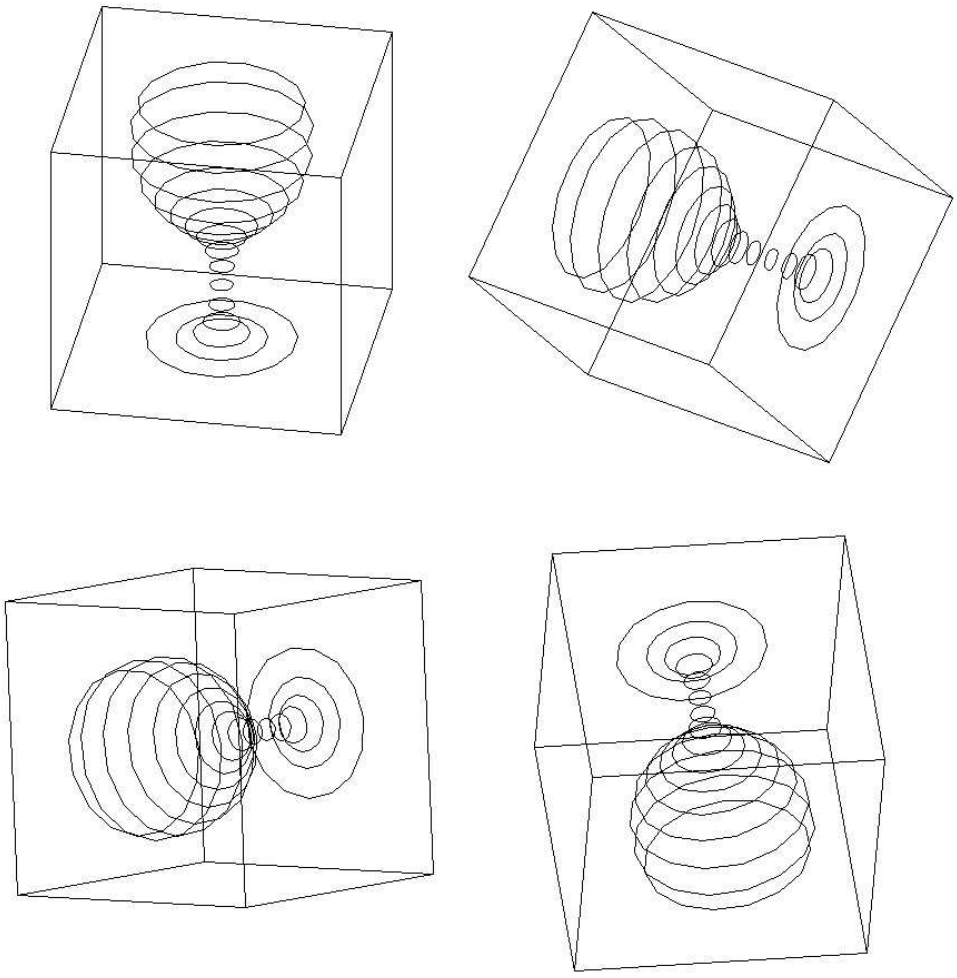
UNIVERSIDADE FEDERAL DA PARAÍBA
CENTRO DE CIÊNCIAS EXATAS E DA NATUREZA
DEPARTAMENTO DE MATEMÁTICA

Animações Gráficas com o MAPLE



Lenimar Nunes de Andrade
lenimar@mat.ufpb.br
28/setembro/2006

Sumário

Prefácio

O Maple é um programa de Computação Algébrica de uso geral que vem sendo desenvolvido desde 1981 no Canadá. Pode ser utilizado como ferramenta para auxiliar o aprendizado de temas matemáticos nos mais variados níveis e das mais diversas áreas.

Este texto corresponde ao minicurso que ministramos no VI Encontro Regional de Matemática e Computacional (ERMAC). É dedicado à construção de animações gráficas relacionadas com temas matemáticos de diferentes níveis. O principal objetivo ao se construir tais animações é o de utilizá-las como eficientes ferramentas no processo de aprendizagem de tais temas.

Experiências em aulas ou em laboratórios mostram que estudantes têm demonstrado grande interesse em visualizações e animações que envolva algum tipo de conceito gemétrico. É um investimento que pode ter um grande retorno.

Há quem acredite que qualquer tópico da Matmática só foi bem abordado se tiver sido apresentado geometricamente, algebricamente e numericamente. Para apresentar determinado tema de forma geométrica, nada melhor do que uma animação para chamar a atenção.

Agradecemos à comissão organizadora do ERMAC pelo convite e pela oportunidade em minstrar este minicurso.

João Pessoa, 28 de setembro de 2006

Lenimar Nunes de Andrade

Capítulo 1

Introdução ao Maple

Neste capítulo, apresentamos brevemente o ambiente de desenvolvimento integrado do Maple e algumas de suas funções e comandos básicos.

1.1 Introdução

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

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

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

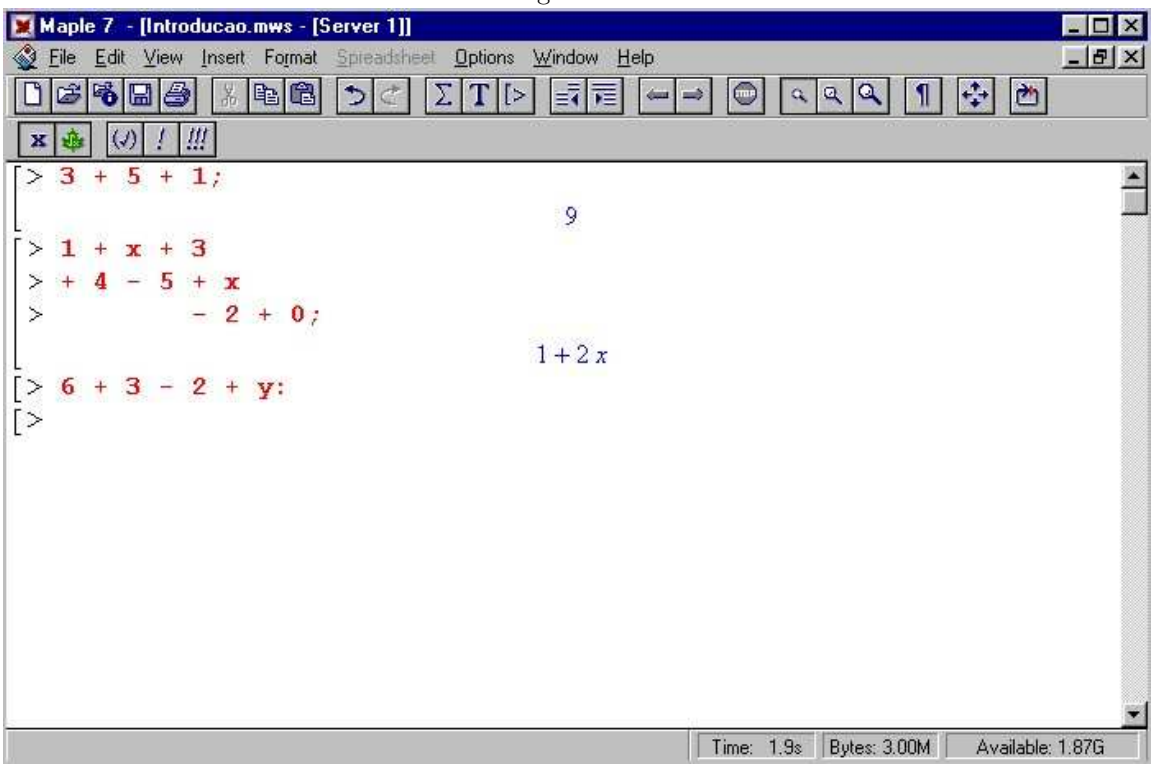
Cada comando digitado deve terminar com um ";" (ponto e vírgula) ou com ":" (dois pontos), seguido de Enter. Se o comando terminar com ponto e vírgula, o resultado da sua execução será mostrado logo em seguida. Se terminar com dois pontos, o resultado não será mostrado, podendo ficar guardado para uso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

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

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

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

Figura 1.1:



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

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

$$1+2x$$

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

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

1.2 Usando as telas de ajuda

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

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

```
> ?cos
```

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

1.3 Operações aritméticas e tipos numéricos básicos

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

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

Exemplo 1.2 *Exemplificando o uso das operações aritméticas básicas.*

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

1.4 Avaliação numérica

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

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

Exemplo 1.3 *Neste exemplo, mostramos os valores de* π *e* $\sqrt{2}$ *com vários algarismos significativos.*

```
> Digits;
```

$$10$$

```
> Pi;
```

$$\pi$$

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

$$3.141592654$$

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

$$\sqrt{2}$$

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

$$1.414213562$$

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

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

$$Digits := 30$$

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

$$3.14159265358979323846264338328$$

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

$$1.41421356237309504880168872421$$

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

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

$$1.41421356237309504880168872420969807856967187537694807317668$$

1.5 Funções matemáticas

O Maple possui muitas funções matemáticas pré-definidas. Além das funções elementares básicas, possui outras especiais, tais como as funções beta, gama, logaritmo integral, seno integral, dilogaritmo, funções de Bessel, etc.

Listamos nesta seção somente uma pequena parte das funções básicas.

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

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

1.6 Pacotes

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

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

O Maple possui vários pacotes. Eles são agrupamentos de comandos com fins específicos. Tem pacote para construção de gráficos (como o `plots`), para Álgebra Linear (como o `LinearAlgebra` e o `linalg`), para a definição de objetos geométricos (como o `geometry`), etc.

Para que um comando de um pacote possa ser utilizado, precisamos fornecer o nome do pacote que o contém. Isso pode ser feito fornecendo-se o nome do pacote seguido do nome do comando entre colchetes: *pacote*[*comando*]. Uma expressão dessa forma é chamada o *nome completo* do comando. Por exemplo, o nome completo do comando `Determinant` que faz parte do pacote `LinearAlgebra` é `LinearAlgebra[Determinant]` e o nome completo de `IsEquilateral` do pacote `geometry` é `geometry[IsEquilateral]`.

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

Exemplo 1.4 *Considere as seguintes atribuições feitas com alguns comandos do pacote* `LinearAlgebra`*:*

```
> d := LinearAlgebra[Determinant](M):
> N := LinearAlgebra[InverseMatrix](M):
```

Veja como é mais simples utilizá-los assim:

```
> with(LinearAlgebra):
> d := Determinant(M):
> N := InverseMatrix(M):
```

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

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

```
> ?geometry
```

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

1.7 Substituição

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

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

Exemplo 1.5 *Na expressão algébrica* $E = x^2 + y^2 + z^2$, *inicialmente substituímos* x *por* -2.

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

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

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

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

```
> E;
```

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

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

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

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

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

```
> E;
```

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

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

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

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

$$E := 14$$

1.8 Simplificação

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

Exemplo 1.6 *Simplificar cada uma das expressões:*

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

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

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

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

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

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

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

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

1.9 Funções

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

- com o "*operador seta*":

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

- com um comando `unapply`:

  `f := unapply(`*expressão*`, x)`

Exemplo 1.7 *Definir a função* $f(x) = x^2$ *e calcular* $f(-3)$.

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

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

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

9

Exemplo 1.8 *Usamos o* `unapply` *para definir uma função* $f(x) = x^2 + 1$ *a partir de uma expressão algébrica.*

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

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

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

10

1.10 Gráficos planos

O Maple possui muitos comandos e opções para construção de gráficos, desde os gráficos planos mais simples até os gráficos tridimensionais mais sofisticados. Possui, também, recursos para a construção de animações envolvendo esses tipos de gráficos. Neste capítulo, apresentamos uma pequena parte desses recursos.

O gráfico de uma função definida por uma expressão algébrica $y = f(x)$ na variável x pode ser construído com o comando `plot`:

`plot(`*f(x), x=a..b, y=c..d, op1, op2, ...*`)`

onde

f(x)	é uma função real
x=a..b	é a variação do x (domínio)
y=c..d	é a variação do y (opcional)
op1, op2, ...	outras opções

Na variação do x ou do y, pode aparecer `-infinity` ou `infinity` ($-\infty$ ou ∞).

