta 11 & \beta 12 \end{bmatrix}$$

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

```
> mulcol(A,1,alpha);
```

$$\begin{bmatrix} \alpha\, A11 & A12 & A13 \\ \alpha\, A21 & A22 & A23 \end{bmatrix}$$

Usando o último resultado, podemos apagar uma ou mais linha com o comando *delrows:*

```
> delrows(",2..2);
```

$$\begin{bmatrix} \alpha\, A11 & A12 & A13 \end{bmatrix}$$

6 Cálculo Vetorial

As operações gradiente, divergente, rotacional e laplaciano estão programadas no Maple como as funções *grad, diverge, curl* e *laplaciano*. 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 e o laplaciano desta função:

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

$$I,\ f$$

```
> 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);
```

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

```
> laplaciano(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,z):*

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

$$I, f, g, h$$

```
> 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*.

Capítulo 5

Gráficos

1 Introdução

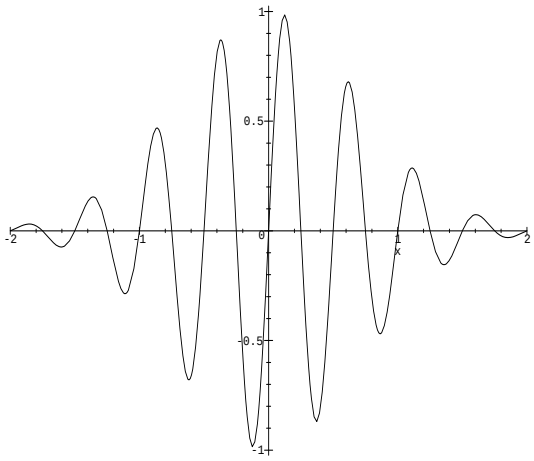
Vamos ver alguns exemplos de gráficos: considere a seguinte expressão como uma função de x:

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

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

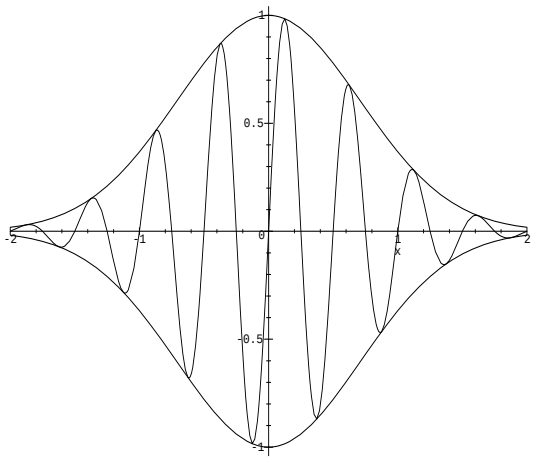
O seu gráfico no intervalo $[-2\ ..\ 2]$ é:

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



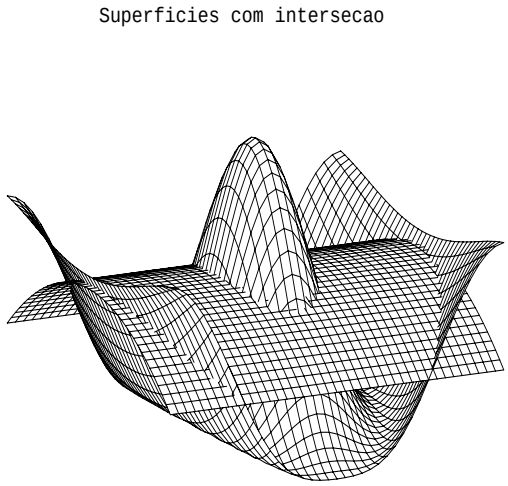
Vamos agora fazer o gráfico de F junto com as envoltórias:

```
> plot( {F, exp(-x^2), -exp(-x^2)}, x=-2..2);
```



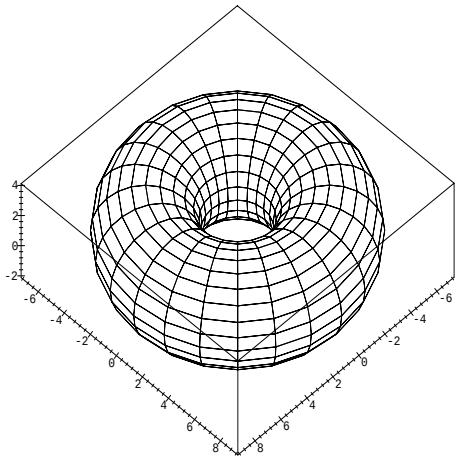
Vamos fazer o gráfico de duas superfícies que têm interseção:

```
> 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‘,
> color=black);
```



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:

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



2 Gráficos em 2 Dimensões

2.1 Introdução

A sintaxe para gerar gráficos em 2 dimensões é:

$$\textbf{\textit{plot(f(x), x = a .. b, opções)}}$$

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:

$$\textbf{\textit{nome da opção = tipo da opção}}$$

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

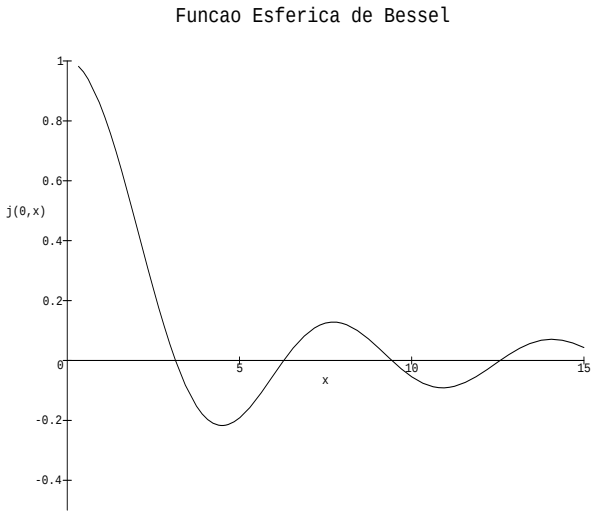
As opções servem para se obter os efeitos desejados em cada gráfico. Vamos ver alguns exemplos com vários tipos de opções. Primeiro, vamos definir a função esférica de Bessel a partir da função cilíndrica de Bessel:

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

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

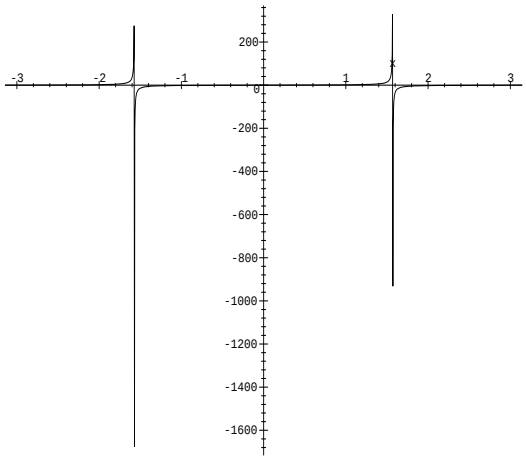
Agora, vamos fazer o gráfico da função de ordem zero com o título "Função Esférica de Bessel", com o nome "j(0,x)" no eixo vertical e controlando o número de marcações no eixo x e no eixo y:

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



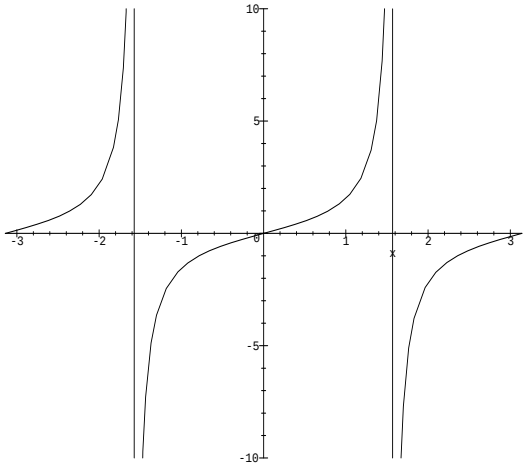
Em várias situações, as opções são necessárias, pois sem elas, o gráfico pode ficar ilegível. Por exemplo, o gráfico da função tangente sem opções:

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



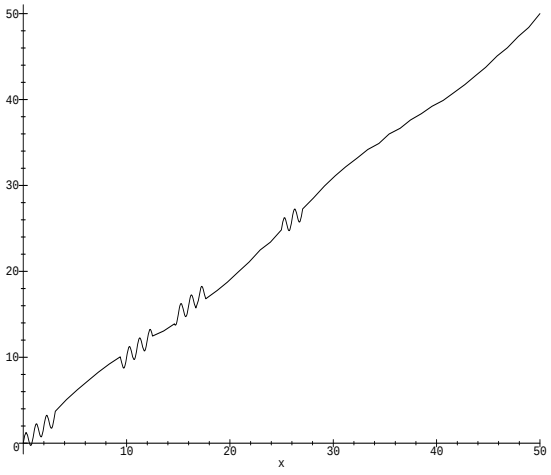
Aqui, precisamos da opção que limita o intevalo do eixo vertical:

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



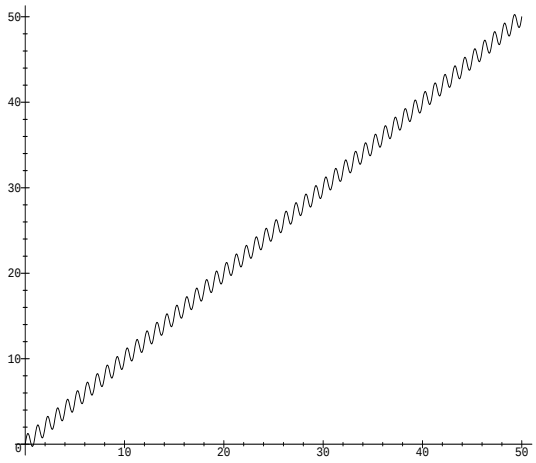
Vejamos outro exemplo. O gráfico da função a função $x + sin(2\,\pi\,x)$ deve apresentar 50 máximos no intervalor $x = 0$ .. 50. Porém, vejamos o resultado:

```
> 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.

2.2 Gráficos de Funções Parametrizadas

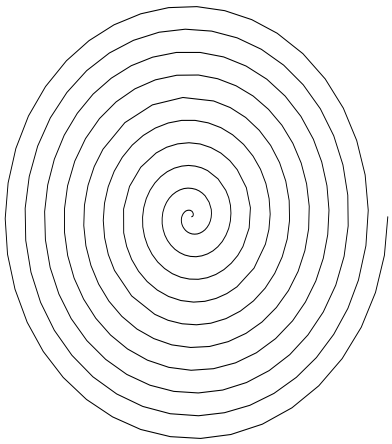
A sintaxe dos gráficos de funções parametrizadas é:

$$\textbf{\textit{plot([x(t), y(t), t = a .. b], opções)}}$$

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 a opção *scaling=constrained* serve para mostrar a característica circular desta figura.

2.3 Gráficos em Coordenadas Polares

A sintaxe dos gráficos de funções em coordenadas polares é:

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

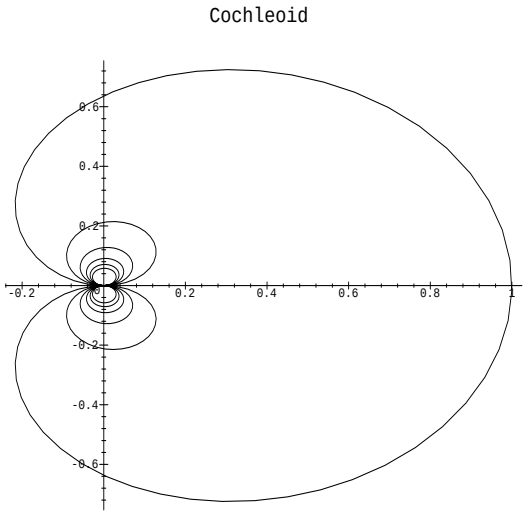
onde as opções são da mesma forma do comando *plot* usual. A lista completa pode ser obtida com o comando *?plot,options*.

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‘);
```



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:

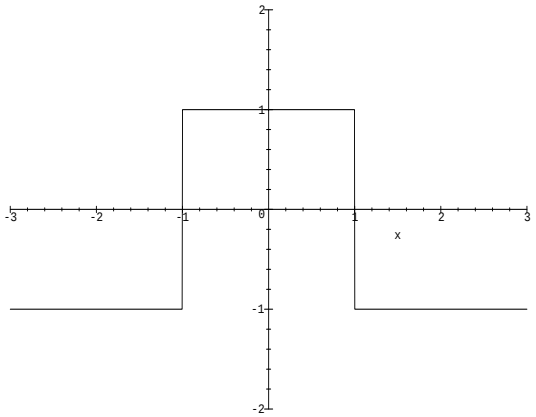
$$f(x) = \begin{cases} -1 & x < -1 \\ 1 & x \leq 1 \\ -1 & 1 < x \end{cases}$$

