O Módulo de Cálculo (Kernel)

■ Avaliação de Expressões

O Kernel do *Mathematica* é o responsável por todos os cálculos, simbólicos ou numéricos, necessários para reponder a um pedido de avaliação do FrontEnd. Por exemplo, se escrevermos uma expressão (**Sin**, **Arc Cos** e **TrigExpand** são nomes de funções nativas do *Mathematica)*,

```
TrigExpand[Sin[x – ArcCos[y]]]
```

dentro de uma célula de **Input**, e avaliarmos esta expressão, todo o trabalho intermédio que envolve a interpretação da célula bem como a consulta de bibliotecas trigonométricas com informação das propriedades das funções nativas envolvidas para executar o pedido é feito pelo **Kernel**, que no fim devolve para o **FrontEnd** o resultado para ser formatado como **Output** numa nova célula:

$$-\sqrt{1-y^2}\ \text{Cos}[x] + y\ \text{Sin}[x]$$

Todas as células **Input** e **Output** visíveis num Notebook podem ser guardadas com o Notebook onde estão escritas. Contudo, os cálculos do **Kernel** e as células que forem apagadas ou reescritas perdem–se cada vez que se desliga o **Kernel**. Por outro lado, numa mesma sessão do Kernel é possível recuperar células de **Input** e **Output** que tenham sido apagados ou alterados a um dado ponto. De facto em cada sessão do **Kernel** existe um contador **$Line** para o número de vezes que este é chamado pelo **FrontEnd** para calcular uma expressão.

```
$Line
```

```
2
```

Pode–se utilizá–lo para reeditar ou reavaliar uma expressão préviamente submetida, por ex **InString[$Line – 2]**.

```
NotebookWrite[SelectedNotebook[], Cell[BoxData[ToExpression[InString[1]]], "Input"]]
```

```
TrigExpand[Sin[x – ArcCos[y]]]
```

□ Primeira avaliação

Para começar a comunicação entre o Kernel e o FrontEnd, basta avaliar uma célula de **Input**. Isto é feito usando a combinação SHIFT RET ou Kernel ▸ Evaluation ▸ Evaluate Cells com o cursor dentro de uma célula de Input ou tendo seleccionado várias células deste tipo.
Se não quiser avaliar nada pode–se começar um Kernel na máquina local usando o seguinte comando do menu Kernel ▸ Start Kernel ▸ LocalKernel.
As avaliações seguintes serão feitas com a mesma combinação SHIFT RET ou com a tecla ENTER do teclado numérico.

Células de Inicialização

Ao iniciar a avaliação de expressões num Notebook o Kernel verifica se existem células especiais designadas de **Initialization Cells** em qualquer parte do Notebook. Estas células são células normais de **Input** com a propriedade **InitializationCell → True**, o que geralmente indica que são definições prévias necessárias para o bom funcionamento de outras funções definidas no Notebook. As células de initialização são distinguidas por terem um pequeno I adicionado ao delimitador da célula, e podem ser marcadas/desmarcadas assim seleccionando–as e usando o menu Cell ▸ Cell Properties ▸ Initialization Cell.

```
<< Utilities`Notation`
<< Graphics`FilledPlot`
```

Figura 9

□ CICLO DE AVALIAÇÃO

(a) – A variável **$Pre** é avaliada antes de cada novo **Input**.

(b) – A expressão a avaliar é verificada para detectar erros de sintaxe.

(c) – Se passar, a expressão resultante é avaliada pelo **Kernel**.

(d) – O resultado é processado adicionalmente se a variável **$Post** estiver definida.

(e) – A expressão resultante é escrita no **FrontEnd** como células de **Output**.

```
?? $Pre
```

$Pre is a global variable whose value, if set, is applied to every input expression. More…

```
$Pre := (Hello[#] &)
```

```
"isto é uma expressão"
```

```
Hello[isto é uma expressão]
```

```
?? $Post
```

$Post is a global variable whose value, if set, is applied to every output expression. More…

```
$Post := (Bye[#] &)
```

```
Bye[Hello[Null]]
```

```
"isto é outra expressão"
```

```
Bye[Hello[isto é outra expressão]]
```

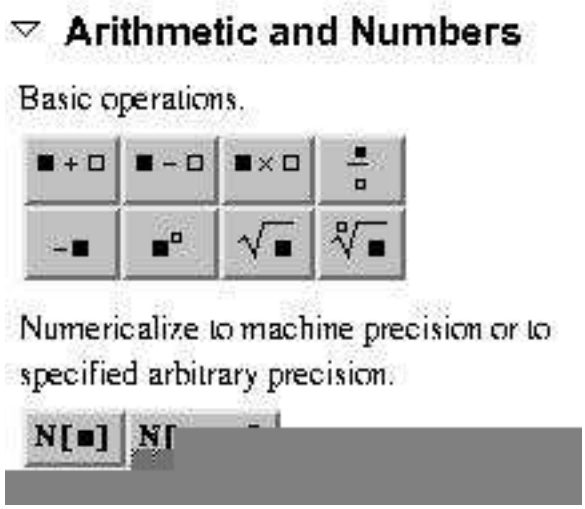
```
Clear[$Pre, $Post]
```

```
Hello[Null]
```

■ Paleta - Cálculos e Funções Básicas do Mathematica

Muitas das expressões matemáticas usadas durante uma sessão no ***Mathematica,*** sejam algébricas ou gráficas, podem ser introduzidas a partir da paleta intitulada **BasicCalculations**, que pode ser aberta com a sequência de comandos de menu File ▸ Palettes ▸ BasicCalculations . De início é aliás um bom auxiliar para garantir a correcta sintaxe e utilização dos comandos.