Exemplo 1.9 *Neste exemplo, construímos o gráfico da parábola $y = x^2$. Definimos a variação do x como sendo o intervalo $[-2, 2]$.*

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

Exemplo 1.10 *O comando a seguir constrói o gráfico da função polinomial $y = -x^5 + 3x - 1$. Definimos o intervalo de variação do x como sendo o intervalo $[-2, 3/2]$ e, o da variação do y, como $[-4, 10]$.*

```
> plot(-x^5 + 3*x -1, x = -2..3/2, y = -4..10);
```

As opções do `plot` são fornecidas em forma de igualdades do tipo *opção = valor*. As possibilidades são:

adaptive Pode ser `true` ou `false`. Se `adaptive=true` (o padrão), o gráfico será construído com maior número de pontos onde ele tiver uma maior curvatura. Se `adaptive=false`, então será usado um espaçamento uniforme entre os pontos do gráfico.

axes Especifica o tipo dos eixos utilizados: `FRAME`, `BOXED`, `NORMAL` ou `NONE`.

axesfont Especifica o tipo de letra para marcar cada eixo. Pode ser em forma de lista [*fonte, estilo, tamanho*], onde *fonte* é um dos tipos `TIMES`, `COURIER`, `HELVETICA` ou `SYMBOL`. Para o tipo `TIMES`, o estilo pode ser `ROMAN`, `BOLD`, `ITALIC` ou `BOLDITALIC`. Para os tipos `HELVETICA` ou `COURIER`, o estilo pode ser omitido ou ser escolhido entre `BOLD`, `OBLIQUE` ou `BOLDOBLIQUE`. O tipo `SYMBOL` não precisa de especificação de estilo. Exemplo: `axesfont = [TIMES, ROMAN, 20]`.

color O mesmo que `colour`. Especifica a cor do gráfico a ser desenhado. Pode ser

aquamarine	black	blue	navy	coral	cyan	brown
gold	green	gray	grey	khaki	magenta	maroon
orange	pink	plum	red	sienna	tan	turquoise
violet	wheat	white	yellow			

ou ser definida na forma `COLOR(RGB,` *r, g, b*`)`, onde *r, g, b* são valores reais no intervalo [0, 1] que especificam a proporção de vermelho, verde e azul que compõem a cor. Exemplos: `color = yellow`; `color = COLOR(RGB, 0.3, 0.5, 0.8)`.

coords Indica o sistema de coordenadas utilizado. Pode ser `bipolar, cardioid, cartesian, elliptic, polar`, entre outros. O padrão é o `cartesian`.

discont Se for ajustado para `true`, evitará que sejam desenhadas linhas verticais nos pontos de descontinuidade do gráfico. Se for `false` (que é o padrão), as descontinuidades podem aparecer ligadas por segmentos de retas verticais.

filled Se for ajustado em `true`, a região entre o gráfico e o eixo x será pintada. Na forma padrão, é ajustado em `false`.

font Tipo de letra eventualmente utilizado no gráfico (como um título). Veja a opção `axesfont`, citada anteriormente, para uma listagem das opções.
Exemplo: `font = [TIMES,ITALIC,14]`

labels=[x,y] Define os nomes dos eixos. O padrão é utilizar os mesmos nomes das variáveis usadas na função fornecida.

labeldirections=[x,y] Especifica a direção dos rótulos que aparecem nos eixos. Os valores de x e y devem ser `HORIZONTAL` or `VERTICAL`. O padrão é `HORIZONTAL`.

labelfont Tipo de letra para os nomes dos eixos. Veja a opção `axesfont` anterior.

legend Legenda do gráfico. É útil quando desenham-se vários gráficos simultaneamente.

linestyle Define se o gráfico é construído pontilhado ou sólido. Pode ser um inteiro de 1 a 4 ou, equivalentemente, uma das palavras: `SOLID, DOT, DASH, DASHDOT`. O padrão é o estilo `SOLID`. Exemplo: `linestyle = 3` (que é o mesmo que `linestyle = DASH`).

numpoints Especifica o número de pontos utilizados para construir o gráfico. O padrão é 50 pontos. Exemplo: `numpoints = 200`

scaling Controla a proporção entre as escalas dos eixos. Pode ser `CONSTRAINED` ou `UNCONSTRAINED`. O padrão é `UNCONSTRAINED`.

style Se for ajustado em `POINT`, o gráfico será construído com pontos isolados. Se for ajustado em `LINE` (que é o padrão), os pontos serão ligados por segmentos de reta.

symbol Símbolo utilizado para marcar os pontos isolados. Pode ser um dos seguintes: `BOX, CROSS, CIRCLE, POINT` ou `DIAMOND`.

symbolsize Tamanho do símbolo utilizado para marcar pontos isolados no gráfico. O padrão é o valor 10.

thickness Largura das linhas usadas para desenhar o gráfico. Pode ser um valor inteiro maior ou igual a 0. O padrão é o valor 0.

tickmarks=[m,n] Especifica as quantidades mínimas de pontos marcados sobre os eixos. Para especificar a quantidade de determinado eixo, deve ser usada uma das opções `xtickmarks = m` ou `ytickmarks = n`.

title Título do gráfico fornecido como *string* (portanto, entre aspas). Se o título contiver mais de uma linha, um sinal de \n pode ser usado para sinalizar o final de cada linha. Exemplo: `title=`*"Gráfico da função f(x) = cos(5x)"*

titlefont Especifica o tipo de letra do título. Exemplo: `titlefont=[COURIER,BOLD,5]`

view=[xmin..xmax, ymin..ymax] Especifica as coordenadas máximas e mínimas da curva a ser mostrada. O padrão é mostrar toda a curva.
Exemplo: `view = [-2..1, -1..0.5]`

xtickmarks Indica a quantidade mínima de pontos a ser marcada sobre o eixo horizontal ou uma lista de coordenadas.
Exemplo: `xtickmarks = [-2, -1, 1, 3, 5, 7]`

ytickmarks Indica a quantidade mínima de pontos a ser marcada sobre o eixo vertical ou uma lista de coordenadas.
Exemplo: `ytickmarks = 4`

1.11 Construindo vários gráficos simultaneamente

Para construir vários gráficos em um mesmo sistema de eixos coordenados, a maneira mais simples é fornecer uma lista de funções `[f1, f2, ...]` em vez de só uma função. Nesses casos, as funções devem ter um domínio comum. As outras opções fornecidas ao comando `plot` também podem ser na forma de lista (`opção = [op1, op2, ...]`).

Exemplo 1.11 *Neste exemplo, construímos, em um mesmo sistema de eixos, as funções seno e cosseno. O gráfico da função seno é contruído com um traçado mais fino (*`thickness = 0`*) e cor azul (*`color = blue`*, que é o mesmo que* `color = COLOR(RGB, 0, 0, 1)`*), enquanto que, a função cosseno, é construída com um traçado mais grosso (*`thickness = 4`*) e cor vermelha (*`color = red`*, o mesmo que* `color = COLOR(RGB, 1, 0, 0)`*).*

```
> plot([sin(t),cos(t)],t=-Pi..Pi,title="Funcoes Trigonometricas Simples",
>      thickness=[0, 4], color = [blue, red], legend=["Seno", "Cosseno"]);
```

Outra maneira de construir vários gráficos simultaneamente é construir cada um deles separadamente e mostrá-los na tela usando o comando `display`, que faz parte do pacote `plots`. Cada gráfico pode ser construído com comandos do tipo

`variável := plot(...):`

para que os pontos obtidos sejam guardados em uma variável. É conveniente usar dois pontos no final desse comando para que a extensa relação dos pontos do gráfico não seja mostrada na tela. No final, é só usar um comando

`display(`*variável1, variável2,* `...)`

para que os gráficos sejam mostrados.

Exemplo 1.12 *Neste exemplo, construímos separadamente os gráficos das funções* $f(x) = e^x$ *e* $g(x) = \ln(x)$ *em um mesmo sistema de eixo. A função* $h(x) = x$ *também é construída com o traçado do gráfico mais fino.*

```
> graf1 := plot(exp(x), x=-4..4, y=-4..4, scaling=constrained,
>                                 color=red, thickness=2):
> graf2 := plot(ln(x), x=0..4, y=-4..4, scaling=constrained,
>                                 color=blue, thickness=2):
> graf3 := plot(x, x=-4..4, y=-4..4, scaling=constrained,
>                                 color=black, thickness=0):
> with(plots): display(graf1, graf2, graf3);
```

No pacote `plots`, há um comando bastante útil chamado `textplot`. Sua função é colocar mensagens (textos) em um sistema de coordenadas. Sua sintaxe é:

`textplot([`*x1, y1, "mensagem1"*`], [`*x2, y2, "mensagem2"*`], `*opções*`);`

onde *(x1, y1)* corresponde às coordenadas onde serão mostradas a *mensagem1*, etc.

As opções são as mesmas do comando `plot`, com apenas um acréscimo: a opção `align`, que pode assumir valor em um conjunto formado por `BELOW` (abaixo), `ABOVE` (acima), `LEFT` (esquerda) ou `RIGHT` (direita). O `align` corresponde ao alinhamento do texto com relação a um ponto escolhido. Exemplo: `align = ABOVE` .

1.12 Gráficos de curvas parametrizadas

Se a curva for definida por equações paramétricas, então seu gráfico pode ser construído com o comando `plot`. Para isso, basta formar uma lista com as equações envolvidas e a variação do parâmetro:

`> plot([`$f1(t), f2(t), \ldots, t = a..b$`], `*opções*`, ...);`

Exemplo 1.13 *Construímos o gráfico da* astróide *definida parametricamente por*

$$\begin{cases} x(t) &= \cos^3 t \\ y(t) &= \operatorname{sen}^3 t \end{cases}, \ 0 \leq t \leq 2\pi.$$

```
> plot([cos(t)^3, sin(t)^3, t=0..2*Pi], thickness=2, color=blue,
>                                       scaling=constrained);
```

É importante não esquecer de colocar a variação do t entre colchetes. Por exemplo, o comando

```
> plot([cos(t)^3, sin(t)^3], t=0..2*Pi, thickness=2, color=blue);
```

é muito parecido com o anterior, mas seu resultado é bem diferente: são construídos os gráficos das funções $y = \cos^3 t$ *e* $y = \operatorname{sen}^3 t$ *em um mesmo sistema de eixos.*

1.13 Gráficos tridimensionais

O Maple possui uma grande variedade de comandos para a construção de gráficos tridimensionais. Apresentamos, aqui, somente alguns comandos básicos.

O gráfico de uma função de duas variáveis reais x, y, definida por uma expressão algébrica $f(x, y)$, pode ser construído com um comando

`plot3d(`*f(x, y), x = a..b, y = c..d, op1, op2, ...*`)`,

onde

f(x, y)	é uma função real
$x = a..b$	é a variação da variável 1
$y = c..d$	é a variação da variável 2
op1, op2, ...	outras opções

Exemplo 1.14 *Neste exemplo, construímos o gráfico do parabolóide* $z = -x^2 - y^2$. *Definimos o intervalo de variação do x e do y como sendo o intervalo* $[-2, 2]$.

```
> plot3d(-x^2 - y^2, x = -2..2, y=-2..2);
```

Após a construção do gráfico na tela, ele pode ser girado com o auxílio do mouse: basta dar um simples clique com o botão esquerdo e, mantendo-o preso, girar o gráfico para uma nova posição. Pode-se também modificar o gráfico construído com os botões da barra de contexto (abaixo da barra de ferramentas).

1.14 Gráficos de superfícies parametrizadas

Em algumas áreas de estudo é muito comum o uso de superfícies definidas por equações paramétricas. Nestes casos, os pontos (x, y, z), que pertencem ao gráfico da superfície, satisfazem equações do tipo

$$\begin{cases} x &= f(u,v) \\ y &= g(u,v) \\ z &= h(u,v) \end{cases},$$

com (u, v) pertencente a um domínio do plano que depende de cada superfície.

Esse tipo de gráfico pode ser construído de modo semelhante aos construídos com o comando `plot3d`, devendo-se apenas fornecer as equações paramétricas em forma de lista, como no exemplo: `plot3d([f, g, h], x=a..b, y=c..d)`.

Cuidado: se usarmos chaves no lugar dos colchetes, como em `plot3d({f, g, h}, x=a..b, y=c..d)`, então, serão construídos, simultaneamente, os gráficos das três funções $f(x, y)$, $g(x, y)$ e $h(x, y)$, que não têm nada a ver com o gráfico da superfície parametrizada por $(f(x, y),\ g(x, y),\ h(x, y))$.

Exemplo 1.15 *Construir o* helicóide $(u\cos v,\ u\,\mathrm{sen}\, v,\ v)$ *com* $-6 \leq u \leq 6$ *e* $-8 \leq v \leq 8$.

```
> plot3d([u*cos(v), u*sin(v), v], u = -6..6, v = -8..8,
>                                 grid=[25, 50], shading=zhue);
```

Capítulo 2

Animações planas

O pacote `plots` possui três comandos para a construção de animações: `animate`, `animate3d` e `animatecurve`. Descrevemos brevemente cada um desses comandos e acrescentamos alguns exemplos.

Uma animação consiste, simplesmente, de uma seqüência de gráficos, mostrados consecutivamente. Isso gera uma ilusão de movimento e, normalmente, chama muito a atenção. Fica parecendo que os gráficos se movem na tela.

2.1 Animações bidimensionais

A sintaxe do `animate` é

`animate(`*F(x, t), x = a..b, t = c..d, opções*`)`

onde $F(x,t)$ é uma expressão (ou função) de duas variáveis, que, nos nossos exemplos consideraremos sempre como x e t. O $F(x,t)$ deve ser seguido pela variação do x e do t e, opcionalmente, por opções (em forma de igualdades) que são as mesmas do comando `plot`. A variação do x corresponde ao domínio das funções envolvidas na animação, enquanto que a variação do t corresponde às posições intermediárias. O valor $t = c$ corresponde ao gráfico inicial e $t = d$ corresponde ao gráfico final. O total de n gráficos contruídos é controlado com uma opção `frames =` n.