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

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

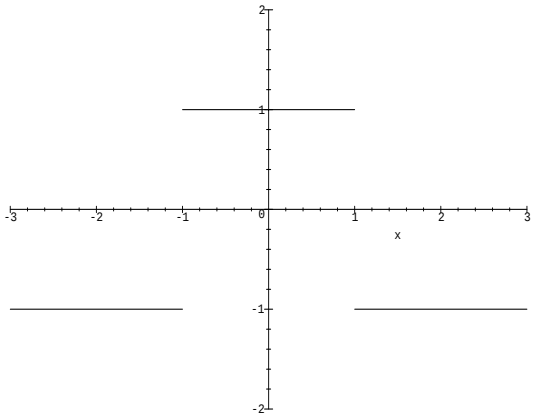
$$F := \begin{cases} -1 & x < -1 \\ 1 & x \leq 1 \\ -1 & otherwise \end{cases}$$

```
> 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*:

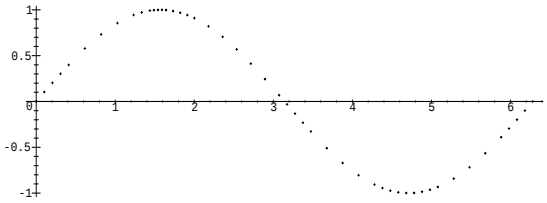
```
> plot( F, x=-3..3, -2..2, scaling=constrained, discont=true, color=black);
```



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 := ";
```

$$sin\_plot := PLOT(CURVES([[0, 0], [.1027166692000000, .1025361420919374],$$
$$[.2054333383000000, .2039914054009955], [.3081500075000000, .3032963054594041],$$
$$[.4108666765000000, .3994040252463781], [.6163000148000000, .5780198520145460],$$
$$[.8217333530000001, .7323272609428949], [1.027166692000000, .8558369097491199],$$
$$[1.232600030000000, .9433546433863553], [1.335316699000000, .9724025515092569],$$
$$[1.438033368000000, .9911999356157914], [1.489391703000000, .9966884729300440],$$
$$[1.540750037000000, .9995486441925453], [1.592108372000000, .9997729069603165],$$
$$[1.643466706000000, .9973606698215630], [1.732839931000000, .9868996387583718],$$
$$[1.822213156000000, .9685609201733348], [1.911586381000000, .9424908984493119],$$
$$[2.000959605000000, .9088976716423971], [2.179706055000000, .8202721118009889],$$
$$[2.358452503000000, .7055083237407663], [2.537198953000000, .5682632892145895],$$
$$[2.715945402000000, .4129103666027981], [2.894691851000000, .2443999250339089],$$
$$[3.073438299000000, .06810160383087029],$$
$$[3.173947839000000, -.03234954049144816],$$
$$[3.274457377000000, -.1324741573740404],$$
$$[3.374966916000000, -.2312616253829775],$$
$$[3.475476453000000, -.3277148139408290],$$

[3.676495530000000, −.5097574846467023],
[3.877514606000000, −.6712708031938160],
[4.078533683000000, −.8057501807167919],
[4.279552759000000, −.9077797574487739],
[4.380062298000000, −.9452858383268380],
[4.480571836000000, −.9732505196271812],
[4.581081374000000, −.9913915356273051],
[4.681590911000000, −.9995257769470791],
[4.781690562000000, −.9975996063229529],
[4.881790212000000, −.9856858912676929],
[4.981889862000000, −.9639039068772925],
[5.081989511000000, −.9324717255078590],
[5.282188811000000, −.8420089761101488],
[5.482388110000000, −.7179112755161254],
[5.682587410000000, −.5651358382775158],
[5.882786709000000, −.3897854445989344],
[5.982886360000000, −.2958057886008316],
[6.082986009000000, −.1988646523330561],
[6.183085659000000, −.0999325665047544 3],
[6.283185308000000, .8204137140694137 10^{-9}]]), $SCALING(CONSTRAINED)$,
$STYLE(POINT)$, $COLOUR(RGB, 0, 0, 0)$)

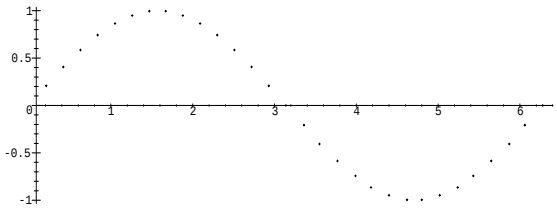
Estes pontos não são igualmente espaçados, pois nas regiões onde a curvatura do gráfico é maior, o Maple escolhe mais pontos, como podemos verificar no gráfico anterior. Vamos fazer os mesmo gráfico, porém vamos nós mesmos escolher os pontos de forma que fiquem igualmente espaçados. Vamos gerar estes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

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

sin_points := [.209, .207], [.418, .406], [.628, .585], [.838, .743], [1.05, .865], [1.26, .948], [1.47, .995], [1.67, .995], [1.88, .948], [2.09, .865], [2.30, .743], [2.51, .585], [2.72, .406], [2.93, .207], [3.14, 0], [3.36, −.207], [3.55, −.406], [3.77, −.585], [3.99, −.743], [4.18, −.865], [4.40, −.948], [4.62, −.995], [4.80, −.995], [5.02, −.948], [5.24, −.865], [5.43, −.743], [5.65, −.585], [5.87, −.406], [6.06, −.207], [6.28, 0]

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);
```

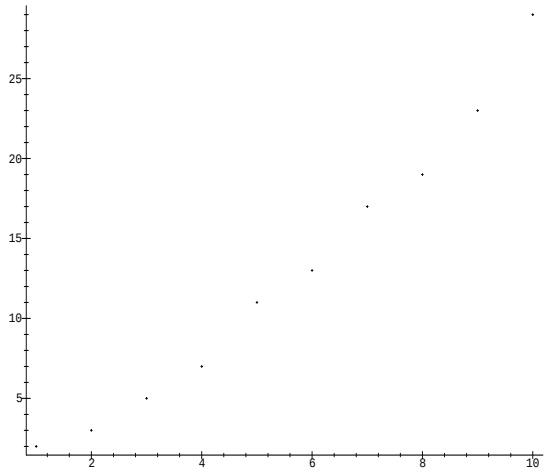


Vejamos outro exemplo:

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

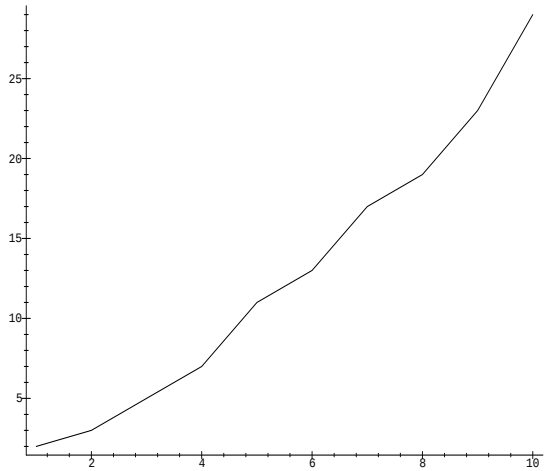
pontos := [1, 2], [2, 3], [3, 5], [4, 7], [5, 11], [6, 13], [7, 17], [8, 19], [9, 23], [10, 29]

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



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

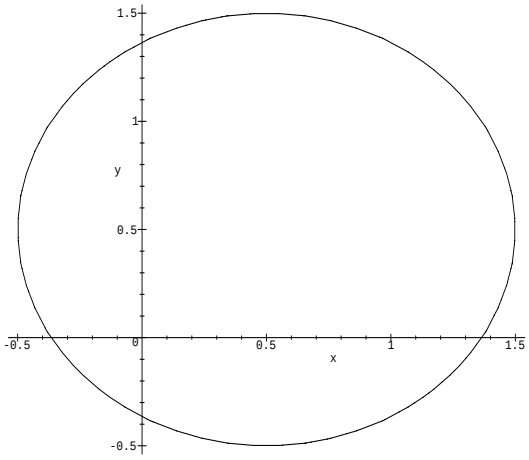
```
> plot( [pontos]);
```



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);
```



3 Gráficos em 3 Dimensões

3.1 Introdução

A sintaxe dos gráficos em 3 dimensões é:

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

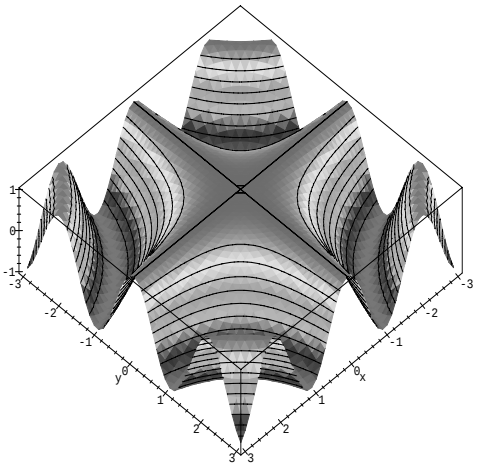
onde ***f(x,y)*** é uma função de duas variáveis, ***a .. b*** é o intervalo para o eixo ***x*** e ***c .. d*** é o intervalo para o eixo ***y***. As opções são da forma

$$\textbf{\textit{nome da opção = tipo da opção}}$$

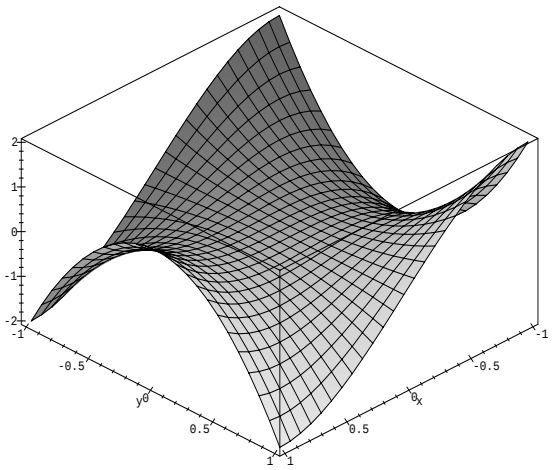
A lista completa dos nomes das opções e seus tipo pode ser obtida através do *help on line* com o comando *?plot3d,options*.

Vejamos alguns exemplos:

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




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



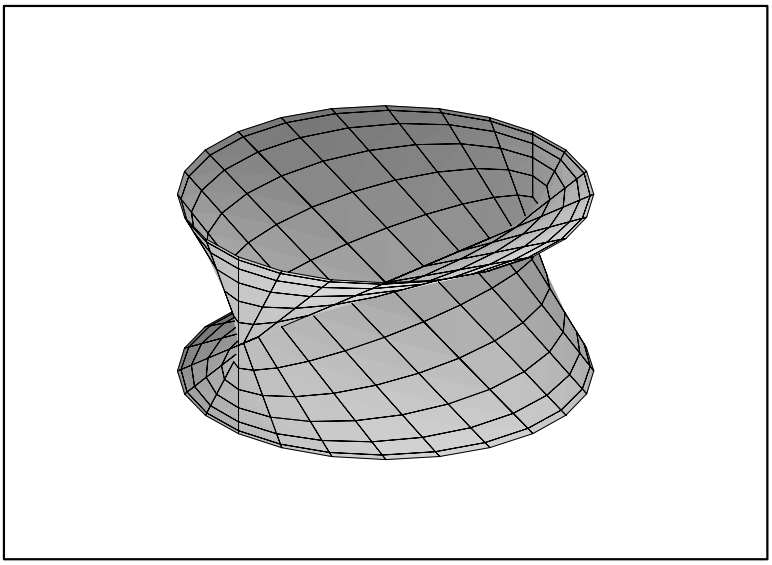
3.2 Gráficos de Funções Parametrizadas

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);
```



3.3 Gráficos em Coordenadas Esféricas

A sintaxe dos gráficos de funções em coordenadas esféricas é:

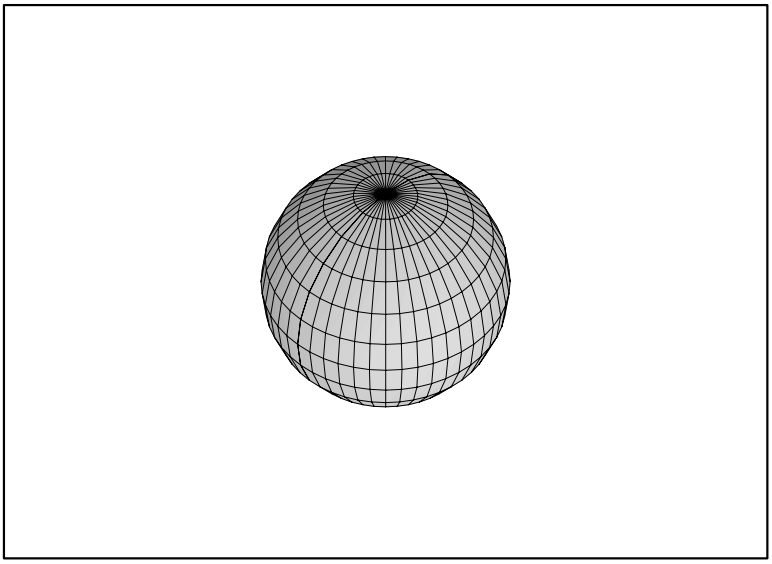
sphereplot(r(theta,phi), theta = a .. b, phi = c .. d, opções)

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)

Vamos ver um exemplo do primeiro caso:

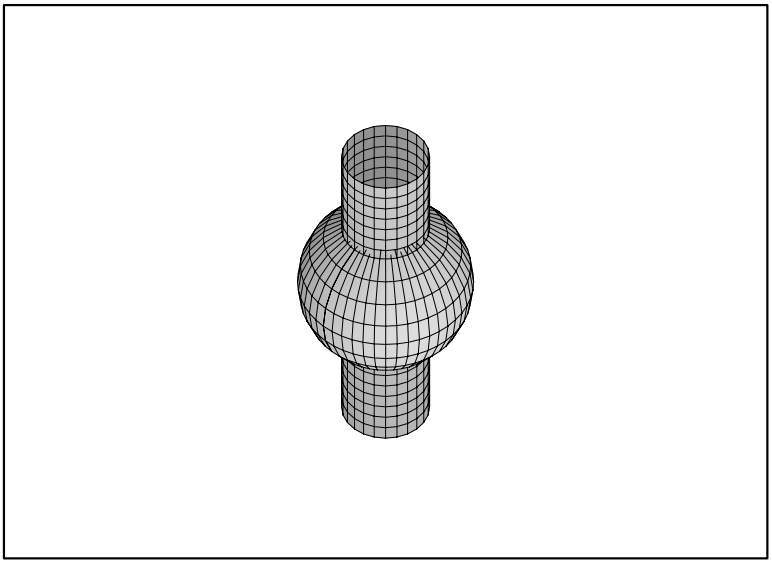
```
> with(plots):
> sphereplot(1, theta=0..Pi, phi=0..2*Pi, scaling=constrained);
```



4 Exibindo vários Gráficos Simultaneamente

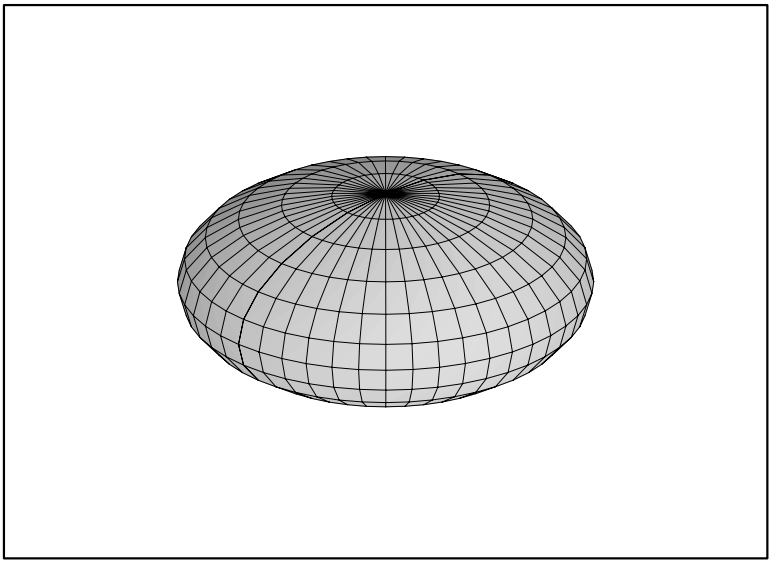
Para exibir dois ou mais gráficos simultaneamente, podemos usar o comando *plot* ou *plot3d* colocando as expressões de cada gráfico dentro de um conjunto, isto é, entre chaves. Vimos um exemplo no início deste capítulo. Desta forma, todas a opções usadas serão comuns a todos os gráficos. Para fazer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

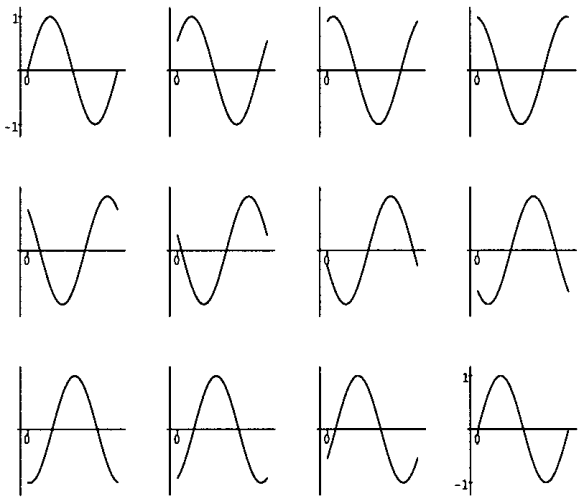
```
> G1;
```



5 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:

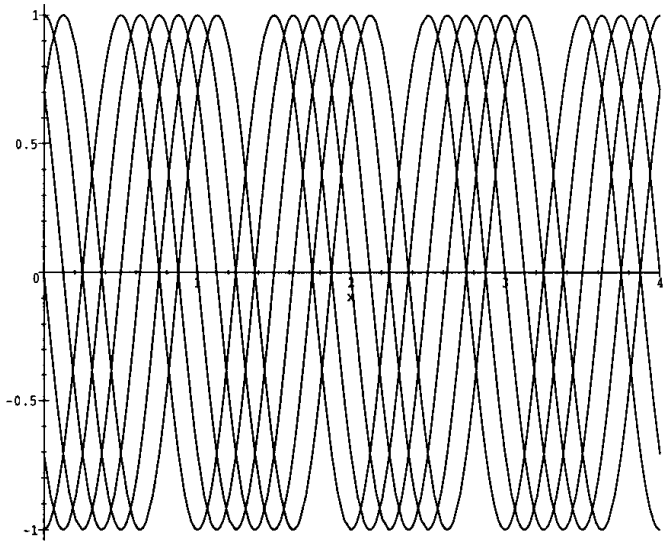
```
> with(plots):
> animate( sin(2*Pi*(x+t)),x=0..1,t=0..1,frames=12);
```



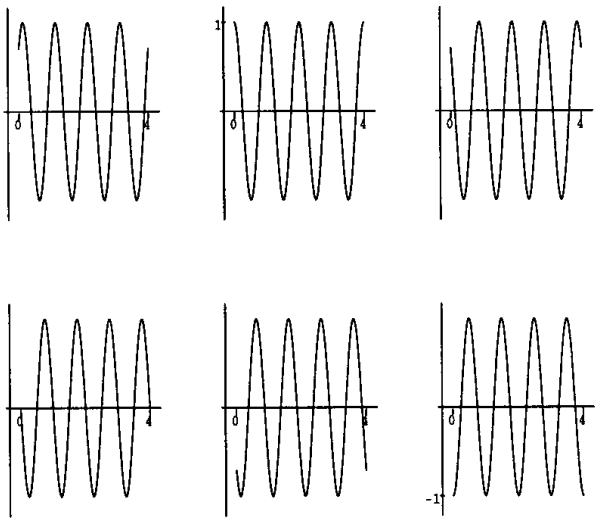
Para disparar a animação devemos clicar no gráfico e depois na tecla *play*. O comando *animate* gera tantos gráficos quanto for o valor da opção *frames*. Os valores que a variável t assume, depende do valor de *frames*. No caso acima, t assumiu os valores $t = 0$, $t = \frac{1}{11}$ e $t = \frac{10}{11}$ 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 atribuido 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 do od* é muito útil neste contexto, pois podemos gerar vários gráficos 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)
> od:
> 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
```



6 Colocando Textos em Gráficos

Os comandos usuais de gerar gráficos colocam uma série de textos no gráfico. 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'])

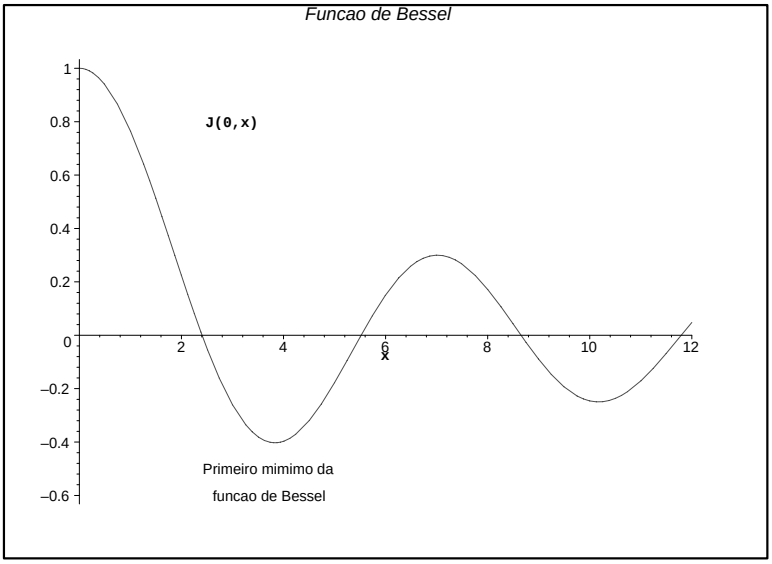
e

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 gráfico 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 imprimir letras gregas é necessário usar a fonte *symbol*.

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*. 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.*

8 Manipulando Gráficos

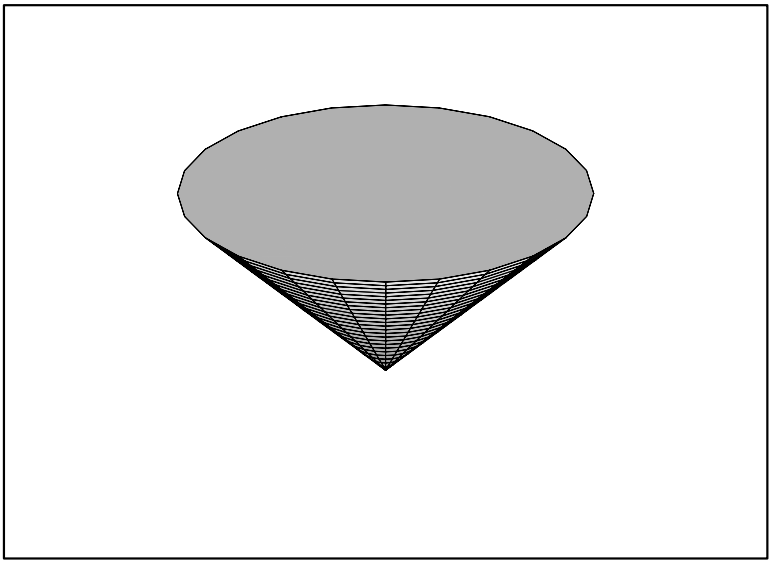
O pacote *plottools* tem vários comandos para manipulação com gráficos além de diversos comandos para criar objetos gráficos. Estes comandos não mostram o gráfico. O comando *display* do pacote

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);
```

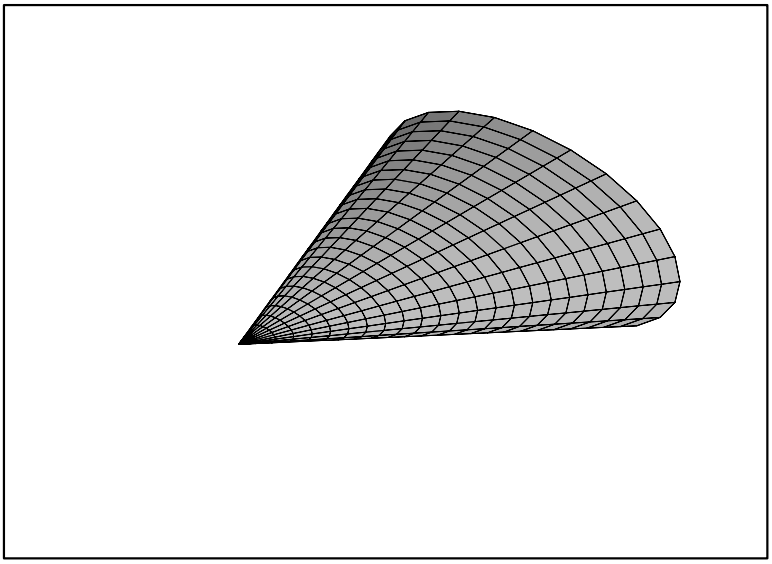
[*arc*, *arrow*, *circle*, *cone*, *cuboid*, *curve*, *cutin*, *cutout*, *cylinder*, *disk*, *dodecahedron*, *ellipse*, *ellipticArc*, *hemisphere*, *hexahedron*, *hyperbola*, *icosahedron*, *line*, *octahedron*, *pieslice*, *point*, *polygon*, *rectangle*, *rotate*, *scale*, *semitorus*, *sphere*, *stellate*,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

```
> C := cone([0,0,0], 1/2, 2, color=black):
> 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));
```



Capítulo 6

Cálculo Diferencial e Integral

1 Funções Matemáticas

1.1 Definindo uma Função

O Maple tem uma grande variedade de funções definidas internamente, incluindo as funções trigonométricas e inversas, hiperbólicas e inversas, 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, que podem expressas em termos das funções internas do Maple. 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.

Primeiramente, é importante distinguir função de expressão algébrica. Por exemplo:

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

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

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 $f(\,\pi)$:

```
> f(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, f);
```

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

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

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

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

Neste caso especificamos que a expressão $sin(a\,x)$ é uma função de x. Até agora, o Maple não tinha condições de saber se queríamos uma função na variável x ou na variável a. 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 $x \rightarrow sin(a\,x)$ enquanto que $sin(a\,x)$ é uma expressão algébrica. Vejamos:

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

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

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. Por exemplo:

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

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

O comando $D(sin(a\,x))$ 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.
```

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

Uma outra maneira de definir uma função é usar diretamente o operador seta (->). Esta forma não é necessariamente equivalente ao comando *unapply*. Por exemplo:

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

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

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

$$5$$

ATENÇÃO: Quando usar o operador seta (->), a expressão da função deve ser dada explicitamente. Não podemos usar ", "" ou """ dentro do operador seta.