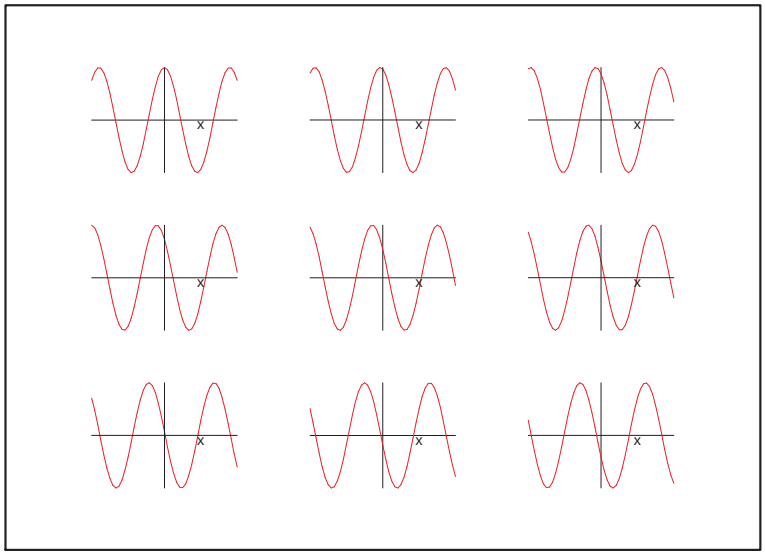
Exemplo 2.1 *Aqui mostramos a animação de uma reta e do gráfico de uma função logarítmica. Note que a diferença entre os comandos é apenas na troca da ordem das opções.*

```
> animate(log(x)+t, x=0..10, t=-3..3, frames=20);
>
> animate(log(x)+t, t=-3..3, x=0..10, frames=20);
```

Exemplo 2.2 *Neste exemplo, criamos uma animação que corresponde a uma seqüência de 9 gráficos, iniciando com* $y = \cos(x)$ *e terminando com* $y = \cos(x+2)$*. Essa seqüência corresponde aos gráficos de* $\cos(x+t)$ *com* t *variando no conjunto* $\{0, 2/8, 4/8, 6/8, 1, 10/8, 12/8, 14/8, 2\}$*, assumindo, assim, 9 valores igualmente espaçados de 0 a 2 (Figura 2.1). Escolhendo o domínio como sendo o intervalo* $[-7, 7]$ *e gráficos sem marcas nos eixos* x *e* y *(*`tickmarks = [0, 0]`*), digitamos o seguinte comando no aviso do Maple:*

```
> with(plots):
> animate(cos(x + t), x =-7..7, t=0..2, frames=9, tickmarks=[0,0]);
```

Figura 2.1:



Para iniciar a animação, deve-se pressionar em cima do gráfico inicial com o botão esquerdo do mouse*. Com isso, a barra de contexto muda e é possível dar início à animação pressionando-se o botão "*Play the animation*". Neste exemplo, a animação dá a impressão de que o gráfico do cosseno está se deslocando para a esquerda.*

Outra maneira de iniciar a animação é pressionar com o botão direito do mouse *em cima do gráfico inicial da animação. Com isso, deve aparecer um menu com as opções: Copy, Style, Legend, Axes, Projection, Animation e Export As. Escolhendo Animation, aparece outro menu de opções: Play, Next, Backward, Faster, Slower, Continuous. Escolhendo Play, terá início a animação na tela. A opção Continuous permite que a animação seja repetida indefinidamente – neste caso, para parar, é só repetir esse procedimento que, no lugar da opção Play, aparecerá uma opção Stop.*

Ao pressionar com o botão direito do mouse em cima do gráfico inicial e escolhendo a opção "*Export As*", é possível salvar a animação em vários formatos, inclusive GIF animado.

Exemplo 2.3 *Para fazer $f(x)$ "transformar-se" em $g(x)$, basta usar, no* `animate`*, uma função $F(x,t) = (1-t)\,f(x) + t\,g(x)$ e $0 \leq t \leq 1$. Neste exemplo, "transformamos" a função $|x| - 1$ na função $-x^2 + 1$, usando um total de 16 gráficos (Figura 2.2).*

```
> with(plots):
> animate((abs(x) - 1)*(1 - t) + (-x^2 + 1)*t, x=-2..2, t=0..1,
>                                    frames = 16, tickmarks = [0, 0]);
```

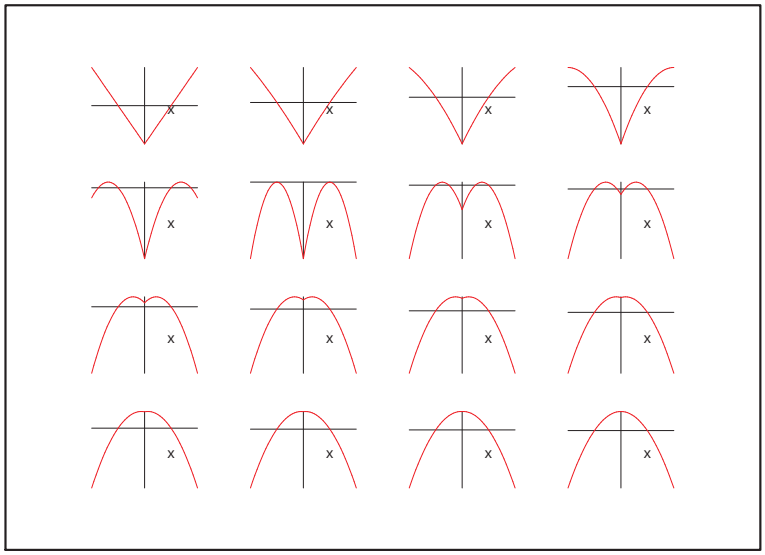
Como no exemplo anterior, é só escolher a opção Animation/Play ao pressionar, com o botão direito do mouse, em cima do gráfico inicial da animação.

2.2 Animatecurve

Com o comando `animatecurve`, é possível observarmos a construção de uma curva por etapas. Sua sintaxe é parecida com a do `plot`:

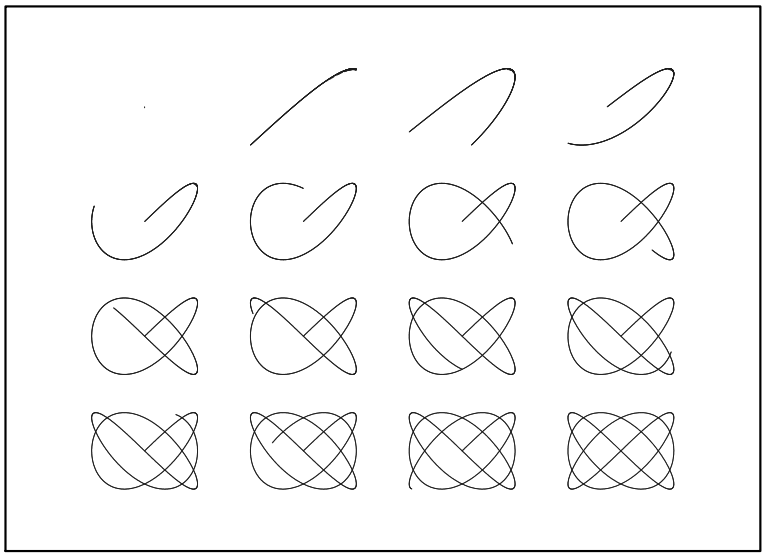
`animatecurve`(*f, domínio, opções,* `frames =` *n*)

Figura 2.2:



onde f pode ser uma função de uma variável real ou uma lista de funções (neste caso, representando uma curva em forma paramétrica).

Figura 2.3:



Exemplo 2.4

```
>with(plots):
> f := t -> exp(cos(t)) - 2*cos(4*t) + sin(t/12)^5:
> animatecurve([f(t)*cos(t), f(t)*sin(t), t=0..50], axes=NONE,
>                           thickness=2, numpoints=1000,
>                           frames = 30, scaling=constrained);
>
>
> g := animatecurve([sin(3*t),sin(4*t), t=0..2*Pi], axes=NONE,
>         thickness=2, color=BLUE, numpoints=1000):
>
```

```
> display(g);
```

2.3 Animações com o comando `display`

O comando `display` do pacote `plots` também pode ser usado como uma forma eficiente de construção de animações. Para isso, basta utilizá-lo com a opção `insequence=true`.

`display(`*imagem1, imagem2, imagem3,* `..., insequence = true)`

Dessa forma, todas as imagens *imagem1, imagem2, ...* são mostradas uma seguida da outra. Se a opção `insequence=true` não for acrescentada, todas as imagens são mostradas ao mesmo tempo, em uma única imagem.

Exemplo 2.5

```
> restart:
> with(plots):
> f1 := x -> cos(x): a := -3: b := 3: c := -1: d := 1:
> f2 := x -> 1 - x^2/2:
> f3 := x -> 1 - x^2/2 + x^4/24:
> f4 := x -> 1 - x^2/2 + x^4/24 - x^6/720:
> f5 := x -> 1 - x^2/2 + x^4/24 - x^6/720 + x^8/40320:
```

Essas funções também poderiam ter sido definas com o comando `taylor`*:*

```
> f5 := unapply(convert(taylor(cos(x), x=0, 10), polynom), x);
```

```
>
> g1 := plot(f1(x), x=a..b, y=c..d, thickness=3, color=RED):
> g2 := plot(f2(x), x=a..b, y=c..d, thickness=2, color=BLUE):
> g3 := plot(f3(x), x=a..b, y=c..d, thickness=2, color=GREEN):
> g4 := plot(f4(x), x=a..b, y=c..d, thickness=2, color=YELLOW):
> g5 := plot(f5(x), x=a..b, y=c..d, thickness=2, color=BLACK):
>
> display(g1, g2, g3, g4, g5);
```

Todos os gráficos são mostrados em uma única imagem na Figura 2.4.

```
> display(g1, g2, g3, g4, g5, insequence=true);
```

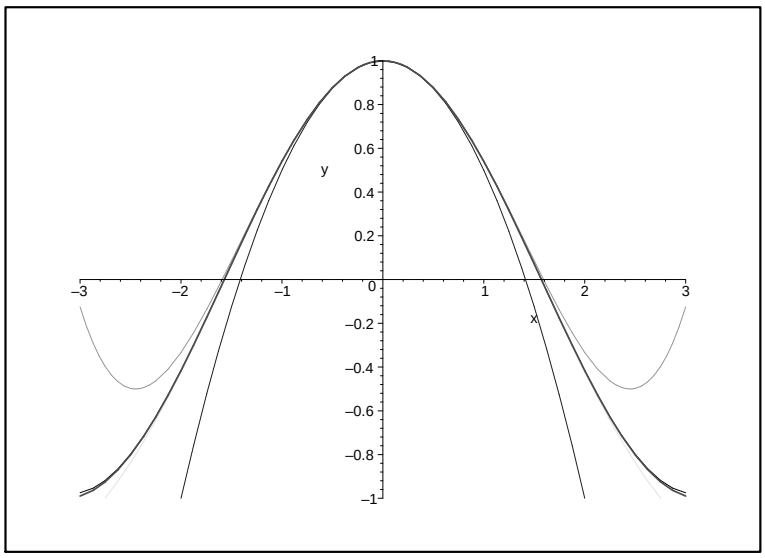
Dessa forma, os gráficos são mostrados um após o outro.

Um tipo mais interessante de animação pode ser construído se as imagens forem sendo agrupadas com um comando `display`*. Assim, é possível mostrar somente a primeira, depois a primeira com a segunda, depois a primeira com a segunda e a terceira, e assim por diante.*

```
>
> gr1 := g1:
> gr2 := display(g1, g2):
> gr3 := display(g1, g2, g3):
```

```
> gr4 := display(g1, g2, g3, g4):
> gr5 := display(g1, g2, g3, g4, g5):
> display(gr1, gr2, gr3, gr4, gr5, insequence=true);
>
```

Figura 2.4:



2.4 Círculo osculador

Dada uma curva parametrizada por $\alpha(t) = (f(t), g(t))$, a curvatura em t é $\kappa(t) = \frac{-f''(t)g'(t)+f'(t)g''(t)}{\sqrt{(f'(t)^2+g'(t)^2)^3}}$ e o centro de curvatura é o ponto

$$C = \left(f(t) - \frac{1}{\kappa(t)} \frac{g'(t)}{\sqrt{f'(t)^2 + g'(t)^2}},\ g(t) + \frac{1}{\kappa(t)} \frac{f'(t)}{\sqrt{f'(t)^2 + g'(t)^2}} \right).$$

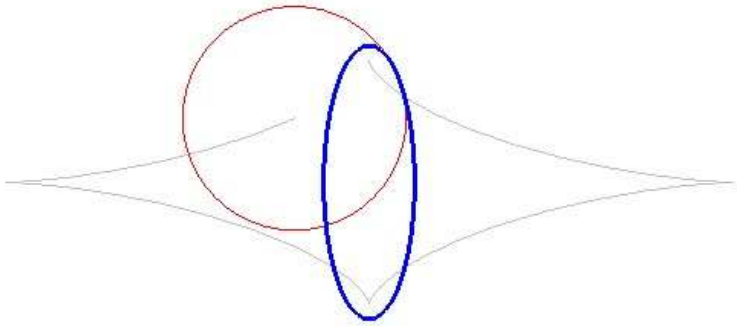
O círculo osculador a $\alpha(t)$ é a circunferência de centro C e raio $1/kappa(t)$. Neste exemplo, calculamos círculos osculadores em diversos pontos e mostramos um atrás do outro em forma de animação, como se o círculo osculador fosse passeando ao longo da curva.