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:

```
> 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);
```

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

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

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

```
> 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);
```

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

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

$$80$$

Um erro muito comum é tentar definir uma uma função da seguinte forma: vamos supor que desejamos definir a função $h(x) = x^2$. Muitos iniciantes usam a seguinte forma:

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

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

Este comando faz com o Maple guarde na memória que o valor da função para x é x^2. Porém, para nenhum outro argumento, esta função foi definida, por exemplo:

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

$$h(2)$$

Observe que o comando $h(2)$ não retornou 4. A maneira correta de definir a função h é usando o comando *unapply* ou o operador seta.

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;
```

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

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

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

Vejamos outros exemplos:

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

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

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

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

```
> (g1@g2)(x); # composição
```

$$g1(g2(x))$$

```
> (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. Vejamos outros exemplos:

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

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

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

$$a^{(a^{(a^{(a^{(a^x)})})})}$$

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

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

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

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

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

$$.6250000000$$

```
> evalf((G@@100)(1)); # razão áurea (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);
```

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

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(");
```

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

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(");
```

$$-.06759071137$$

Capítulo 7

Equações Diferenciais

1 Equações Diferenciais Ordinárias

1.1 Introdução

Para resolver uma equação diferencial ordinária sem condicões iniciais a sintaxe é

dsolve(EDO, f(t), método, explicit)

onde o *método* pode ser *exact, laplace, series* ou *numeric*. O método *default* é *exact*. O argumento *explicit* é opcional. Vejamos um exemplo de uma equação linear não-homogênea de segunda ordem:

```
> alias(y=y(x)):
> EDO1 := diff(y, x$2) - y = 1;
```

$$EDO1 := (\frac{\partial^2}{\partial x^2}\, y) - y = 1$$

```
> dsolve( EDO1, y );
```

$$y = -1 + \_C1\, e^{x} + \_C2\, e^{(-x)}$$

_C1 e _C2 são constantes arbitrárias. Vejamos um exemplo de uma equação não linear de primeira ordem:

```
> EDO2 := x*diff(y,x)=y*ln(x*y)-y;
```

$$EDO2 := x\,(\frac{\partial}{\partial x}\, y) = y\, ln(y\, x) - y$$

```
> dsolve( EDO2, y);
```

$$x = \_C1\, ln(y) + \_C1\, ln(x)$$

Note que o resultado está dado implicitamente. Vamos usar agora o opção *explicit*:

```
> dsolve( EDO2, y, explicit);
```

$$y = e^{(\frac{x - \_C1\, ln(x)}{\_C1})}$$

Para encontrar soluções analíticas com condições iniciais, a sintaxe é:

dsolve({EDO, condições iniciais}, f(t), método, explicit)

Considere a seguinte equação diferencial:

```
> EDO3 := diff(v(t),t)+2*t=0;
```

$$EDO3 := (\frac{\partial}{\partial t} v(t)) + 2\,t = 0$$

Vamos resolvê-la com a condição inicial v(1) = 5:

```
> dsolve( { EDO3, v(1)=5 } , v(t) );
```

$$v(t) = -t^2 + 6$$

Quando há condições iniciais não é recomendado usar a função *alias*. Reparemos nas dificuldades para resolver a mesma equação do exemplo anterior usando *alias*:

```
> alias(v=v(t));
```

$$I,\, v$$

```
> EDO4 := diff(v,t)+2*t=0;
```

$$EDO4 := (\frac{\partial}{\partial t} v) + 2\,t = 0$$

```
> infolevel[dsolve]:=1:  # para obter informações
> dsolve( { EDO4, v(1)=5 } , v );

dsolve/diffeq/system/linear:   determining if system is linear
dsolve/diffeq/system:   cannot solve non-linear systems
dsolve/diffeq/system/linear:   determining if system is linear
dsolve/diffeq/system:   cannot solve non-linear systems
dsolve:   Warning: no solutions found

> infolevel[dsolve]:=0:
```

Podemos ver que nenhuma solução foi encontrada. O problema que ocorreu acima é de difícil detecção para um iniciante. Devido ao comando *alias(v=v(t))*, escrever *v* é o mesmo que escrever *v(t)*. Assim, *v(1)* quer dizer *v(t)(1)*. Vamos confirmar:

```
> subs(t=1000, v(1));
```

$$v(1000)(1)$$

Quando se usa um *alias*, a condição inicial tem que ser dada da seguite forma:

```
> dsolve( { EDO4, subs(t=1,v) = 5 } , v );
```

$$v = -t^2 + 6$$

Agora, vamos ver um exemplo onde as condições iniciais envolvem a derivada da função. Por exemplo: *y(0)=0* e *y'(0)=1*.

```
> alias(y=y); # para tirar o apelido de y
```

$$I,\ v$$

```
> EDO5 := diff( y(t),t$2) + 5*diff(y(t),t) + 6*y(t) =0;
```

$$EDO5 := (\frac{\partial^2}{\partial t^2}\, y(t)) + 5\,(\frac{\partial}{\partial t}\, y(t)) + 6\, y(t) = 0$$

```
> sol5 := dsolve( {EDO5, y(0)=0, D(y)(0)=1 }, y(t) );
```

$$sol5 := y(t) = e^{(-2\,t)} - e^{(-3\,t)}$$

Observe que usamos a notação *D(y)(0)=1* para *y'(0)=1*. A função *diff* não é apropriada para descrever essa condição inicial. Vamos confirmar que solução está correta:

```
> expand(subs(sol5, EDO5));
```

$$0 = 0$$

Vamos confirmar que *y(0)=0*:

```
> subs(t=0,sol5);
```

$$y(0) = 0$$

Vamos confirmar que *y'(0)=1*:

```
> unapply(rhs(sol5),t); # para criar uma funcao
```

$$t \rightarrow e^{(-2\,t)} - e^{(-3\,t)}$$

```
> D(")(0);
```

$$1$$

1.2 Método Numérico

Podemos resolver uma equação diferencial sem parâmetros em aberto 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. Vejamos um exemplo:

```
> EDO1 := diff(y(x),x,x) - (1-y(x)^2)*diff(y(x),x) + y(x) =0;
```

$$EDO1 := (\frac{\partial^2}{\partial x^2}\, y(x)) - (1 - y(x)^2)\,(\frac{\partial}{\partial x}\, y(x)) + y(x) = 0$$

```
> ci := y(0)=0, D(y)(0)=-0.1;
```

$$ci := y(0) = 0,\ D(y)(0) = -.1$$

```
> F := dsolve( { EDO1, 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);
```

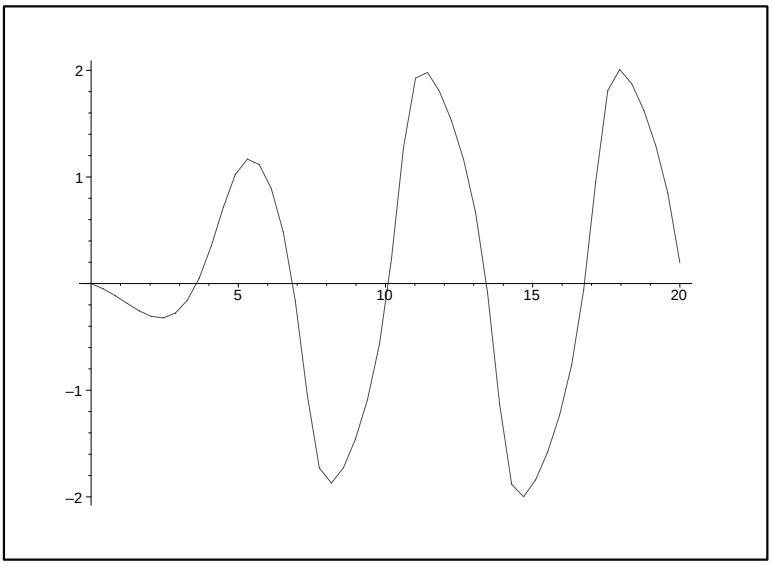
$$[x = 0,\ y(x) = 0,\ \frac{\partial}{\partial x}\, y(x) = -.1]$$

```
> F(1);
```

$$[x = 1,\ y(x) = -.1447686096006437,\ \frac{\partial}{\partial x}\, y(x) = -.1781040958088073]$$

Para fazer o gráfico, podemos usar o comando *odeplot* do pacote *plots*:

```
> plots[odeplot](F,[x,y(x)], 0..20);
```



1.3 Método de Séries

É possível encontrar uma expansão em séries para a solução de uma equação diferencial. O método usado é conhecido como Método de Frobenius. Vejamos um exemplo:

```
> EDO2 := 2*x*diff(y(x),x,x) + diff(y(x),x) +y(x) = 0;
```

$$EDO2 := 2\,x\,(\frac{\partial^2}{\partial x^2}\, y(x)) + (\frac{\partial}{\partial x}\, y(x)) + y(x) = 0$$

```
> dsolve(EDO2, y(x), series);
```

$$y(x) = \_C1\,\sqrt{x}\,(1 - \frac{1}{3}\,x + \frac{1}{30}\,x^2 - \frac{1}{630}\,x^3 + \frac{1}{22680}\,x^4 - \frac{1}{1247400}\,x^5 + O(x^6))$$
$$+ \_C2\,(1 - x + \frac{1}{6}\,x^2 - \frac{1}{90}\,x^3 + \frac{1}{2520}\,x^4 - \frac{1}{113400}\,x^5 + O(x^6))$$

A ordem da solução pode ser controlada através da variável *Order*.

1.4 Método de Séries de Potência

Além do comando *dsolve*, existem outros comandos dentro de alguns pacotes que resolvem equações diferenciais ou que servem para manipulá-las. Podemos citar o pacote *powseries* e *DEtools*. O pacote *powseries* usa o método de séries de potência para resolver equações diferenciais ordinárias lineares. Este pacote trata de um conjunto menor de equações comparado com o método de séries do comando *dsolve*, no entanto ele fornece mais informações sobre a solução, como a fórmula de recorrência dos coeficientes da séries de potência. Vejamos um exemplo:

```
> EDO3 := x*diff(y(x),x,x) + diff(y(x),x) + 4*x^2*y(x) = 0;
```

$$EDO3 := x\,(\frac{\partial^2}{\partial x^2}\,y(x)) + (\frac{\partial}{\partial x}\,y(x)) + 4\,x^2\,y(x) = 0$$

```
> ci:=y(0)=1,D(y)(0)=0;
```

$$ci := y(0) = 1,\ D(y)(0) = 0$$

```
> with(powseries);
```

[*compose*, *evalpow*, *inverse*, *multconst*, *multiply*, *negative*, *powadd*, *powcos*, *powcreate*, *powdiff*, *powexp*, *powint*, *powlog*, *powpoly*, *powsin*, *powsolve*, *powsqrt*, *quotient*, *reversion*, *subtract*, *tpsform*]

```
> sol := powsolve({EDO3,ci}):
```

A solução é um procedimento que deve ser usado nos outros comandos do pacote para se obter o resultado desejado. Por exemplo, a expansão em série de potência até ordem 15 é:

```
> tpsform(sol,x,15);
```

$$1 - \frac{4}{9}\,x^3 + \frac{4}{81}\,x^6 - \frac{16}{6561}\,x^9 + \frac{4}{59049}\,x^{12} + O(x^{15})$$

A relação de recorrência dos coeficientes é:

```
> a(_k) = sol(_k);
```

$$a(\_k) = -4\,\frac{a(\_k - 3)}{\_k^2}$$

2 Equações Diferenciais Parciais

O comando para resolver equações diferenciais parciais é *pdesolve*. Vejamos como exemplo a equação de onda:

```
> PDE := diff(u(x,t),t,t)-c^2*diff(u(x,t),x,x)=0;
```

$$PDE := (\frac{\partial^2}{\partial t^2}\, u(x,\, t)) - (\frac{\partial^2}{\partial x^2}\, u(x,\, t)) = 0$$

A solução é dada em termos das funções arbitrárias $\_F1$ e $\_F2$.

```
> pdesolve( PDE , u(x,t) );
```

$$u(x,\, t) = \_F1(t + x) + \_F2(t - x)$$

Podemos fazer gráficos da solução de equações diferenciais parciais de primeira ordem. Considere a seguinte equação diferencial:

```
> EDP1 := diff(f(x,y), x) + cos(2*x)*diff(f(x,y), y) = -sin(y);
```

$$EDP1 := (\frac{\partial}{\partial x}\, f(x,\, y)) + cos(2\, x)\, (\frac{\partial}{\partial y}\, f(x,\, y)) = -sin(y)$$

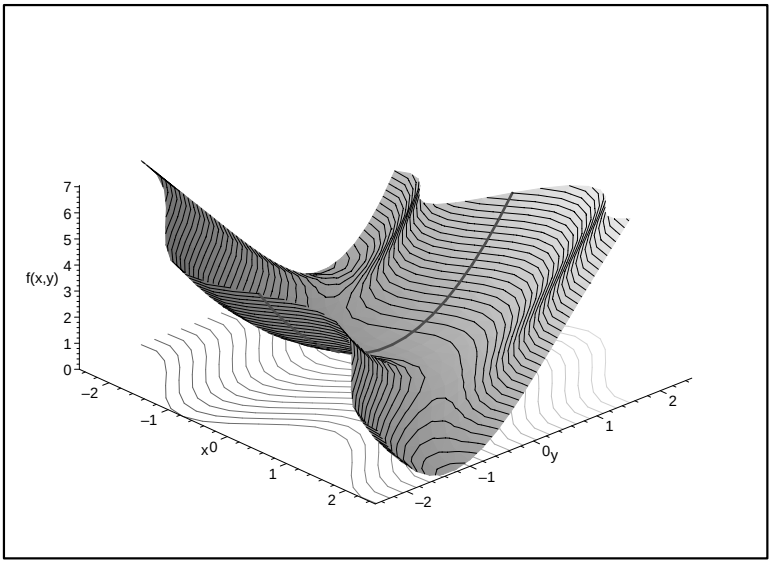
com as seguintes condições iniciais $f(0,\, y) = 1 + y^2$:

```
> ini := [0,s,1+s^2];
```

$$ini := [0,\, s,\, 1 + s^2]$$

O gráfico pode ser gerado com o comando *PDEplot* do pacote *PDEtools*:

```
> with(DEtools):
> PDEplot( EDP1, f(x,y), ini, s=-2..2, basechar=true, initcolor=red, contours=20,
> orientation=[-43,45], style=patchcontour);
```


f(x,y)
7
6
5
4
3
2
1
0
–2
–1
x0
1
2
–2
–1
0y
1
2

Capítulo 8

Introdução à Programação

1 Introdução

A linguagem de programação do Maple e a linguagem que o usuário emprega quando está fazendo um cálculo para resolver um certo problema, são as mesmas. Os comandos para resolver um problema específico poderão ser usados no momento da elaboração de um programa. E vice-versa, qualquer comando de programação pode ser usado no modo interativo. Existem alguns comandos que são frequentemente utilizados em programas e que raramente são utilizados de modo interativo. Por exemplo, o comando *for do od* para gerar iterações, em geral, só é empregado em programação.

O Maple é uma linguagem interativa, porque ele permite que cada comando venha imediatamente acompanhado de sua resposta. Isso não quer dizer que não seja possível agrupar uma série de comandos, cuja resposta é dada por bloco. Na verdade, é isso que faz um programa. Estamos chamando de "uso interativo", o modo comando-resposta que geralmente é empregado para resolver problemas específicos. A título de exemplo, vamos resolver dois problemas simples usando o modo interativo. A partir da solução particular, vamos transformá-la em um programa, que é capaz de resolver o problema proposto e qualquer outro do mesmo tipo.

2 Completando o Quadrado

No primeiro exemplo, vamos ver como completar o quadrado do seguinte polinômio do segundo grau:

```
> polinomio1 := 2*x^2+x-3;
```

$$polinomio1 := 2\,x^2 + x - 3$$

Queremos colocar o *polinomio1* na forma:

```
> polinomio2 := a*(x+b)^2 + c;
```

$$polinomio2 := a\,(x+b)^2 + c$$

Ou seja, queremos obter os coeficientes a, b e c. Podemos construir uma equação e guardar na variável *equacao*:

```
> equacao := polinomio1=polinomio2;
```

$$equacao := 2\,x^2 + x - 3 = a\,(x+b)^2 + c$$

No Maple podemos somar funções e aplicá-las ao argumento. A funções *rhs* e *lhs* fornecem o lado direito e o lado esquerdo de uma equação respectivamente. Assim:

```
> ( rhs - lhs )( equacao ) = 0;
```

$$a\,(x+b)^2 + c - 2\,x^2 - x + 3 = 0$$

Vamos agora fatorar o polinômio acima em relação à variável x:

```
> collect(", x);
```

$$(-2+a)\,x^2 + (2\,a\,b - 1)\,x + a\,b^2 + 3 + c = 0$$

Para obter os valores de a, b e c, temos que igualar a zero os coeficientes do polinômio acima e resolver as equações resultantes em relação a essas variáveis. O comando *coeff* fornece o coeficiente de um único termo especificado pelo segundo argumento, no entanto,como precisamos de todos os coeficientes, vamos usar o comando *coeffs*:

```
> coeffs(lhs("), x);
```

$$a\,b^2 + 3 + c,\ 2\,a\,b - 1,\ -2 + a$$

```
> solve( {"} );
```

$$\{b = \frac{1}{4},\ a = 2,\ c = \frac{-25}{8}\}$$

Na forma como foi usado, o comando solve automaticamente iguala a zero cada termo e resolve as equações para todos os nomes que elas possuem. Assim, obtivemos a solução para o nosso problema. Falta apenas substituir os valores acima na variável *polinomio2*:

```
> subs(", polinomio2);
```

$$2\,(x + \frac{1}{4})^2 - \frac{25}{8}$$

É sempre bom confirmar se fizemos o cálculo correto:

```
> expand(polinomio1 - ");
```

$$0$$

Suponha agora que queremos completar o quadrado de um outro polinômio. Teremos que repetir todos os cálculos acima novamente. Quando queremos usar um conjunto de comandos várias vezes é interessante fazer um programa. No caso acima, queremos construir um programa tal que, dado um polinômio do segundo grau, ele completa o quadrado e apresenta o resultado na tela. Os cálculos intermediários não precisam ser mostrados, a princípio. Queremos agrupar os comandos de alguma forma e usá-los quando necessitarmos. Para criar um bloco de comandos, usamos o comando *proc(...)* no início e *end* no final. Esse bloco pode ser atribuído a uma variável, que será o nome do programa. Toda vez que precisarmos executar o programa, nós o chamamos pelo seu nome. Abaixo, colocamos juntos todos os comandos usados dentro do bloco *proc(polinomio1) ... end*. A entrada deve ser o *polinomio1*:

```
> completarquadrado :=
> proc(polinomio1)
> polinomio2 := a*(x+b)^2+c;
```

```
> equacao := polinomio1 = polinomio2;
> (rhs-lhs)(equacao);
> collect(",x);
> coeffs(",x);
> solve({"});
> subs(",polinomio2)
> end;
Warning, `polinomio2` is implicitly declared local

Warning, `equacao` is implicitly declared local
```

$$
\begin{array}{l}
completarquadrado := proc(polinomio1) \\
\quad localpolinomio2,\ equacao; \\
\qquad polinomio2 := a\ (x+b)^2 + c\,; \\
\qquad equacao := polinomio1 = polinomio2\,; \\
\qquad (rhs - lhs)(equacao)\,; \\
\qquad collect(",\ x)\,; \\
\qquad coeffs(",\ x)\,; \\
\qquad solve(\{"\})\,; \\
\qquad subs(",\ polinomio2) \\
\quad end
\end{array}
$$

O Maple deu dois avisos dizendo que as variáveis *polinomio2* e *equacao* foram declaradas locais. Podemos ver que o Maple também modificou o programa acrescentando o comando:

local polinomio2,equacao;

Esse comando garante que as variáveis *polinomio2* e *equacao* usadas dentro do programa não entrarão em conflito com as variáveis de mesmo nome usadas fora do programa. Podemos acrescentar esse comando no nosso programa para não termos mais as mensagem de aviso:

```
> completarquadrado :=
> proc(polinomio1)
> local polinomio2,equacao;
> polinomio2 := a*(x+b)^2+c;
> equacao := polinomio1 = polinomio2;
> (rhs-lhs)(equacao);
> collect(",x);
> coeffs(",x);
```

```
> solve({"});
> subs(",polinomio2)
> end;
```

$$
\begin{array}{l}
completarquadrado := proc(polinomio1) \\
\quad localpolinomio2,\ equacao; \\
\quad\quad polinomio2 := a\ (x+b)^2 + c\,; \\
\quad\quad equacao := polinomio1 = polinomio2\,; \\
\quad\quad (rhs - lhs)(equacao)\,; \\
\quad\quad collect(",\ x)\,; \\
\quad\quad coeffs(",\ x)\,; \\
\quad\quad solve(\{"\})\,; \\
\quad\quad subs(",\ polinomio2) \\
\quad end
\end{array}
$$

Podemos agora testar o programa para o polinômio do nosso exemplo:

```
> completarquadrado(2*x^2+x-3);
```

$$2\,(x+\frac{1}{4})^2 - \frac{25}{8}$$

Podemos completar o quadrado de outros polinômios do segundo grau:

```
> completarquadrado(x^2-17*x+11);
```

$$(x-\frac{17}{2})^2 - \frac{245}{4}$$

Em geral, para executar a sequência dos comandos que estão no bloco:

```
NOME := proc(VARIAVEL)
comando1;
comando2;
...
comandoN
end;
```

temos que dar o comando:

```
> NOME(expr1);
```

Os comandos do bloco serão executados em ordem, porém seus resultados não serão mostrados na tela, exceto o resultado do último comando. Toda vez que aparecer a palavra VARIAVEL dentro do bloco, ela será substituída por *expr1*. Isso explica porque quando demos o comando completarquadrado($x^2 - 17\,x + 11$), a variável *polinomio1*, que aparece dentro do programa, foi substituída por $x^2 - 17\,x + 11$.

3 Diagonalizando uma Matriz

Vamos ver o segundo exemplo. Dada a matriz A, queremos achar a matriz S que diagonaliza A, ou seja, achar S tal que $S^{(-1)}\,A\,S$ seja uma matriz diagonal. Os elementos da diagonal serão os autovalores de A. Para trabalhar com matrizes no Maple, temos que carregar o pacote *linalg*:

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

```
Warning, new definition for norm


Warning, new definition for trace
```

Vamos supor que A é uma matriz 3x3 dada por:

```
> A := matrix( 3, 3, [$1..9] );
```

$$A := \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

As colunas da matriz S são os autovetores de A. Vamos então determinar os autovetores através do comando *eigenvects*:

```
> autovts := eigenvects(A);
```

$$autovts := [0,\ 1,\ \{[1,\ -2,\ 1]\}],\ [\frac{15}{2} + \frac{3}{2}\sqrt{33},\ 1,\ \{\left[\frac{19}{4} - \frac{3}{4}\sqrt{33},\ 1,\ -\frac{11}{4} + \frac{3}{4}\sqrt{33}\right]\}],$$
$$[\frac{15}{2} - \frac{3}{2}\sqrt{33},\ 1,\ \{\left[\frac{19}{4} + \frac{3}{4}\sqrt{33},\ 1,\ -\frac{11}{4} - \frac{3}{4}\sqrt{33}\right]\}]$$

O resultado é uma sequência de listas, onde cada lista tem 3 argumentos: o autovalor, a multiplicidade e o conjunto dos autovetores. Queremos selecionar apenas os autovetores. Para selecionar o primeiro autovetor, usamos o seguinte comando:

```
> autovts[1][3][1];
```

$$[1,\ -2,\ 1]$$

O comando *autovts[1]* fornece o primeiro elemento da sequência *autovts*. O comando *autovts[1][3]* fornece o terceiro elemento da primeira lista que é o autovetor dentro de um conjunto e finalmente *autovts[1][3][1]* fornece o autovetor.