Quando o círculo se move ao longo da curva, o centro de curvatura descreve uma curva chamada *evoluta*. Veja a Figura 2.5.

Alguns comandos utilizados foram os seguintes:

- `for w from 1 to n` – faz com que uma variável w varie de 1, 2, 3, ... até n.
- `gr[w] := display(gr0, gr1, gr2)` – acumula gr0, gr1 e gr2 em uma única imagem gr[w].
- `seq(gr[w], w=1..n)` – seqüência formada por gr[1], gr[2], gr[3], ... gr[n].
- `display(seq(gr[w], w=1..n), insequence=true)` – mostra todos os termos da seqüência um atrás do outro.

Figura 2.5:



Exemplo 2.6

```
> restart:
> with(plots):
>
> f := t -> 3*cos(t): g := t -> sin(t): a:= 8: n := 100:
> k := t -> (-(D@@2)(f)(t)*D(g)(t) +
>                    D(f)(t)*(D@@2)(g)(t))/(D(f)(t)^2+D(g)(t)^2)^1.5:
> xc := t -> f(t) - 1/k(t)*D(g)(t)/sqrt(D(f)(t)^2 + D(g)(t)^2):
> yc := t -> g(t) + 1/k(t)*D(f)(t)/sqrt(D(f)(t)^2 + D(g)(t)^2):
> gr0 := plot([f(t), g(t), t=0..2*Pi], x=-a..a, y=-a..a,
>                      scaling=CONSTRAINED, color=BLUE, thickness=3):
> for w from 1 to n do
>   gr1 := plot([1/k(w*6.28/n)*cos(t) + xc(w*6.28/n),
>                 1/k(w*6.28/n)*sin(t) + yc(w*6.28/n), t = 0..2*Pi],
>                 x=-a..a, y=-a..a, scaling=CONSTRAINED, axes=NONE):
>   gr2 := plot([xc(t), yc(t), t = 0..2*Pi*w/n], x=-a..a, y=-a..a,
>                                  scaling=CONSTRAINED, color=GRAY):
>   gr[w] := display(gr0, gr1, gr2):
> end do:
>
> display(seq(gr[w], w=1..n), insequence=true);
```

2.5 Problemas de máximos e mínimos

Animações relacionadas com problemas de máximos e mínimos podem ser bastante interessantes. Constrói-se lado a lado ou em uma mesma figura uma ilustração do problema e a função da qual se quer maximizar ou minimizar. Assim, o máximo ou o mínimo podem ser determinados de forma "experimental".

Exemplo 2.7 *Este exemplo é uma adaptação de um exemplo mostrado em [1]. Queremos calcular a área máxima de um retângulo inscrito em um triângulo retângulo de base B e altura H. Algebricamente, corresponde a determinadar o ponto de máximo do polinômio* $p(x) = \frac{Hx(B-x)}{B}$. *Veja a Figura 2.6.*

O comando `convert(`x`, string)` *converte o valor de* x *para o formato string e* `cat(`$s1$`,` $s2$`)` *faz a operação de concatenação dos strings* $s1$ *e* $s2$*, ou seja, emenda os strings* $s1$ *e*

*s*2..

```
> restart:
> with(plots):
> with(plottools):
> B := 7: H := 10: # base e altura do triangulo
> n := 30: # numero de graficos da animacao
>
> triangulo := display(polygon([[0, 0], [B, 0], [B, H]],
>                                        color=BLUE, thickness=2)):
>
> retangulo := proc(k::integer)
>   local g, graf, f, xi, area, r, mens;
>   xi := k * B/n;
>   f := piecewise(x <= xi + 0.001, H/B*x*(B-x), x > xi, 0):
>   graf := plot(f(x), x=0..B, color=BLACK, scaling=constrained,
>                                      thickness=2,  discont=true):
>   area := H/B*xi*(B-xi);
>   r := display(rectangle([xi, H/B*xi], [B, 0], color=RED,
>                                                    thickness=2));
>   mens := textplot([
>           [0.5, H + 0.8, cat("Largura = ",
>                               convert(evalf(B-xi, 4), string))],
>           [0.5, H, cat("Altura  = ",
>                              convert(evalf(H/B*xi,4), string))],
>           [0.5, H - 0.8, cat("Area    = ",
>                               convert(evalf(area,4), string))]],
>           align=RIGHT, color=RED);
>   display(r, triangulo, graf, mens):
> end proc:
>
> display(seq(retangulo(i), i=0..n), insequence=true);
```

2.6 Coordenadas polares

Construímos os gráficos das funções $f(x) = 4 + 4\cos(x)$, $f(x) = 3 + \operatorname{sen}(5x)$ e $f(x) = 2\pi - x$ de duas maneiras: usando coordenadas cartesianas e usando coordenadas polares. Veja as Figuras 2.7, 2.8 e 2.9.

Para mostrar os gráficos lado a lado, utilizados uma array de gráficos: `G := array(1..k)`. Com isso, é possível definir k gráficos G[1], G[2], ..., G[k].

Exemplo 2.8

```
> with(plots):
> G := array(1..2):
> n := 90:
> f := x -> 4+4*cos(x):   # opcao 1
> # f := x -> 3+sin(5*x): # opcao 2
> # f := x -> 2.0*Pi - x: # opcao 3
```

Figura 2.6:

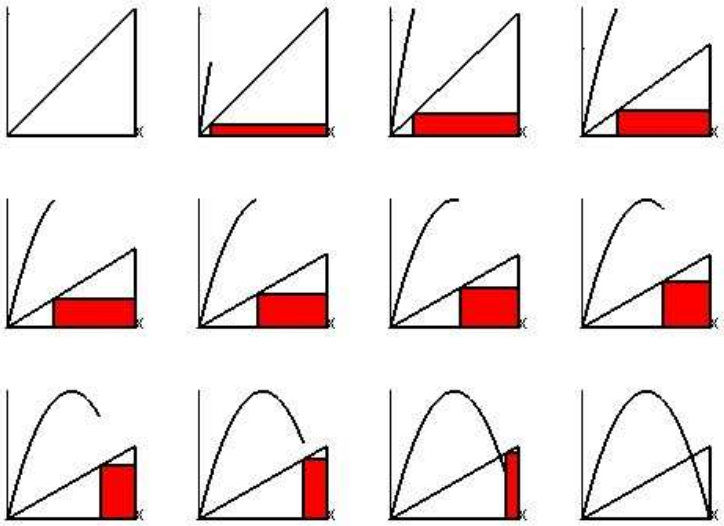


Figura 2.7:

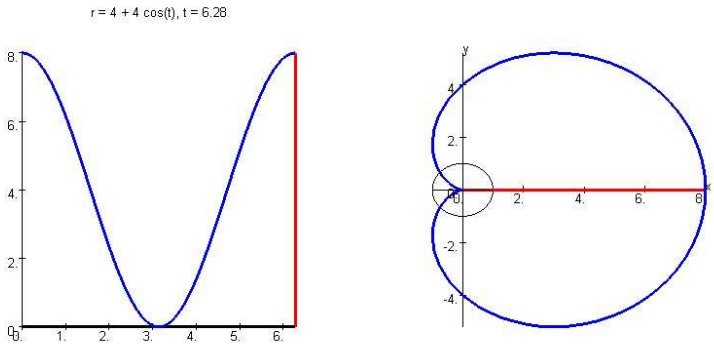
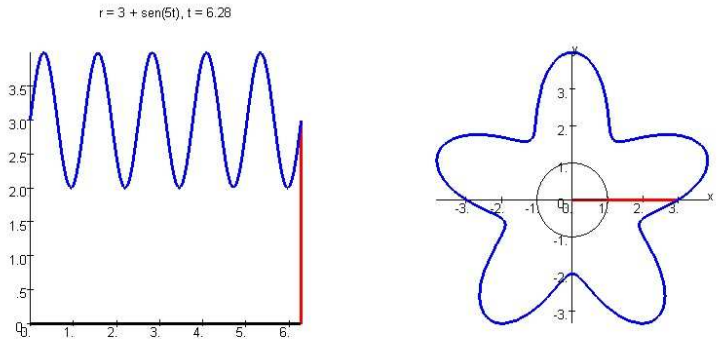
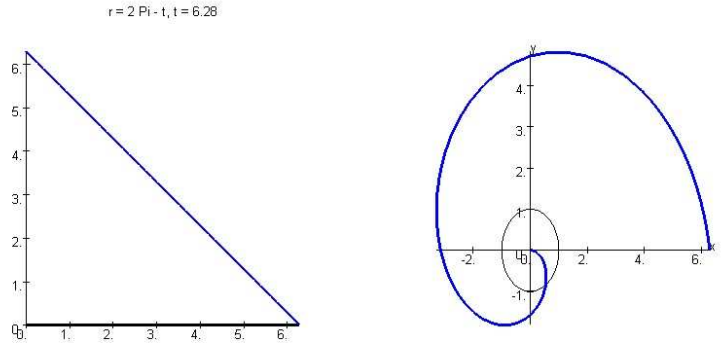


Figura 2.8:




```
> g1 := plot([cos(t), sin(t), t=0..2*Pi], x=-2..2, y=-2..2,
>                                scaling=CONSTRAINED, color=BLACK):
>
> for k from 1 to n do
>   angulo := 2.0*k*Pi/n: X := f(angulo): Y := sin(angulo):
>   lista := [[f(t)*cos(t), f(t)*sin(t), t=0..angulo],
```

Figura 2.9:




```
>                          [cos(angulo)*t, sin(angulo)*t, t=0..1],
>                  [cos(angulo)*t, sin(angulo)*t, t=0..(X+0.001)]]:
>   opcoes := scaling=CONSTRAINED, thickness=[3,1,3],
>                                          color=[BLUE,BLACK,RED]:
>   gr[k] := plot(lista, opcoes):
> end do:
>
> g2 := display(seq(gr[k], k=1..n), insequence=true):
>
> G[2] := display(g1, g2):
>
> for k from 1 to n do
>   angulo := 2.0*k*Pi/n:
>   X := f(angulo)+0.001: Y := f(angulo)+0.001:
>   graf2 := [[t, f(t), t=0..angulo], [t, 0, t=0..angulo],
>                                            [angulo, t, t=0..X]];
>   opcoes2 := thickness=[3,3,3], color=[BLUE,BLACK,RED],
>                                             scaling=CONSTRAINED;
>   gr2[k] :=plot(graf2,opcoes2,title=cat("r = 4 + 4 cos(t), t = ",
>                               convert(evalf(angulo, 3),string))):
> end do:
>
> G[1] := display(seq(gr2[k], k=1..n), insequence=true):
>
> display(G);
```

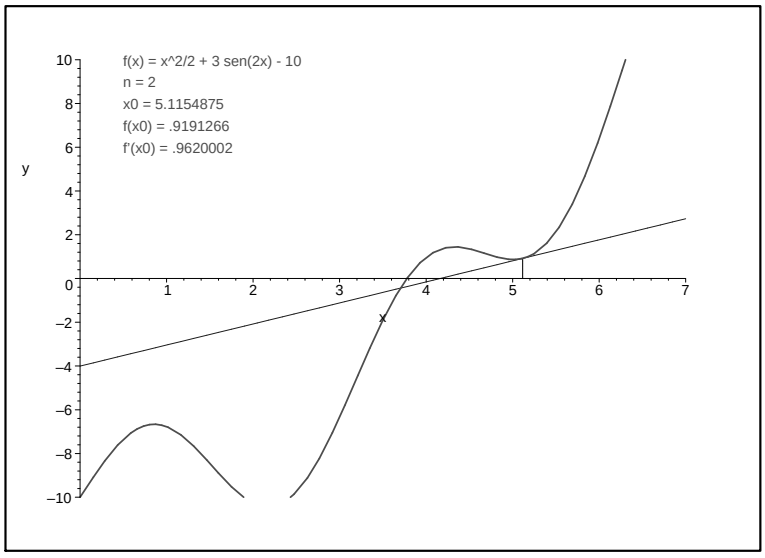
2.7 Método de Newton

Exemplo 2.9 *O método de Newton pode ser utilizado para determinar uma raiz aproximada da equação* $f(x) = 0$. *Para isso, é utilizada a interseção* x_1 *da reta tangente ao gráfico de* $f(x)$ *no ponto* $(x_0, f(x_0))$ *que é dada por* $x_1 = x_0 - f(x_0)/f'(x_0)$. *Veja Figura 2.10.*

```
> restart;
> with(plots):
> f := x -> x^2/2 - 10 + 3*sin(2*x):
```

```
> fstr := "f(x) = x^2/2 + 3 sen(2x) - 10":
> x0 := 5.5: a := 0: b := 7: n := 10:
>
> # Outra opcao:
> # f := x -> 0.2*x^3 - 0.5*x^2 + x - 5:
> # fstr := "f(x) = x^3/5 - x^2/2 + x - 5":
> # x0 := 1.0: a := 0: b := 10: n := 10:
>
> G0 := plot(f(x), x=a..b, y=-10..10, thickness=3, color=RED):
>
> for k from 1 to n do
>   x1 := x0 - f(x0)/D(f)(x0);
>   g1 := plot([x0, t, t=0..f(x0)], color=BLUE):
>   g2 := plot(f(x0) + (x - x0)*D(f)(x0), x=a..b, color=BLUE):
>   g3 := textplot([
>      [0.5, 10.0, fstr],
>      [0.5, 9.0, cat("n = ", convert(k, string))],
>      [0.5, 8.0, cat("x0 = ", convert(evalf(x0, 8), string))],
>      [0.5, 7.0, cat("f(x0) = ",convert(evalf(f(x0), 8),string))],
>      [0.5, 6.0, cat("f'(x0) = ",
>                          convert(evalf(D(f)(x0), 8), string))]],
>      color = RED, align=RIGHT):
>   graf[k] := display(g1, g2, g3):
>   x0 := x1;
> end do:
>
> G1 := display(seq(graf[k], k=1..n), insequence=true):
> display(G0, G1);
```