Agora, vamos construir a função F, tal que, dada uma lista com 3 elementos, F seleciona o conteúdo do terceiro argumento:

```
> F := L -> L[3][1];
```

$$F := L \rightarrow L_{31}$$

Por exemplo:

```
> teste := [2, 1, {[a,b,c]}];
```

$$teste := [2,\ 1,\ \{[a,\ b,\ c]\}]$$

```
> F(teste);
```

$$[a,\ b,\ c]$$

Temos que aplicar a função F a cada elemento da sequência autovts. Isso é feito pelo comando *map*, da seguinte forma:

```
> autovts1 := map(F,[autovts]);
```

$$autovts1 := [[1,\ -2,\ 1],\ \left[\frac{19}{4} - \frac{3}{4}\sqrt{33},\ 1,\ -\frac{11}{4} + \frac{3}{4}\sqrt{33}\right],\ \left[\frac{19}{4} + \frac{3}{4}\sqrt{33},\ 1,\ -\frac{11}{4} - \frac{3}{4}\sqrt{33}\right]]$$

Observe que colocamos a sequência *autovts* dentro de uma lista. Isso é necessário porque o primeiro argumento do comando *map* deve ser uma função e o segundo deve ser uma lista ou um conjunto. Já selecionamos os autovetores. Queremos agora guardar cada um numa variável, da seguinte forma: *v1* será o primeiro autovetor, *v2* o segundo e *v3* o terceiro. O operador ponto cria essas variáveis. Vamos fazer a seguinte iteração:

```
> for i to 3 do v.i := autovts1[i] od;
```

$$v1 := [1,\ -2,\ 1]$$

$$v2 := \left[\frac{19}{4} - \frac{3}{4}\sqrt{33},\ 1,\ -\frac{11}{4} + \frac{3}{4}\sqrt{33}\right]$$

$$v3 := \left[\frac{19}{4} + \frac{3}{4}\sqrt{33},\ 1,\ -\frac{11}{4} - \frac{3}{4}\sqrt{33}\right]$$

Vamos agora juntar esse autovetores numa matriz. Cada autovetor deve ser uma coluna da matriz S, ou seja queremos juntar *v1, v2* e *v3* horizontalmente. Nesse caso vamos usar o comando *augment*:

```
> S := augment(v1,v2,v3);
```

$$S := \begin{bmatrix} 1 & \frac{19}{4} - \frac{3}{4}\sqrt{33} & \frac{19}{4} + \frac{3}{4}\sqrt{33} \\ -2 & 1 & 1 \\ 1 & -\frac{11}{4} + \frac{3}{4}\sqrt{33} & -\frac{11}{4} - \frac{3}{4}\sqrt{33} \end{bmatrix}$$

A matriz S é a solução do nosso problema. Vamos confirmar se o cálculo está correto, ou seja, vamos verificar se $S^{(-1)}\,A\,S$ é uma matriz diagonal:

```
> Sinv := inverse(S);
```

$$Sinv := \begin{bmatrix} \frac{1}{6} & \frac{-1}{3} & \frac{1}{6} \\ \frac{1}{198}(3+\sqrt{33})\sqrt{33} & \frac{1}{198}(5+\sqrt{33})\sqrt{33} & \frac{1}{198}(7+\sqrt{33})\sqrt{33} \\ \frac{1}{198}(-3+\sqrt{33})\sqrt{33} & \frac{1}{198}(-5+\sqrt{33})\sqrt{33} & \frac{1}{198}(-7+\sqrt{33})\sqrt{33} \end{bmatrix}$$

```
> map(simplify,evalm(Sinv &* A &* S));
```

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & \frac{15}{2}+\frac{3}{2}\sqrt{33} & 0 \\ 0 & 0 & \frac{15}{2}-\frac{3}{2}\sqrt{33} \end{bmatrix}$$

O resultado foi confirmado. Podemos verificar também que os elementos da diagonal de são os autovalores de A.

Queremos agora construir um programa que calcule a matriz S, que diagonaliza uma matriz 3x3 qualquer. Da mesma forma como fizemos anteriormente, vamos juntar todos os comandos usados no exemplo acima para diagonalizar a matriz A e construir o bloco *proc(A) ... end*, da seguinte forma:

```
> MatrizS := proc(A)
> local autovts, F,  autovts1, i, S;
> F := L -> L[3][1];
> autovts1 := map(F,[autovts]);
> for i to 3 do v.i := autovts1[i] od;
> S := augment(v1,v2,v3);
> end;
```

$$\begin{array}{l} MatrizS := proc(A) \\ \quad local autovts,\, F,\, autovts1,\, i,\, S; \\ \quad\quad autovts := eigenvects(A)\,; \\ \quad\quad F := L \rightarrow L_{31}\,; \\ \quad\quad autovts1 := map(F,\, [autovts])\,; \\ \quad\quad for\, i\, to\, 3\, do\, v.i := autovts1_i\, od\,; \\ \quad\quad S := augment(v1,\, v2,\, v3) \\ \quad end \end{array}$$

Já declaramos locais todas as variáveis usadas dentro do bloco, de forma a evitar os avisos de declaração implícita das variáveis locais. Observe que uma variável local pode ser uma função. Vamos testar o programa na matriz A do exemplo:

```
> MatrizS( A );
```

$$\begin{bmatrix} \frac{19}{4}-\frac{3}{4}\sqrt{33} & \frac{19}{4}+\frac{3}{4}\sqrt{33} & 1 \\ 1 & 1 & -2 \\ -\frac{11}{4}+\frac{3}{4}\sqrt{33} & -\frac{11}{4}-\frac{3}{4}\sqrt{33} & 1 \end{bmatrix}$$

Observe que obtivemos uma matriz diferente da matriz encontrada anteriormente. No entanto, essa matriz também é solução do problema, como podemos verificar através do seguinte teste:

```
> map(simplify,evalm(1/" &* A &* "));
```

$$\begin{bmatrix} \frac{15}{2}+\frac{3}{2}\sqrt{33} & 0 & 0 \\ 0 & \frac{15}{2}-\frac{3}{2}\sqrt{33} & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Comentamos anteriormente que os resultados dos comandos intermediários dentro do bloco não são mostrados na tela. Somente o resultado do último comando é mostrado. A fim de acompanhar o algoritmo, podemos aumentar o valor da variável *printlevel*, o que faz com que os resultados intermediários passem a ser mostrados na tela. O valor inicial de *printlevel* é 1, como podemos verificar:

```
> printlevel;
```

$$1$$

Vamos agora modificar o seu valor:

```
> printlevel:=8;
```

$$printlevel := 8$$

Veja o efeito:

```
> MatrizS(A);

enter MatrizS, args = A
```

$$autovts := [\frac{15}{2}+\frac{3}{2}\sqrt{33},\ 1,\ \{\left[-\frac{1}{2}+\frac{3}{22}\sqrt{33},\ \frac{1}{4}+\frac{3}{44}\sqrt{33},\ 1\right]\}],$$
$$[\frac{15}{2}-\frac{3}{2}\sqrt{33},\ 1,\ \{\left[-\frac{1}{2}-\frac{3}{22}\sqrt{33},\ \frac{1}{4}-\frac{3}{44}\sqrt{33},\ 1\right]\}],\ [0,\ 1,\ \{[1,\ -2,\ 1]\}]$$

$$F := L \rightarrow L_{31}$$

$$autovts1 := [\left[-\frac{1}{2}+\frac{3}{22}\sqrt{33},\ \frac{1}{4}+\frac{3}{44}\sqrt{33},\ 1\right],\ \left[-\frac{1}{2}-\frac{3}{22}\sqrt{33},\ \frac{1}{4}-\frac{3}{44}\sqrt{33},\ 1\right],\ [1,\ -2,\ 1]]$$

$$v1 := \left[-\frac{1}{2}+\frac{3}{22}\sqrt{33},\ \frac{1}{4}+\frac{3}{44}\sqrt{33},\ 1\right]$$

$$v2 := \left[-\frac{1}{2}-\frac{3}{22}\sqrt{33},\ \frac{1}{4}-\frac{3}{44}\sqrt{33},\ 1\right]$$

$$v3 := [1,\ -2,\ 1]$$

$$S := \begin{bmatrix} -\frac{1}{2}+\frac{3}{22}\sqrt{33} & -\frac{1}{2}-\frac{3}{22}\sqrt{33} & 1 \\ \frac{1}{4}+\frac{3}{44}\sqrt{33} & \frac{1}{4}-\frac{3}{44}\sqrt{33} & -2 \\ 1 & 1 & 1 \end{bmatrix}$$

```
exit MatrizS (now at top level) = array(1\ ..\ 3, 1 ..
3,[(1, 2)=-1/2-3/22*33(1/2),(1,
1)=-1/2+3/22*33(1/2),(2, 2)=1/4-3/44*33(1/2),(1,
3)=1,(2, 1)=1/4+3/44*33(1/2),(2, 3)=-2,(3, 1)=1,(3,
2)=1,(3, 3)=1])\}
```

$$\begin{bmatrix} -\frac{1}{2}+\frac{3}{22}\sqrt{33} & -\frac{1}{2}-\frac{3}{22}\sqrt{33} & 1 \\ \frac{1}{4}+\frac{3}{44}\sqrt{33} & \frac{1}{4}-\frac{3}{44}\sqrt{33} & -2 \\ 1 & 1 & 1 \end{bmatrix}$$

```
> printlevel := 1;
```

$$printlevel := 1$$

Para valores entre 5 e 10, os resultados das atribuições são mostrados. Para valores acima de 10, mais e mais informações serão dadas. Em geral, só é necessário valores mais alto quando se tem sub-rotinas dentro do programa, nesse caso podemos querer ver os resultados intermediários das sub-rotinas também. Para valores acima de 30, os resultados intermediários dos comandos do próprio Maple ser o mostrados.

Alguns problemas podem surgir na elaboração de um programa usando o método descrito nesse capítulo. O problema mais provável é com relação a declaração de variáveis locais. No modo interativo, as variáveis são ditas globais. Quando o conjunto de comandos é envolvido por *proc( ) ... end*, uma série de variáveis podem ser declaradas locais implicitamente ou explicitamente. Neste caso, elas terão um comportamento diferente do modo interativo, o que pode alterar o funcionamento do conjunto de comandos. Uma possível forma de solucionar um problema deste tipo, é declarar todas as variáveis como globais. Isto é feito com o comando *global*, que tem a mesma sintaxe do comando *local*. Dessa forma, o comportamento das variáveis dentro e fora do programa serão iguais. Esta solução gera outros problemas, pois as variáveis usadas no programa podem entrar em conflito com as variáveis de outros programas.

Pode ocorrer outro problema cuja detecção é mais difícil. Quando um programa chama outro programa ou quando ele usa funções, as variáveis de mesmo nome usadas nessas estruturas podem parecer para o usuário que são as mesmas variáveis, mas não serem na realidade. Declarar essas variáveis como globais soluciona apenas uma parte do problema.

Capítulo 9

Aplicação: Oscilação de Membranas

1 Equação de Movimento da Membrana

Como aplicação das ferramentas tratadas anteriormente, vamos fazer uma análise do movimento oscilatório dos modos normais de membranas. Considere uma membrana horizontal uniformemente distentida em todas as direções por uma tração T. A membrana pode ter qualquer forma que será especificada pelas condições de contorno. Qualquer que seja a forma, vamos analisar o caso em que toda a borda da membrana está fixa. As oscilações são sempre na direção vertical. Sob certas restrições, como por exemplo pequenas oscilações, a equação de onda que descreve o deslocamento vertical $u(x, y)$ na ausência de forças externas é dada por:

$$c^2 \nabla^2 u = \frac{\partial^2}{\partial t^2} u$$

onde $c^2 = \frac{T}{\mu}$ e μ é a densidade de massa. Esta equação diferencial parcial pode ser resolvida pelo método de separação de variáveis, de maneira que a solução geral fica expressa na forma de uma série de Fourier generalizada. No caso da membrana retangular, a solução é uma série de Fourier tripla usual e no caso da membrana circular axialmente simétrica, a solução é uma série de Fourier-Bessel na parte espacial e Fourier usual na parte temporal. Vamos analisar o movimento oscilatório dos modos normais destes destes dois tipos de membranas (ver referência [1]).