Figura 2.10:

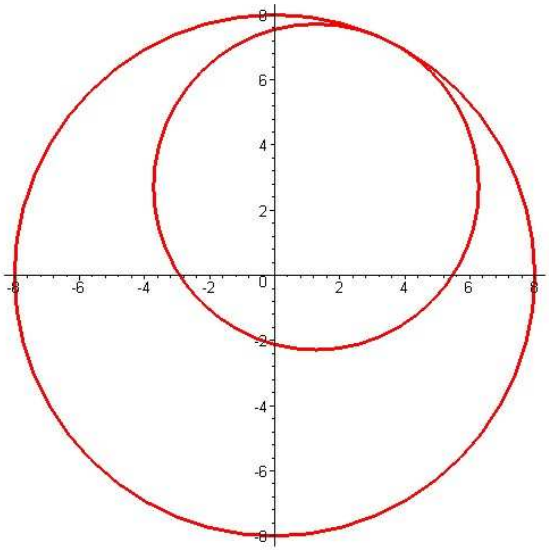


2.8 Hipotrocóides

Exemplo 2.10 *Mostramos um círculo menor de raio b girando por dentro de um círculo maior de raio a. Veja a Figura 2.11.*

```
> restart:
> with(plots):
> g1 := plot([a*cos(t), a*sin(t), t=0..6.29], thickness=3):
> n := 50:
> for k from 0 to n do
>   angulo := 2.0*Pi*k/n;
>   g[k] := plot([(a-b)*cos(angulo) + b*cos(t), (a-b)*sin(angulo)
>                              + b*sin(t), t=0..6.29], thickness=3):
> end do:
>
> g2 := display(seq(g[k], k=0..n), insequence=true):
>
> display(g1, g2);
```

Figura 2.11:



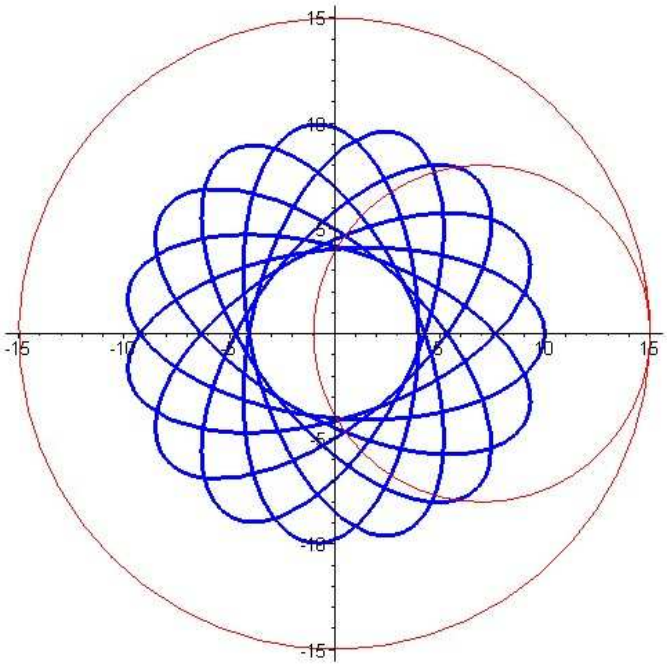
Exemplo 2.11 *Uma hipotrocóide é uma curva parametrizada por* $\alpha(t) = ((a-b)\cos(t) + r\cos(\frac{(a-b)t}{b}), (a-b)\,\mathrm{sen}\,(t) - r\,\mathrm{sen}\,(\frac{(a-b)t}{b}))$ *onde t varia de* 0 *até* 2π *vezes o denominador de a/b. É obtida pela trajetória de um ponto em um pcírculo menor que vai girando sem deslizar por dentro de um círculo maior. O valor de a corresponde ao raio do círculo maior, o b é o raio do círculo menor e r é a distância de um ponto do círculo menor ao centro.*

Se o círculo menor girar por fora do círculo maior, obtemos uma epitrocóide. Este caso corresponde a um valor de b negativo. Veja as Figuras 2.12 e 2.13.

```
> with(plots):
> a := 15: b := 8: r := 3:
```

```
>
> f1:= t -> (a - b)*cos(t) + r*cos((a - b)/b*t):
> f2 := t -> (a - b)*sin(t) - r*sin((a - b)/b*t):
>
> h := plot([f1(t), f2(t), t=0..2*Pi*denom(a/b)],
>                                  scaling=CONSTRAINED, axes=NONE):
> g1 := plot([a*cos(t), a*sin(t), t=0..6.5], axes=NONE):
>
> n := 120:
> for k from 0 to n do
>   angulo := denom(a/b)*2.0*Pi*k/n+0.0001;
>   g[k] := plot([[(a-b)*cos(angulo) + b*cos(t), (a-b)*sin(angulo)
>   +b*sin(t),t=0..6.5],[(a-b)*cos(angulo)+t*r*cos((a-b)/b*angulo),
>   (a-b)*sin(angulo)-t*r*sin((a-b)/b*angulo), t=0..1 ],
>   [f1(t),f2(t),t=0..angulo]],color=[RED,BLACK,BLUE],
>                                    thickness=[1,1,2], axes=NONE):
> end do:
> g2 := display(seq(g[k], k=0..n), insequence=true):
> display(g1, g2);
```

Figura 2.12:

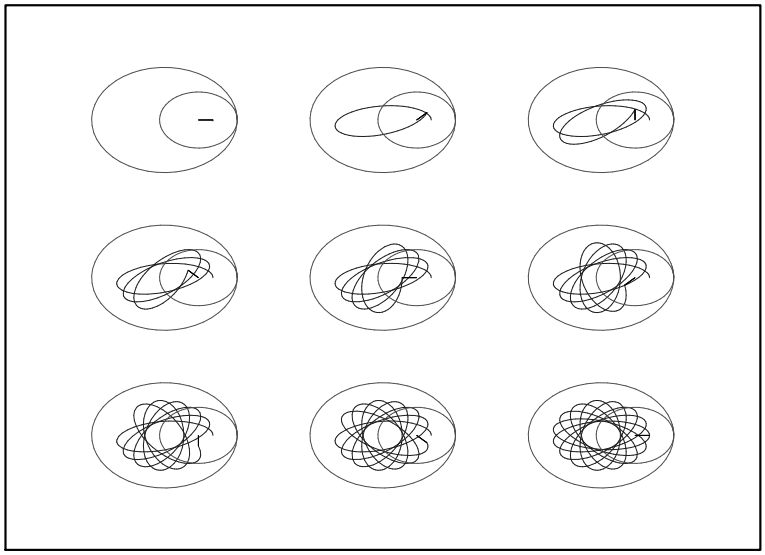


2.9 Funções trigonométricas

Construímos duas animações para ilustrar as definições das funções seno e cosseno. Para isso, são mostrados lado a lado um ciclo trigonométrico e os gráficos dessas funções.

Exemplo 2.12 *Inicialmente, ilustramos a definição do seno. Um ponto percorre um círculo de raio unitário, e, em cada posição, é mostrado um segmento ligando esse ponto*

Figura 2.13:



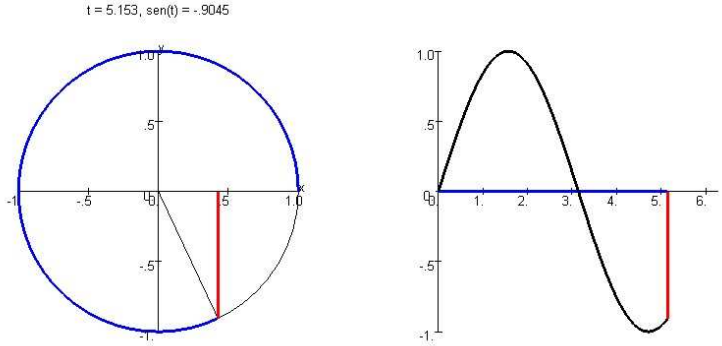
ao eixo dos x. Em um gráfico ao lado, é construído um gráfico com as medidas desse segmento, ou seja, é construído o gráfico da função seno.

```
> restart:
> with(plots):
>
> G := array(1..2):
> g1 := plot([cos(t), sin(t), t=0..2*Pi], x=-2..2, y=-2..2,
>                                 scaling=CONSTRAINED, color=BLACK):
> n := 100:
> for k from 1 to n do
> xcs := cos(2.0*k*Pi/n); yse:= 0.0001+sin(2.0*k*Pi/n);
>   gr[k] := plot([[cos(t), sin(t), t=0..k*2*Pi/n],
>                  [xcs, t, t=0..yse],
>                  [cos(2.0*k*Pi/n)*t, sin(2.0*k*Pi/n)*t, t=0..1]],
>                  thickness=[3,3,1], color=[BLUE, RED,BLACK],
>                  scaling=CONSTRAINED,
>   title = cat("t = ", convert(evalf(2.0*k*Pi/n, 4), string), ",
>        sen(t) = ", convert(evalf(sin(2.0*k*Pi/n), 4), string))):
> end do:
>
> g2 := display(seq(gr[k], k=1..n), insequence=true):
>
> G[1] :=  display(g1, g2):
>
> for k from 1 to n do yse := 0.0001+sin(2.0*k*Pi/n);
>   sen[k] := plot([[t, sin(t), t=0..k*2*Pi/n],
>              [k*2.0*Pi/n, t, t=0..yse], [t,0,t=0..2.0*Pi*k/n ]],
>                           x=-2..10, y=-2..2, thickness=[3,3,3],
>                 color=[BLACK, RED, BLUE], scaling=CONSTRAINED):
> end do:
```

```
>
> G[2] := display(seq(sen[k], k=1..n), insequence=true):
>
> display(G);
```

Veja a Figura 2.14.