2 Membrana Retangular

2.1 Resolução da equação de onda

A membrana retangular de comprimento a e largura b presa nas bordas é caracterizada pelas seguintes condições de contorno: $u(0, y, t) = 0$ e $u(a, y, t) = 0$ para $y \leq b$ e $u(x, 0, t) = 0$ e $u(x, b, t) = 0$ para $x \leq a$. Vamos atribuir a equação diferencial deste problema à variável *wave_eq*:

```
> wave_eq := Diff(u,x,x) + Diff(u,y,y) = Diff(u,t,t)/c^2;
```

$$wave\_eq := (\frac{\partial^2}{\partial x^2} u) + (\frac{\partial^2}{\partial y^2} u) = \frac{\frac{\partial^2}{\partial t^2} u}{c^2}$$

Vamos resolver essa equação usando o método de separação de variáveis. Vamos supor que $u(x,\, y,\, t)$ é uma função separável nas variáveis x, y, e t. Assim, $u(x,\, y,\, t) = X(x)\, Y(y)\, T(t)$:

```
> eq1 := expand(value(subs(u=X(x)*Y(y)*T(t),wave_eq)/(X(x)*Y(y)*T(t))));
```

$$eq1 := \frac{\frac{\partial^2}{\partial x^2} X(x)}{X(x)} + \frac{\frac{\partial^2}{\partial y^2} Y(y)}{Y(y)} = \frac{\frac{\partial^2}{\partial t^2} T(t)}{T(t)\, c^2}$$

Cada termo da expressão acima deve ser constante, pois cada um é função de uma variável independente. Vamos selecionar o termo dependente de x, e igualar a constante $\lambda 1$:

```
> eqx := expand(X(x)*(select(has, lhs(eq1), x) = lambda1));
```

$$eqx := \frac{\partial^2}{\partial x^2} X(x) = X(x)\, \lambda 1$$

A equação diferencial acima pode ter soluções oscilatórias, lineares ou exponenciais dependendo se a constante $\lambda 1$ for negativa, zero ou positiva. Para que as condições de contorno sejam satisfeitas, isto é, $X(0) = 0$ e $X(a) = 0$, temos que impor que a constante $\lambda 1$ seja negativa e que ela assuma valores discretos dados por:

```
> lambda1 := -m^2*Pi^2/a^2;
```

$$\lambda 1 := -\frac{m^2\, \pi^2}{a^2}$$

onde m é um inteiro positivo. A solução geral da equação da parte x é:

```
> dsolve({eqx,X(0)=0}, X(x));
```

$$X(x) = \_C2\, sin(\frac{\pi\, m\, x}{a})$$

Vamos eliminar a constante arbitrária por enquanto:

```
> assign(simplify(subs(_C1=1, _C2=1, ")));
> X(x);
```

$$sin(\frac{\pi\, m\, x}{a})$$

Vamos repetir o mesmo procedimento para o termo dependente de y:

```
> eqy := expand(Y(y)*(select(has, lhs(eq1), y) = lambda2)):
> lambda2 := -n^2*Pi^2/b^2:
> dsolve({eqy,Y(0)=0}, Y(y)):
> assign(simplify(subs(_C1=1, _C2=1, ")));
> Y(y);
```

$$sin(\frac{\pi\, n\, y}{b})$$

A equação diferencial para a função dependente da variável t é obtida da seguinte forma:

```
> eqt := normal(c^2*T(t)*subs(eqx, eqy, eq1));
```

$$eqt := -\frac{c^2\, T(t)\, \pi^2\, (m^2\, b^2 + n^2\, a^2)}{a^2\, b^2} = \frac{\partial^2}{\partial t^2} T(t)$$

cuja solução geral é:

```
> dsolve(eqt, T(t));
```

$$T(t) = \_C1\, e^{(\frac{\sqrt{-m^2\,b^2-n^2\,a^2}\,\pi\,c\,t}{b\,a})} + \_C2\, e^{(-\frac{\sqrt{-m^2\,b^2-n^2\,a^2}\,\pi\,c\,t}{b\,a})}$$

Vamos colocar esta solução numa forma mais conveniente:

```
> simplify(evalc("),{_C1+_C2=A(m,n), _C1-_C2=B(m,n)/I});
> assign(");
```

$$T(t) = cos(\frac{\sqrt{m^2\,b^2+n^2\,a^2}\,\pi\,c\,t}{b\,a})\,A(m,\,n) + sin(\frac{\sqrt{m^2\,b^2+n^2\,a^2}\,\pi\,c\,t}{b\,a})\,B(m,\,n)$$

onde A e B são constantes que podem depender dos autovalores m e n. Vamos definir a frequência de cada modo normal:

```
> unprotect(omega):
> omega := (m,n) -> (m^2*b^2+n^2*a^2)^(1/2)*Pi*c/b/a;
> protect(omega):
```

$$\omega := (m,\,n) \rightarrow \frac{\sqrt{m^2\,b^2+n^2\,a^2}\,\pi\,c}{b\,a}$$

e definir os modos normais como uma função de m e n. Existem duas possibilidades que são:

```
> normal_mode := unapply(simplify(subs(A=1, B=0, X(x)*Y(y)*T(t))),m,n);
```

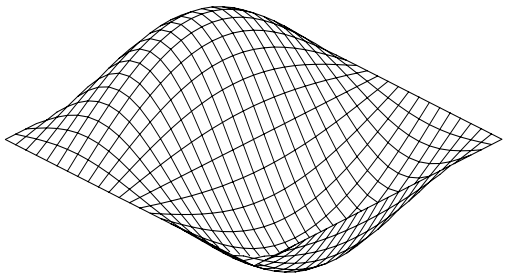
$$normal\_mode := (m,\,n) \rightarrow sin(\frac{\pi\,m\,x}{a})\,sin(\frac{\pi\,n\,y}{b})\,cos(\frac{\sqrt{m^2\,b^2+n^2\,a^2}\,\pi\,c\,t}{b\,a})$$

```
> normal_mode1 := unapply(simplify(subs(A=0, B=1, X(x)*Y(y)*T(t))),m,n);
```

$$normal\_mode1 := (m,\,n) \rightarrow sin(\frac{\pi\,m\,x}{a})\,sin(\frac{\pi\,n\,y}{b})\,sin(\frac{\sqrt{m^2\,b^2+n^2\,a^2}\,\pi\,c\,t}{b\,a})$$

O gráfico da posição inicial do modo <1,2> de uma membrana quadrada de comprimento unitário num certo sistema de unidades é:

```
> a:=1:  b:=1:  c:=1:
> plot3d(subs(t=0,normal_mode(1,2)), x=0..1, y=0..b);
```



2.2 Animação dos Modos Normais

Os modos normais podem ser animados como a função *animate3d*. Vamos construir o procedimento *animate_mode*, que anima o modo especificado pelos seus dois primeiros argumentos. Por exemplo, *animate_mode*(1,2) faz a animação do modo m=1 e n=2. Sendo um comando de animação, ele deve aceitar as mesmas opções do comando *animate3d*. Se nada for dito sobre as opção *style*, este procedimento assume que o estilo é *patch*, que é indicado para este tipo de aplicação. As outras opções seguem o *default* do comando *animate3d*.

```
> animate_mode := proc(m::posint,n::posint)
> local FRAM,Options;
> global a,b,x,y,t,animate3d;
> if nargs=2
> then Options:='frames'=8,'style=patch'; FRAM:=8;
> else Options:=args[3..nargs];
> if has({Options},'frames')
> then FRAM:=subs({Options},'frames');
> else Options:=Options,'frames'=8; FRAM:=8
> fi;
```

```
> if not has({Options},'style')
> then Options:=Options,'style=patch'
> fi
> fi;
> plots[animate3d](normal_mode(m,n), x=0..a, y=0..b,
> t=0..2*Pi*FRAM/omega(m,n)/(FRAM+1), Options)
> end;
```

$$
\begin{array}{l}
animate\_mode := proc(m::posint,\ n::posint) \\
\quad local FRAM,\ Options; \\
\quad global a,\ b,\ x,\ y,\ t,\ animate3d; \\
\quad\quad if\ nargs = 2\ then\ Options := 'frames' = 8,\ 'style = patch'\ ;\ FRAM := 8 \\
\quad\quad else \\
\quad\quad\quad Options := args_{3..nargs}\ ; \\
\quad\quad\quad if\ has(\{Options\},\ 'frames')\ then\ FRAM := subs(\{Options\},\ 'frames') \\
\quad\quad\quad else\ Options := Options,\ 'frames' = 8\ ;\ FRAM := 8 \\
\quad\quad\quad fi; \\
\quad\quad\quad if\ \ not\ has(\{Options\},\ 'style')\ then\ Options := Options,\ 'style = patch'\ fi \\
\quad\quad fi; \\
\quad\quad plots_{animate3d}(normal\_mode(m,\ n),\ x = 0..a,\ y = 0..b, \\
\quad\quad\quad t = 0..2\ \pi\ FRAM/(\omega(m,\ n)\ (FRAM + 1)),\ Options) \\
\quad end
\end{array}
$$

Para fazer a animação de um determinado modo, temos que dar valores numéricos para os parâmetros a, b e c. Vamos supor que o comprimento é *a=1*, a largura é *b=2* e a constante *c=1* em um certo sistema de unidades:

```
> a := 1; b := 2; c := 1;
```

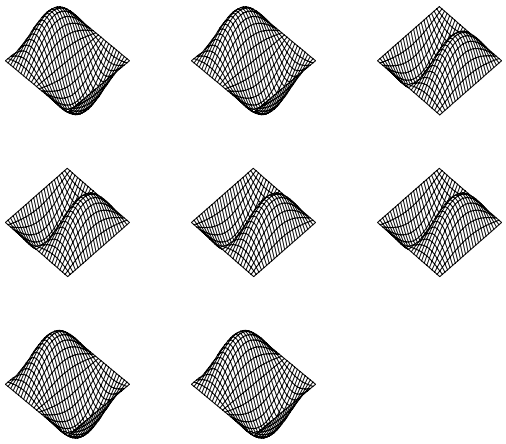
$$a := 1$$

$$b := 2$$

$$c := 1$$

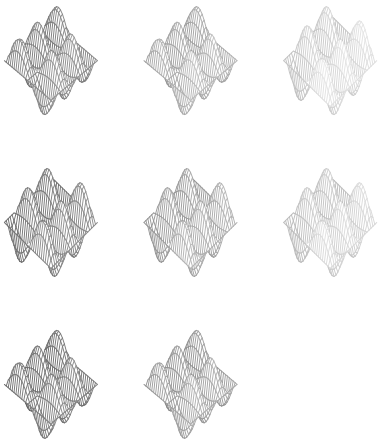
Vamos fazer a animação do modo <1,2> nas opções *default*:

```
> animate_mode(1,2);
```



As linhas nodais são curvas cujos pontos se mantêm em repouso durante todo instante. Elas podem ser melhor visualizadas através da opção *style=patchcontour*. Por exemplo, vejamos a animação do modo <2,5>:

```
> animate_mode(5, 2,style=patchcontour, scaling=constrained);
```



Podemos ver que há 4 retas nodais paralelas ao eixo y e 1 reta nodal paralela ao eixo x. Esta característica pode ser generalizada para qualquer modo de vibração. O modo <*m,n*> tem *m-1* retas nodais paralelas ao eixo y e *n-1* retas nodais paralelas ao eixo x. Em geral, as linhas nodais nem sempre são retas (ver referência [1]).