Figura 2.14:

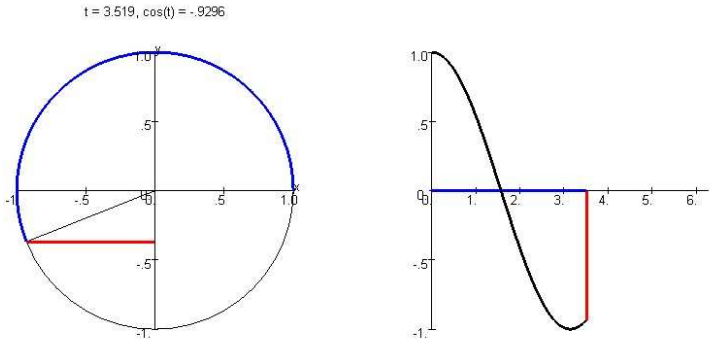


Exemplo 2.13 *Ilustramos agora a definição da função cosseno.*
Veja a Figura 2.15.

```
> with(plots):
> G := array(1..2):
> n := 100:
>
> g1 := plot([cos(t), sin(t), t=0..2*Pi], x=-2..2, y=-2..2,
>                                  scaling=CONSTRAINED, color=BLACK):
>
> for k from 1 to n do
>   angulo := 2.0*k*Pi/n: X := cos(angulo): Y := sin(angulo):
>   lista := [[cos(t), sin(t), t=0..angulo],
>   [cos(angulo)*t, sin(angulo)*t, t=0..1], [t, Y, t=0..(X+0.001)]]:
>   opcoes := scaling=CONSTRAINED, thickness=[3,1,3],
>                                              color=[BLUE,BLACK,RED],
>   title = cat("t = ", convert(evalf(2.0*k*Pi/n, 4), string), ",
>           cos(t) = ", convert(evalf(cos(2.0*k*Pi/n), 4), string)):
>   gr[k] := plot(lista, opcoes):
> end do:
>
> g2 := display(seq(gr[k], k=1..n), insequence=true):
>
> G[1] := display(g1, g2):
>
> for k from 1 to n do
>   angulo := 2.0*k*Pi/n: X := cos(angulo)+0.001:
>                                          Y := sin(angulo)+0.001:
>   graf2 := [[t, cos(t), t=0..angulo], [t, 0, t=0..angulo],
```

```
>                                                [angulo, t, t=0..X]];
>   opcoes2 := thickness=[3,3,3], color=[BLACK,BLUE,RED],
>                                                 scaling=CONSTRAINED;
>   gr2[k] := plot(graf2, opcoes2):
> end do:
>
> G[2] := display(seq(gr2[k], k=1..n), insequence=true):
>
> display(G);
```

Figura 2.15:



2.10 Definição de integral

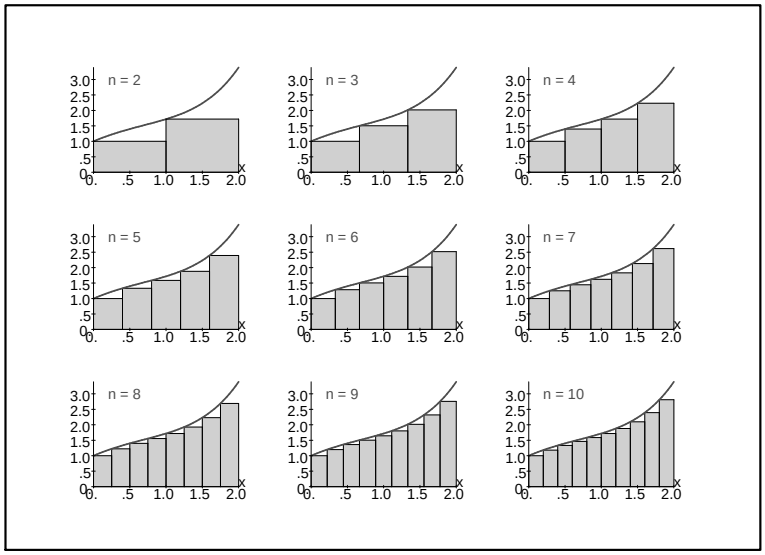
Exemplo 2.14 *Ilustramos a definição de integral através dos gráficos de somas de Riemann.*

Dada uma função definida em um intervalo (por exemplo, $f(x) = e^x - x^2$ *no intervalo* $[a, b] = [0, 2]$*) e escolhido um valor para* n*, dividimos o intervalo* $[a, b]$ *em* n *subintervalos de comprimento* $\Delta x = (b-a)/n$ *e calculamos o somatório* $\sum_{k=0}^{n-1} f(a + k\frac{b-a}{n})\Delta x$*. Quanto maior o valor de* n*, mais próximo da área sob o gráfico será o valor do somatório.*

O gráfico da soma de Riemann é feita com um comando `leftbox` *e o valor do somatório é calculado com um* `leftsum`*, ambos do pacote* `student`*. (Figura 2.16).*

```
> with(plots): with(student):
> nmin := 2: nmax := 100:
> a := 0: b := 2:
> f := x -> exp(x)-x^2:
>
> for k from nmin to nmax do
>   t[k] := textplot([
>      [0.2, 3, cat("n = ", convert(k, string))],
>      [0.2, 2.8, cat("Soma = ", convert(evalf(leftsum(f(x),
>                                        x=a..b, k), 3), string))],
>      [0.2, 2.6, cat("Integral = ",
>      convert(evalf(int(f(x), x=a..b), 3), string))]],
>      align=RIGHT, color=RED):
```

Figura 2.16:




```
>   g[k] := display(t[k], leftbox(f(x), x=a..b, k,
>                                          color=RED, thickness=3)):
> end do:
>
> display(seq(g[k], k=nmin..nmax), insequence=true);
```

2.11 Animações relacionadas com fenômenos físicos e equações diferenciais

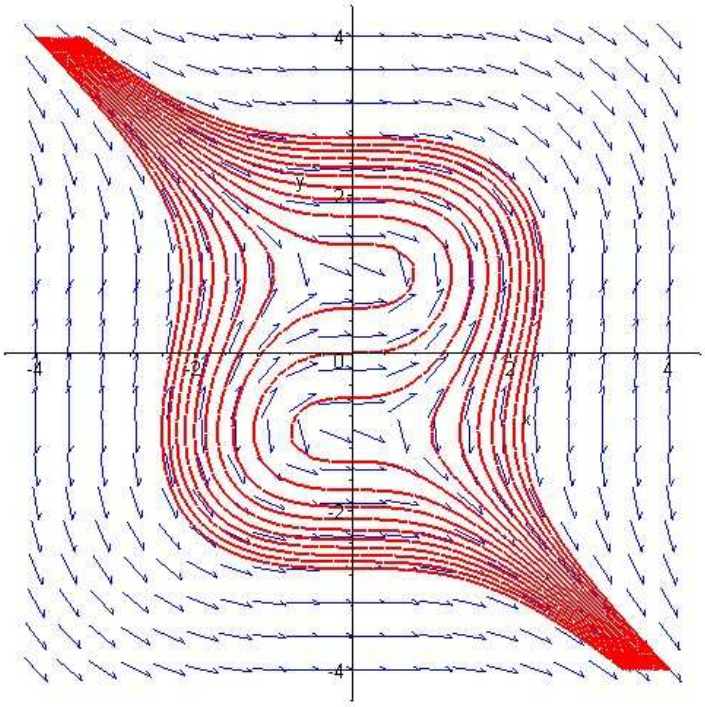
Inúmeros fenômenos físicos podem ser representados por animações, não só fenômenos mecânicos, mas também de outras áreas como ótica, ondulatória etc. Ilustramos aqui animações das soluções de uma equação diferencial (Figura 2.17) e de um choque de ondas (Figura 2.18).

Exemplo 2.15

```
> with(plots):
> with(DEtools):
> eqdif := diff(y(x), x) = x^2/(1- y(x)^2):
> direcoes := dfieldplot(eqdif, y(x), x=-4..4, y=-4..4, color=blue):
> sol := dsolve(eqdif, y(x), implicit):
> sol := subs(_C1 = (k-8)/2, sol):
> for k from 0 to 16 do
>   graf[k] := implicitplot(sol, x=-4..4, y=-4..4, thickness=2,
>                                                  numpoints=1500):
> end do:
> gsol := display(seq(graf[k], k=0..16), insequence=true):
> display(direcoes, gsol);
```

O trecho a partir do `for k ...` *pode ser substituído pelo trecho a seguir. Com isso, as soluções vão sendo acumuladas em vez de mostradas uma a uma como no trecho anterior.*

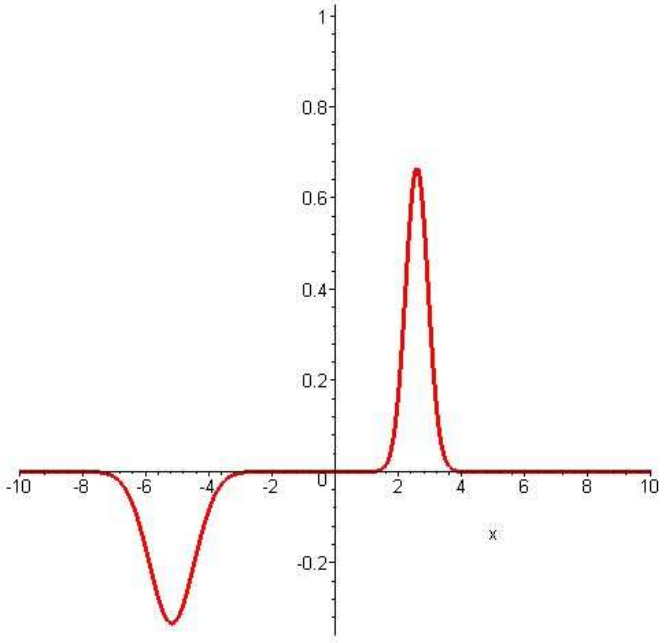
Figura 2.17:




```
> for k from 0 to 16 do
>   graf2[k] := display(seq(graf[j], j=0..k)):
> end do:
> gsol2 := display(seq(graf2[k], k=0..16), insequence=true):
> display(direcoes, gsol2);
```

Exemplo 2.16 *Neste exemplo, é simulado a propagação de duas ondas (Figura 2.18).*

Figura 2.18:



```
> with(plots):
> f1 := (x, t) -> exp(-(x - v1*t)^2) +
>                          (v2 - v1)/(v2 + v1)*exp(-(x + v1*t)^2):
> f2 := (x, t) -> 2*v2/(v1 + v2)*exp(-(v1*(x - v2*t)/v2)^2):
> f := (x, t) -> f1(x, t)*(1 - Heaviside(x)) +
>                                          f2(x, t)*Heaviside(x):
> v1 := 2: v2 := 1:
> animate(f(x, t), x=-10..10, t=-10..10, frames=90, thickness=3,
>                                                  numpoints=1000);
```

2.12 Convergência de uma série de funções

Figura 2.19:

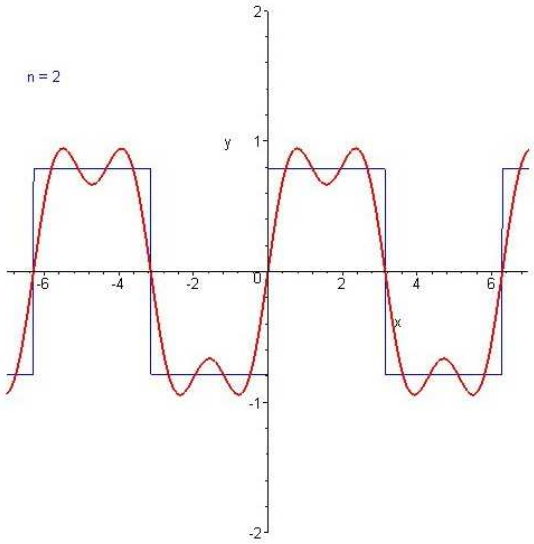
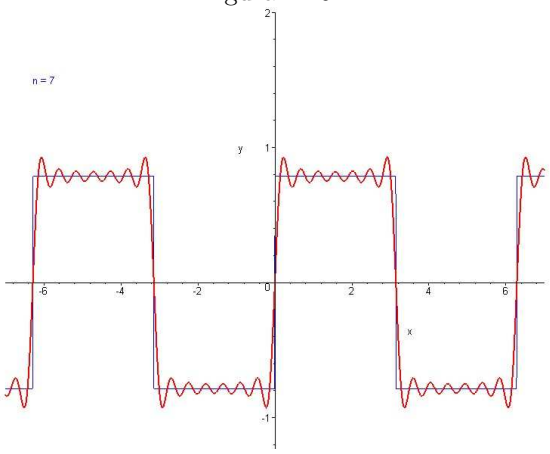
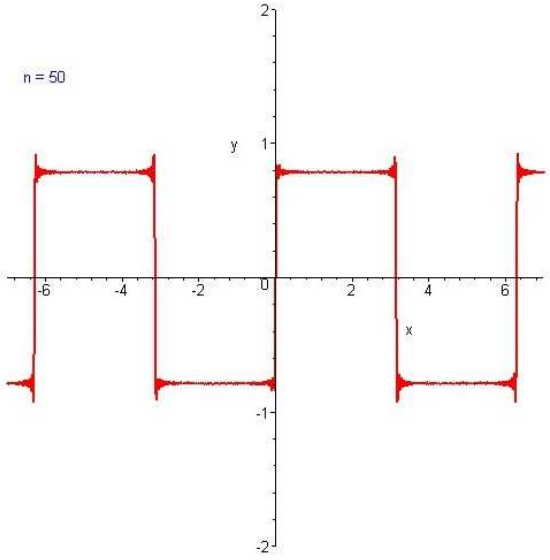


Figura 2.20:



Fazemos os gráficos das somas parciais que mostram a convergência de uma série de funções. Veja Figuras 2.19, 2.20 e 2.21.

Figura 2.21:



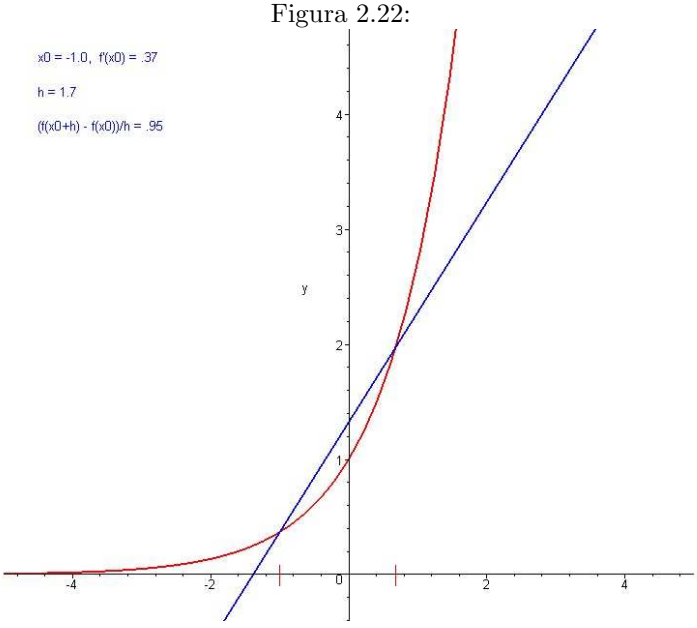

```
> with(plots):
>
> soma := proc(n::integer)
>   local k, s, p;
>   s := 0;
>   for k from 1 to n do
>     s := s + sin((2*k-1)*x)/(2*k-1);
>   end do;
>   return unapply(s, x);
> end proc:
>
> h := piecewise(x < -2*evalf(Pi), -evalf(Pi)/4, x >= -2*evalf(Pi)
>      and x < -evalf(Pi), evalf(Pi)/4, x >= -evalf(Pi) and x < 0,
>             -evalf(Pi)/4, x >= 0 and x < evalf(Pi), evalf(Pi)/4,
>                x >= evalf(Pi) and x < 2*evalf(Pi), -evalf(Pi)/4,
>                                  x >= 2*evalf(Pi),  evalf(Pi)/4):
> fundo2 := plot(h(x), x=-7..7, y=-2..2, color=BLUE):
>
> for k from 1 to 50 do
>   funcao[k] := soma(k);
>   gr := display(plot(funcao[k](x), x=-7..7, y=-2..2,thickness=2,
>                                          numpoints=300), fundo2):
> texto := textplot([-6, 1.5, cat("n = ", convert(k, string))],
>                                                    color=BLUE);
> graf[k] := display(gr, texto);
> end do:
>

> display(seq(graf[k], k=1..50), insequence=true);
```

2.13 Definição de derivada

Exemplo 2.17 *Ilustramos a definição de derivada de uma função através da convergência de várias retas secante ao gráfico que converge para a reta tangente (Figura 2.22).*

```
> with(plots):
> f := x -> exp(x):
> g1 := plot(f(x), x=-5..5, y=-1..5, thickness=2):
> x0 := -1.0: n := 50: p := 2.0:
> for k from 1 to n  do
>   graf := plot(f(x0) +
>               (f(x0+(1+n-k)*p/n) - f(x0))/((1+n-k)*p/n)*(x - x0),
>                       x=-5..5, y=-1..5, color=BLUE, thickness=2);
>   texto1 := textplot([[-4.5, 4.5, cat("x0 = -1.0,  f'(x0) = ",
>                             convert(evalf(D(f)(x0), 2), string))],
> [-4.5, 4.2, cat("h = ", convert(evalf((1+n-k)*p/n, 2), string))],
>                           [-4.5, 3.9, cat("(f(x0+h) - f(x0))/h = ",
>    convert(evalf((f(x0 + (1+n-k)*p/n) - f(x0))/((1+n-k)*p/n), 2),
>                              string))]], align=RIGHT, color=BLUE);
>   texto2 := textplot([[x0,0,"|"],[x0+(1+n-k)*p/n,0,"|"]],
>                                             align=ABOVE, color=RED);
>   texto3 := textplot([[x0,0,"|"],[x0+(1+n-k)*p/n,0,"|"]],
>                                             align=BELOW, color=RED);
>   gr[k] := display(graf, texto1, texto2, texto3):
> end do:
> g2 := display(seq(gr[k], k=1..n), insequence=true):
>
> display(g1, g2);
```

Figura 2.22:



Capítulo 3

Animações tridimensionais

3.1 O comando `animate3d`

O comando `animate3d` pode ser usado para gerar animações tridimensionais. Seu uso é semelhante ao do comando `animate`:

`animate3d(`*F(x, y, t), x = a..b, y = c..d, t = e..f, opções*`)`

Exemplo 3.1 *O seguinte comando gera uma animação que inicia com o gráfico de* $\operatorname{sen}(x)\operatorname{sen}(y)$ *(que corresponde a* $t = 1$*) e termina com o gráfico de* $\operatorname{sen}(2x)\operatorname{sen}(2y)$ *(que corresponde a* $t = 2$*).*

```
> with(plots):
> animate3d( sin(x*t)*sin(y*t), x=-3..3, y=-3..3, t=1..2, frames=10,
>                                                  scaling=constrained);
```

Neste caso, é usado um total de 10 gráficos na animação.

Com o `animate3d`, é possível "deformar" o gráfico de uma superfície parametrizada por

$$X_1(u,v) = (f_1(u,v),\ g_1(u,v),\ h_1(u,v))$$

em outra superfície parametrizada por

$$X_2(u,v) = (f_2(u,v),\ g_2(u,v),\ h_2(u,v)).$$

Para isso, basta usar no `animate3d` uma função $F(u,v,t) = (1-t)X_1(u,v) + tX_2(u,v)$ $= ((1-t)f_1(u,v) + tf_2(u,v),\ (1-t)g_1(u,v) + tg_2(u,v),\ ((1-t)h_1(u,v) + th_2(u,v))$ com $0 \leq t \leq 1$.

Exemplo 3.2 *Para deformar* $(u, v, \operatorname{sen}(u^2+v^2))$ *em* $(u, v, 5-\cos(u))$ *com um total de 20 gráficos na animação e grade de tamanho* 30×30*, basta usar, depois do* `with(plots)`*, o seguinte comando:*

```
> animate3d([(1-t)*u+t*u, (1-t)*v+t*v, (1-t)*sin(u^2+v^2)+t*(5-cos(u))],
>                   u=-4..4, v=-4..4, t=0..1, frames=20, grid=[30,30]);
```

Com o `animate3d`, é possível girar na tela o gráfico de qualquer superfície parametrizada por $X(u,v) = (f(u,v),\ g(u,v),\ h(u,v)$. Para isso, basta fazer o t variar no intervalo $[0, 2\pi]$ e usar uma função $F(u,v,t)$ nos seguintes formatos:

- $[f(u,v), g(u,v)\cos t - h(u,v)\operatorname{sen} t, g(u,v)\operatorname{sen} t + h(u,v)\cos t]$, para girar o gráfico em torno do eixo Ox

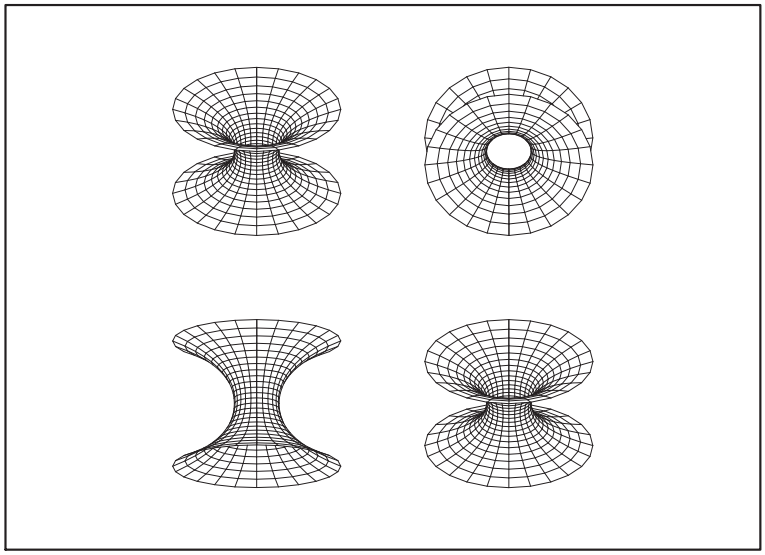
- $[f(u,v)\cos t - h(u,v)\operatorname{sen} t, g(u,v), f(u,v)\operatorname{sen} t + h(u,v)\cos t]$, para girar o gráfico em torno do eixo Oy
- $[f(u,v)\cos t - g(u,v)\operatorname{sen} t, f(u,v)\operatorname{sen} t + g(u,v)\cos t, h(u,v)]$, para girar o gráfico em torno do eixo Oz

Exemplo 3.3 *Vamos girar o gráfico de um* catenóide *em torno do eixo Oy:*

```
> f := (u, v) -> cosh(v)*cos(u);
> g := (u, v) -> cosh(v)*sin(u);
> h := (u, v) -> v;
> F := (u,v,t) ->
> [f(u,v)*cos(t)-h(u,v)*sin(t), g(u,v), f(u,v)*sin(t)+h(u,v)*cos(t)];
> teste:=plots[animate3d](F(u,v,t), u=-Pi..Pi, v=-2..2, t=0..2*Pi,
>     frames=4, orientation=[0,45],scaling=constrained,shading=none):
>
> display(teste);  # mostra todos os gráficos
> teste;           # executa a animação
```

A Figura 3.1 representa a animação. Ela foi construída com um `display(teste);`

Figura 3.1:



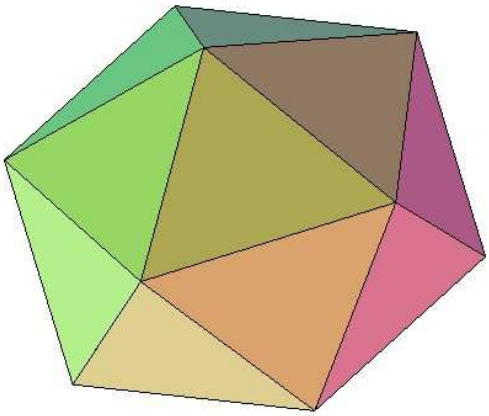
3.2 Sólidos

Para girar um sólido ou uma superfície tridimensional, podemos multiplicar por combinações de senos e cossenos, conforme exemplo anterior. Mas, isso pode ser feito mais facilmente se forem alterados os ângulos de orientação do gráfico.

Exemplo 3.4 *O pacote* `plottools` *traz vários sólidos pré-definidos tais como cubos, tetraedros, icosaedros etc. Neste exemplo, mostramos um icosaedro e alteramos sua orientação. Isso faz com que o sólido ocupe várias posições diferentes (Figura 3.2).*

```
> with(plottools):
> with(plots):
```

Figura 3.2:



```
> n := 50: for k from 1 to 50 do
>   gr[k] := display(icosahedron([0,0,0],0.8),
>                                  orientation=[360/n*k, 360/n*k]):
> end do:
>
> display(seq(gr[k], k=1..n), insequence=true);
```

No lugar de `icosahedron([0,0,0], 0.8 )` *poderíamos ter algo como* `octahedron([0,0,0], 0.8)` *ou* `hexahedron([1,1,1], 0.5)`.

Exemplo 3.5 *Construímos o gráfico de uma superfície que parece com uma mola e giramos o gráfico em torno do eixo vertical através da alteração do primeiro parâmetro de* `orientation`.

```
> with(plots):
> X := (u, v) -> [(11 + 2*cos(u))*cos(v), (11 + 2*cos(u))*sin(v),
>                                                    v + 2*sin(u)]:
> n := 50:
> for k from 0 to n do
>   g[k] := plot3d(X(u, v), u=0..2*Pi, v=-12..12, grid=[20, 70],
>                                    orientation=[ 360/n*k, 70]):
> end do:
> display(seq(g[k], k=0..n), insequence=true);
```

Podemos também trocar o `[360/n*k, 70]` *de orientation por* `[70, 360/n*k]` *ou por* `[360/n*k, 360/n*k]`. *Trocar o "11"por "v"para ver o que acontece.*

3.3 Superfícies de revolução

Ao girarmos o gráfico de uma curva em torno do eixo dos z, obtemos uma superfície de revolução que pode ser parametrizada por $X(u, v) = (f(u)\cos v, f(u)\operatorname{sen} v, g(u))$.