3 Membrana Circular

O objetivo desta seção é fazer a animação dos modos normais axialmente simétricos de uma membrana circular de raio a. A equação de onda pode ser resolvida no sistema de coordenadas cilíndricas, que é o mais adequado para descrever as condições de contorno. A equação de onda é:

```
> wave_eq := linalg[laplacian](u(r,t),[r,theta,z],coords=cylindrical) =
1/c^2*diff(u(r,t),t,t);
```

$$wave\_eq := \frac{(\frac{\partial}{\partial r}\,u(r,\,t)) + r\,(\frac{\partial^2}{\partial r^2}\,u(r,\,t))}{r} = \frac{\frac{\partial^2}{\partial t^2}\,u(r,\,t)}{c^2}$$

De maneira análoga a forma usada para a membrana retangular, vamos usar o método de separação de variáveis:

```
> sol := expand(subs(u(r,t)=R(r)*T(t),wave_eq)/(R(r)*T(t)));
```

$$sol := \frac{\frac{\partial}{\partial r}\,R(r)}{R(r)\,r} + \frac{\frac{\partial^2}{\partial r^2}\,R(r)}{R(r)} = \frac{\frac{\partial^2}{\partial t^2}\,T(t)}{T(t)\,c^2}$$

```
> dsolve((lhs(sol)=-k(n)^2)*R(r),R(r));
```

$$R(r) = \_C1\, BesselY(0,\, k(n)\, r) + \_C2\, BesselJ(0,\, k(n)\, r)$$

Vamos eliminar o termo com a função Bessel Y uma vez que ela não é analítica em $r = 0$:

```
> eval(subs(BesselY=0,_C1=1,_C2=1,"));
```

$$R(r) = BesselJ(0,\, k(n)\, r)$$

```
> assign(");
> dsolve((rhs(sol)=-k(n)^2)*T(t),T(t));
```

$$T(t) = \_C1\, cos(c\, k(n)\, t) + \_C2\, sin(c\, k(n)\, t)$$

```
> assign(");
```

Os modos normais são descritos pela funções:

```
> normal_mode := unapply(R(r)*eval(subs(sin=0,_C1=1,_C2=1,T(t))),n);
```

$$normal\_mode := n \rightarrow BesselJ(0,\, k(n)\, r)\, cos(c\, k(n)\, t)$$

```
> normal_mode1 := unapply(R(r)*eval(subs(cos=0,_C1=1,_C2=1,T(t))),n);
```

$$normal\_mode1 := n \rightarrow BesselJ(0,\, k(n)\, r)\, sin(c\, k(n)\, t)$$

A função $R(r)$ deve se anular em $r = a$. Assim, k deve ser da forma:

```
> k := n -> alpha(n)/a;
```

$$k := n \rightarrow \frac{\alpha(n)}{a}$$

onde $\alpha(n)$ é o n-ésimo zero da função de Bessel J de ordem zero. Um procedimento para achar estes zeros é:

```
> alpha := proc(n)
> options remember;
> if type(n,name) then 'alpha(n)';
> elif n=0 then 0
> else fsolve(BesselJ(0,x),x,(n-1)*Pi..n*Pi)
> fi
> end;
```

$$\begin{array}{l} \alpha := proc(n) \\ \quad option remember; \\ \quad\quad if\, type(n,\, name)\, then\, '\alpha(n)' \\ \quad\quad elif\, n = 0\, then\, 0 \\ \quad\quad else\, fsolve(BesselJ(0,\, x),\, x,\, (n-1)\, \pi..n\, \pi) \\ \quad\quad fi \\ \quad end \end{array}$$

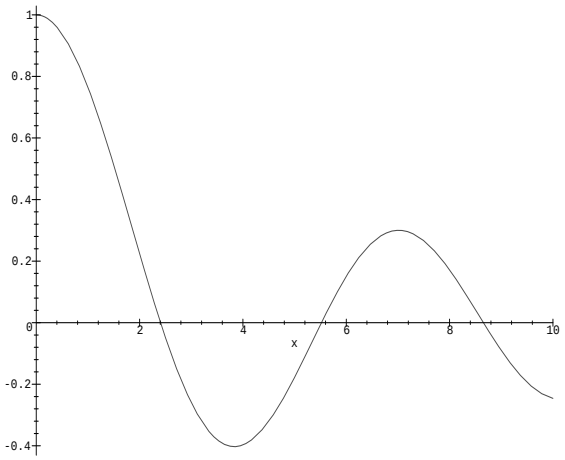
Por exemplo, os três primeiros zeros são:

```
> seq(alpha(i),i=1..3);
```

$$2.404825558,\ 5.520078110,\ 8.653727913$$

Através do grafico podemos confirmar estes resultados:

```
> plot(BesselJ(0,x),x=0..10);
```



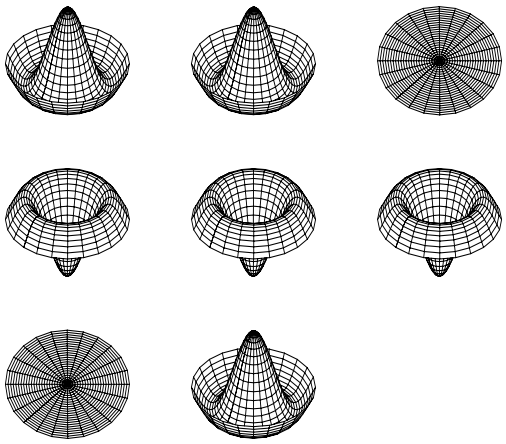
O procedimento a seguir gera a animação dos modos axialmente simétricos de ordem n:

```
> animate_mode := proc(n::nonnegint)
> local FRAM,STY,i,Options,G,z;
> global a,c,r,t;
> c:=1; if nargs=1
> then Options:='frames'=8,'style=patch'; FRAM:=8;
> else Options:=args[2..nargs];
> if has({Options},'frames')
> then FRAM:=subs({Options},'frames');
> else Options:=Options,'frames'=8; FRAM:=8
> fi;
> if not has({Options},'style')
```

```
> then Options:=Options,'style=patch'
> fi
> fi;
> for i from 0 to FRAM-1 do
> z:= eval(subs(t=2*Pi*i/(FRAM*k(n)*c),normal_mode(n)));
> G[i] := plots[cylinderplot]([r,theta,z],r=0..a,theta=0..2*Pi, style=patch,Options)
> od:
> plots[display]([seq(G[i],i=0..FRAM-1)],insequ ence=true);
> end;
```

$$
\begin{array}{l}
animate\_mode := proc(n::nonnegint) \\
\quad local FRAM,\ STY,\ i,\ Options,\ G,\ z; \\
\quad global a,\ c,\ r,\ t; \\
\quad\quad c := 1; \\
\quad\quad if\ nargs = 1\ then\ Options := 'frames' = 8,\ 'style = patch';\ FRAM := 8 \\
\quad\quad else \\
\quad\quad\quad Options := args_{2..nargs}; \\
\quad\quad\quad if\ has(\{Options\},\ 'frames')\ then\ FRAM := subs(\{Options\},\ 'frames') \\
\quad\quad\quad else\ Options := Options,\ 'frames' = 8;\ FRAM := 8 \\
\quad\quad\quad fi; \\
\quad\quad\quad if\ not\ has(\{Options\},\ 'style')\ then\ Options := Options,\ 'style = patch'\ fi \\
\quad\quad fi; \\
\quad\quad for\ i\ from\ 0\ to\ FRAM - 1\ do \\
\quad\quad\quad z := eval(subs(t = 2\ \pi\ i/(FRAM\ k(n)\ c),\ normal\_mode(n))); \\
\quad\quad\quad G_i := plots_{cylinderplot}([r,\ \theta,\ z],\ r = 0..a,\ \theta = 0..2\ \pi,\ style = patch,\ Options) \\
\quad\quad od; \\
\quad\quad plots_{display}([seq(G_i,\ i = 0..FRAM - 1)],\ insequence = true) \\
\quad end
\end{array}
$$

```
> a:=1:  c:=1:
> animate_mode(2);
```



Exercícios[1]

1 - Ache a expansão em frações contínuas de $\frac{1+\sqrt{5}}{2}$. [Sug.: Veja o pacote *numtheory*].

2 - Mostre que

$$tanh(\frac{1}{2}\,z) = \frac{sinh(a) + I\,sin(b)}{cosh(a) + cos(b)}$$

onde $z = a + I\,b$.

3 - Considere a seguinte expressão: $\frac{x^4+x^3-4\,x^2-4\,x}{x^4+x^3-x^2-x}$. Usando os comandos de simplificação, transforme-a em:

a) $\frac{(x-2)\,(x+2)\,(x+1)}{x^3+x^2-x-1}$
b) $\frac{x^4+x^3-4\,x^2-4\,x}{x\,(x-1)\,(x+1)^2}$
c) $\frac{(x-2)\,(x+2)}{(x-1)\,(x+1)}$
d) $\frac{x^2}{(x-1)\,(x+1)} + \frac{4}{(x-1)\,(x+1)}$

4 - Faça as seguintes transfomações no Maple:

a) $(x+y)^2 + \frac{1}{x+y}$ <——> $\frac{(x+y)^3+1}{x+y}$
b) $x^2 + 2\,x + 1 + \frac{1}{x^2+2\,x+1}$ <——> $\frac{(x+1)^4+1}{(x+1)^2}$

5 - Usando os comandos de simplificação, mostre que $ln(tan(\frac{x}{2} + \frac{\pi}{4})) = arcsinh(tan(x))$.

6 - a) Construa uma lista com 100 números randômicos maiores que zero e menores que 100. [Sug.: Use o comando *seq* e *rand*]

b) Elimine as repetições.

c) Elimine as repetições mantendo a ordem da lista original.

7 - Suponha que $b^2 = 2\,a\,(1-a)$ e que

$$A = \begin{bmatrix} a & a-1 & b \\ a-1 & a & b \\ -b & -b & 2\,a-1 \end{bmatrix}$$

[1] A maioria destes exercícios foi extraída das referências [3] e [4].

a) Mostre no Maple que A é uma matriz ortogonal com determinante 1.
b) Uma vez que A descreve uma rotação, determine o eixo de rotação.

8 - Faça o gráfico da função $max(x, x^3)$ junto com o gráfico da sua derivada no intervalo $[-2..2]$. Explique o resultado.

9 - a) Resolva a equação recursiva de Fibonacci $f(n) = f(n-1) + f(n-2)$, $f(0) = 0$, $f(1) = 1$. [Sug.: use o comando *rsolve*]

b) A partir da solução defina uma função de nome *Fibonacci* que retorna os números de Fibonacci na forma já simplificada. Por exemplo:

```
> Fibonacci(20);
```

$$6765$$

10 - Resolva a equação de condução de calor $\frac{\partial}{\partial t} u = \frac{\partial^2}{\partial x^2} u$ usando o método da transformada de Fourier na variável x. A condição inicial é $u(x, 0) = u_0\, \delta(x)$. [Sug.: use o comando *fourier* e depois *invfourier*.]

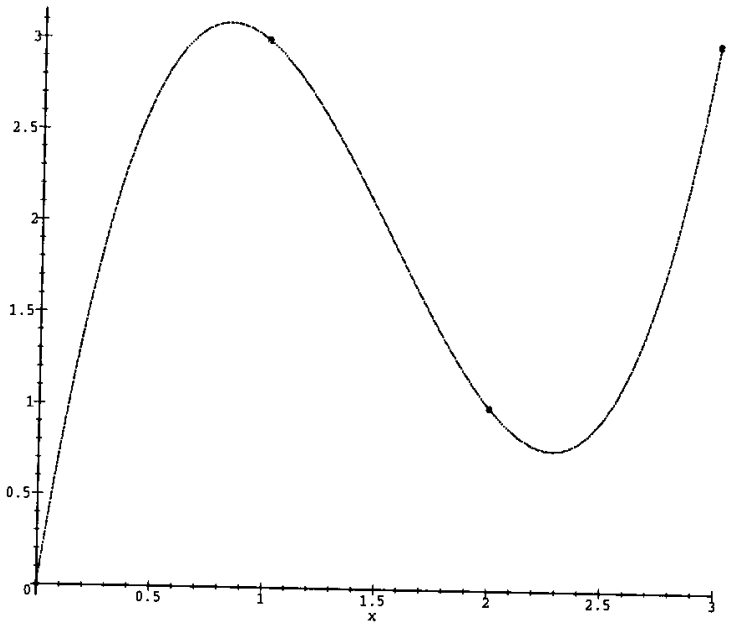
11 - a) Faça um procedimento que encontra o polinômio de grau 3 na variável x que interpola os pontos $[[x_1, y_1], [x_2, y_2], [x_3, y_3], [x_4, y_4]]$. Por exemplo:

```
> interpola([ [0,0], [1,3], [2,1], [3,3] ]);
```

$$\frac{3}{2} x^3 - 7 x^2 + \frac{17}{2} x$$

b) Faça o gráfico dos pontos [[0,0], [1,3], [2,1], [3,3]] com a opção *symbol=circle* e guarde em uma variável. Faça depois o gráfico do polinômio encontrado em a) e guarde em outra variável. O comando *plots*[*display*] permite mostrar os dois gráficos juntos. Faça um programa tal que dada a lista de pontos, ele gera esses dois gráficos juntos. Por exemplo:

```
> plotinterp([ [0,0], [1,3], [2,1], [3,3] ]);
```



12 - Desenvolva um procedimento que faz a animação dos modos normais de oscilação de uma corda presa nas extremidades.

Referências:

[**1**] D. Frenkel, L. Golebiowski e R. Portugal, *Estudo da Oscilação de Membranas usando Maple*, monografia a ser publicada, CBPF, 1996.

[**2**] K. M. Heal, M. L. Hansen e K. M. Rickard, *Maple V: Learning Guide*, Springer-Verlag, 1996.

[**3**] André Heck, *Introduction to Maple*, Springer-Verlag, 1996.

[**4**] Renato Portugal, *Introdução à Programação em Maple*[2], Série: Notas de Aula, vol. 1, CBPF, 1996.

[2]Este livro é distribuído pelo CBPF. Para encomendá-lo ou para obter informações sobre ele, deve-se enviar um *e-mail* para MYRIAM@NOVELL.CAT.CBPF.BR ou escrever para Coordenação de Formação Científica, CBPF, Rua Dr. Xavier Sigaud 150, Urca, Rio de Janeiro, RJ, cep 22290-180. O preço do livro é de R$ 10,00. A despesa de correio está incluida.

As *worksheets* dos exemplos do livro estão disponíveis por *ftp* anônimo no seguinte endereço ANONYMOUS@LCA1.DRP.CBPF.BR no diretório *pub/livro*.