Exemplo 3.6 *Giramos uma hipérbole (parametrizada por* $f(t) = 3\cosh t, g(t) = 3\operatorname{senh} t$*) em torno do eixo z e obtemos um hiperbolóide de uma folha. O gráfico do hiperbolóide é mostrado apenas no final. Veja Figura 3.4.*

Figura 3.3:

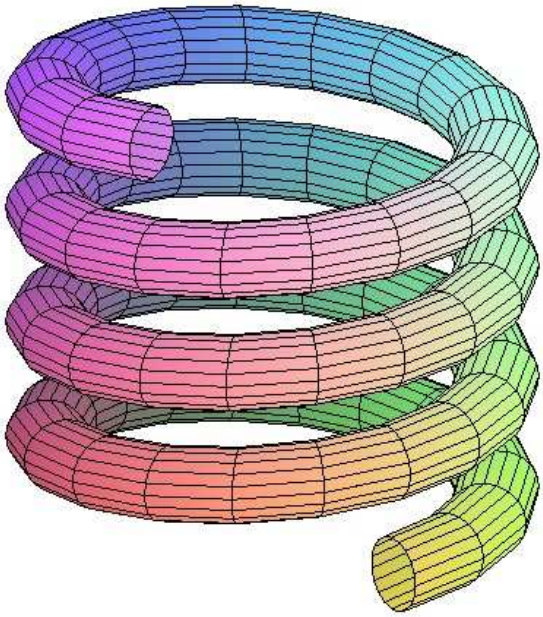
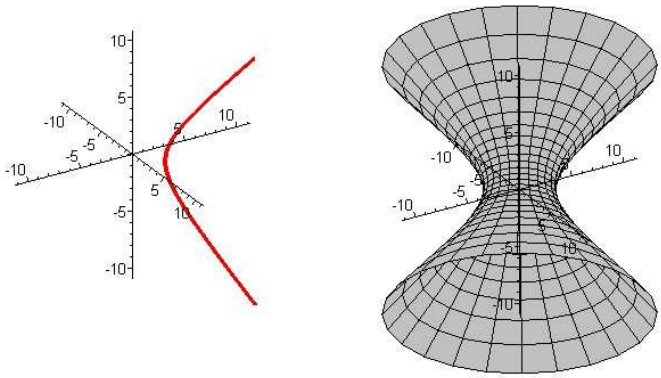


Figura 3.4:




```
> with(plots):
> f := x -> 3*cosh(x): g := x -> 3*sinh(x):
> a := -2: b := 2: c := 0: d := 2*Pi: n := 50:
>
> X := (u, v) -> [ f(u)*cos(v), f(u)*sin(v), g(u) ]:
> g2 := plot3d(X(u, v), u=a..b, v=c..d, color=GRAY): #shading=zhue):
> for k from 1 to n do
> gr[k] := spacecurve(X(t, k*2*Pi/n), t=a..b, thickness=4,
>                          axes=normal, color=red): #shading=zhue);
> end do:
> # display([seq(gr[k], k=1..n), g2], insequence=true);
> g1 := display(seq(gr[k], k=1..n), insequence=true):
> display(g1, g2, insequence=true);
```

```
>
> display(g1, g2);
```

Repetimos a construção do hiperbolóide, mas, neste caso, ele vai sendo construído a medida que a hipérbole vai girando.

```
> with(plots):
> f := x -> 3*cosh(x): g := x -> 3*sinh(x):
> X := (u, v) -> [ f(u)*cos(v), f(u)*sin(v), g(u) ]:
>
> n := 50:
> for k from 1 to n do
>   gr1 := spacecurve(X(t, k*2*Pi/n), t=a..b,
>                          thickness=4,axes=normal, color=red):
>   gr2 := plot3d(X(u, v), u=a..b, v=0..k*2*Pi/n, color=GRAY):
>   gr[k] := display(gr2, gr1):
> end do:
>
> display(seq(gr[k], k=1..n), insequence=true);
```

Outras opções de construção são as seguintes:

- Cilindro: `f := x -> 3: g := x -> x: a:=-2: b:=2:`
- Cone: `f := x -> x: g := x -> x: a:=-2: b:=2:`
- Esfera: `f := x -> cos(x): g := x -> sin(x): a:=-Pi/2: b:=Pi/2:`
- Parabolóide: `f := x -> sqrt(x): g := x -> x: a:=0: b:=4:`
- Esfera: `f := x -> sqrt(25-x^2): g := x -> x: a:=-5: b:=5:`
- Hiperbolóide de 1 folha: `f := x->sqrt(1+x^2): g := x->x: a:=-3: b:=3:`

3.4 Superfícies geradas pelo deslocamento de curvas

Usando a mesma parametrização das superfícies de revolução, podemos obter novos tipos de animação apenas trocando a ordem da variação dos parâmetros.

Exemplo 3.7 *Neste exemplo, construímos um cilindro através do deslocamento na direção vertical de uma circunferência.*

```
> with(plots):
>
> f := x -> 3: g := x -> x:  # cilindro
> a := -3: b := 3: c := 0: d := 2*Pi: n := 30:

> X := (u, v) -> [ f(u)*cos(v), f(u)*sin(v), g(u) ]:
> g2 := plot3d(X(u, v), u=a..b, v=c..d, shading=zhue):
> for k from 1 to n do
```

```
>   gr[k] := spacecurve(X(a+k*(b-a)/n, t), t=c..d, thickness=4,
>                         axes=normal, color=black):
> end do:
> g1 := display(seq(gr[k], k=1..n), insequence=true):
> display(g1, g2, insequence=true);
```

O trecho `for k ...` *acima pode ser trocado por:*

```
> for k from 1 to n do
>   gr1 := spacecurve(X(a+k*(b-a)/n, t), t=c..d, thickness=4,
>                                           axes=none, color=black):
>   gr2 := plot3d(X(u, v), u=a..a+k*(b-a)/n, v=c..d, shading=zhue):
>   gr[k] := display(gr2, gr1):
> end do:
>
> display(seq(gr[k], k=1..n), insequence=true);
```

Exemplo 3.8 *Aqui, um gráfico é deslocado no espaço gerando uma superfície X. Veja Figura 3.5.*

```
> with(plots):
> f := x -> x: g := x -> sin(2*x) + sin(x):
> a := 0: b := 2*Pi: c := -2: d := 2: n := 100:
>
> X := (u, v) -> [f(u), g(u), v]:
>
> g2 := plot3d(X(u, v), u=a..b, v=c..d, shading=zhue):
>
> for k from 1 to n do
>   gr[k] := spacecurve(X(a+k*(b-a)/n, t), t=c..d, thickness=4,
>                         axes=normal, color=black):
> end do:
>
> g1 := display(seq(gr[k], k=1..n), insequence=true):
>
> display(g1, g2, insequence=true);
>
>
> for k from 1 to n do
>   gr1 := spacecurve(X(a+k*(b-a)/n, t), t=c..d, thickness=4,
>                                           axes=none, color=black):
>   gr2 := plot3d(X(u, v), u=a..a+k*(b-a)/n, v=c..d, shading=zhue):
>   gr[k] := display(gr2, gr1):
> end do:
>
> display(seq(gr[k], k=1..n), insequence=true);
```

Exemplo 3.9 *Neste exemplo, uma reta se desloca para cima e girando em torno do eixo vertical. A superfície obtida é chamada helicóide (Figura 3.6) .*

Figura 3.5:

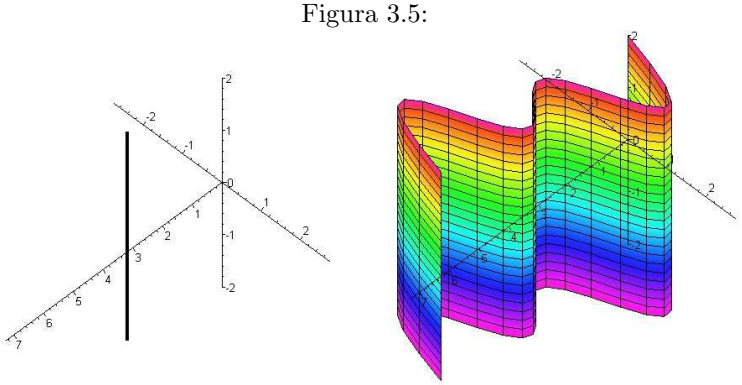
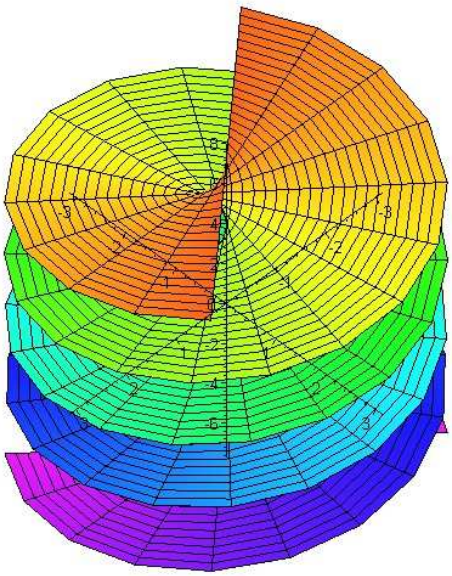


Figura 3.6:

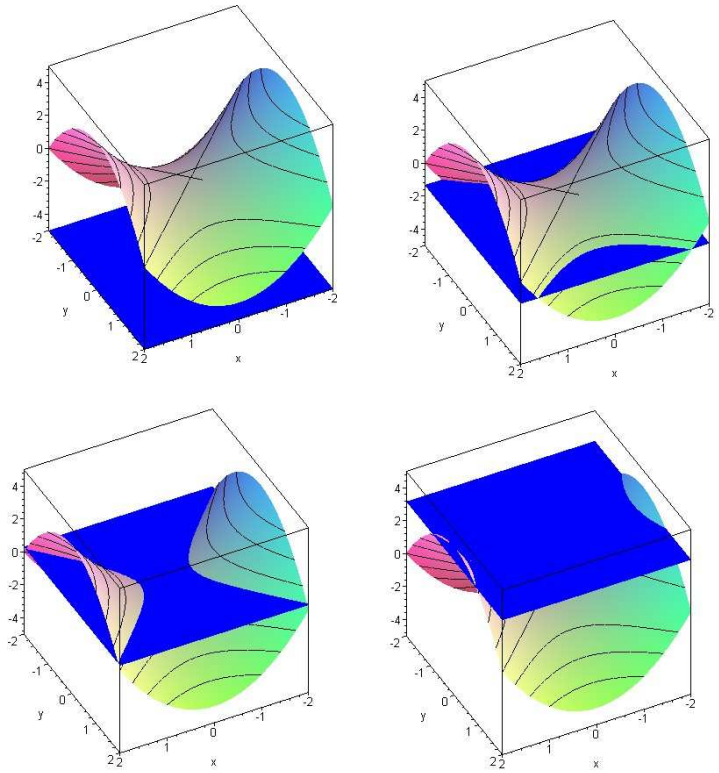


```
> with(plots):
> f := x -> x: g := x -> x:
> a := -Pi: b := Pi: c := -7: d := 7: n := 50:
>
> X := (u, v) -> [f(u)*cos(v), f(u)*sin(v), g(v)]:
>
> for k from 1 to n do
>   gr1 := spacecurve(X(t,c+k*(d-c)/n), t=a..b, thickness=4,
>                                          axes=none, color=black):
>   gr2 := plot3d(X(u, v), u=a..b, v=c..c+k*(d-c)/n,
>                                    shading=zhue, grid=[50,50]):
>   gr[k] := display(gr2, gr1):
> end do:
>
> display(seq(gr[k], k=1..n), insequence=true);
```

Exemplo 3.10 *Um plano se desloca para cima, depois para baixo, e vai interceptando*

um gráfico tridimensional, deixando perceber quais as curvas de nível da superfície. Veja Figura 3.7.

Figura 3.7:



```
> with(plots):
> f := (x, y) -> x^2 - y^2:
>
> a := -2: b := 2: c := -2: d := 2: p := -5: q := 5: n := 50:
> anima1 := plot3d(f(x, y), x=a..b, y=c..d, axes=boxed,
>                                              style=patchcontour):
> anima2 := animate3d([x, y, z], x=a..b, y=c..d, z=p..q,
>                          frames=n, style=patchnogrid, color=BLUE):

> anima3 := animate3d([x, y, p+q-z], x=a..b, y=c..d, z=p..q,
>                            frames=n,style=patchnogrid,color=BLUE):
> teste := display(anima2, anima3, insequence=true):
> display(anima1, teste);
```

Outras opções para f: `f := (x, y) -> 10*exp((-x^2 - y^2)/4)` *ou* `f := (x, y) -> cos(x^2 + y^2)`.

Referências Bibliográficas

[1] Blyth, B. M. (2004) *Animations using Maple*, 17th. Annual International Conference on Technology in Collegiate Mathematics.

[2] Corless, R. M. (2001) *Essential Maple 7*, University of Western Ontario, disponível em www.mapleapps.com.

[3] Harris, K. (1992) *Discovering Calculus with Maple*, John Wiley & Sons.

[4] Heath, D. *Maple Animations for Teaching Mathematics*, Pacific Lutheran University, disponível em www.calculus.org/Heath/maple_anims.html.

[5] Heck, A. (1996) *Introduction to Maple*, Second Edition, Springer.

[6] Irby, V. *Reflection and Transmission*, University of South Alabama, disponível em www.southalabama.edu/physics/animation/reflect21.html.

[7] Portugal, R. (2002) *Introdução ao Maple*, Centro Brasileiro de Pesquisas Físicas, disponível em www.cbpf.br/∼portugal/

[8] Waterloo Maple Inc. (2001) *Maple 7 Learning Guide*.

[9] Waterloo Maple Inc. (2001) *Maple 7 Programming Guide*.

Lenimar Nunes de Andrade é Bacharel em Matemática pela Universidade Federal da Paraíba (1982), Mestre em Matemática pela Universidade Federal de Pernambuco (1987), Doutor em Engenharia Elétrica pela Universidade Estadual de Campinas (1998), professor da Universidade Federal da Paraíba desde março de 1984. e autor do livro *"Introdução à Computação Algébrica com o Maple"*, lançado pela Sociedade Brasileira de Matemática em julho de 2004.