Plus[a, b]	$a + b$
Subtract[a, b]	$a - b$
Times[a, b]	$a \times b$
Divide[a, b]	$\frac{a}{b}$
Minus[a]	$-a$
Power[a, b]	a^b
Sqrt[a]	$\sqrt{a}$
Power[a, Power[b, −1]]	$\sqrt[b]{a}$
N[π]	3.14159
N[π, 8]	3.1415927



Note que **Subtract[a, b]** é sempre convertido em **Plus[a, −b]** Além disso a operação Plus tem atributo Orderless, o que significa também que **b − a = Plus[b, −a]** é sempre convertido em **−a + b = Plus**[−a, b] devido à ordem lexicográfica dos símbolos.

Times[a, b], **a × b** e **a ∗ b** são sempres convertidos em **a b** onde o espaço entre **a** e **b** também representa o produto de **a** e **b**. A operação de produto **Times** tem igualmente o atributo Orderless, o que significa que é comutativo e ordena automáticamente os multiplicandos pela ordem lexicográfica, i.e. z X a 4 d → 4 a d X.

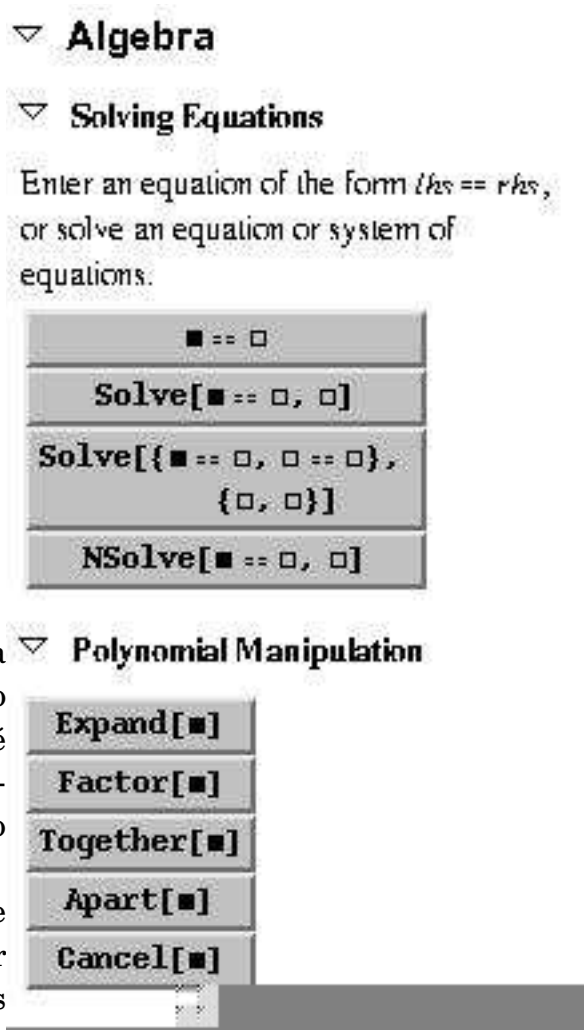
A operação **Divide**[a, b] de facto não existe de 'per se', e reduz−se a Times[a, Power[b, −1]]. Esta diferença é importante de notar porque a expressão $\sqrt{\text{expr}}$ não aparece em $\frac{1}{\sqrt{\text{expr}}}$ embora pareça existir no denominador. De facto $\sqrt{\text{expr}}$ ≡ Power[expr, Rational[1, 2]] e $\frac{1}{\sqrt{\text{expr}}}$ ≡ Power[expr, Rational[−1, 2]].

4 d

4 d

O Mathematica não faz hipóteses sobre a natureza dos símbolos que encontra. Assim $\sqrt{-a}$ não é necessáriamente um imaginário, nem $\sqrt{(1+a)^2}$ se reduz a 1 + a.

Equal[a, b]	a == b
Solve[a x – c == b, x]	$\{\{x \to \frac{b+c}{a}\}\}$
Solve[{x – y == a, x + y == b}, {x, y}]	$\{\{x \to \frac{a+b}{2}, y \to \frac{-a+b}{2}\}\}$
NSolve$\left[x^{2/3} + 2x == 2, x\right]$	{{x → 0.631837}}
Expand$\left[(a+b)^2\right]$	$a^2 + 2ab + b^2$
Factor$\left[x a^2 + x^2 a\right]$	a x (a + x)
Together$\left[\frac{a}{b} + \frac{c}{d}\right]$	$\frac{bc+ad}{bd}$
Apart$\left[\frac{bc+ad}{bd}\right]$	$\frac{a}{b} + \frac{c}{d}$
Cancel$\left[\frac{bd+ad}{bd}\right]$	$\frac{a+b}{b}$



A igualdade lógica é representada pelo símbolo == obtido a partir de dois caracteres = seguidos. Um só = é reservado para atribuição de valores a símbolos (Set). Assim x == a+y é uma equação, enquanto x = a+y não o é porque é imediatamente substituído pelo valor a+y associado doravante ao símbolo x.

O resultado de Solve [eqns,vars] é sempre uma lista de regras de substituição que podem ser usadas para avaliar expressões que incluam as variáveis vars em relação às quais foram resolvidas as equações eqns.

Simplify$[4\,a^2 - 12\,c\,a + 9\,c^2]$	$(2\,a - 3\,c)^2$
FullSimplify[x(1 + x) Gamma[x]]	Gamma[2 + x]
Simplify$\left[\frac{d+\sqrt{4\,a^2-8\,a\,b+4\,b^2}}{a-b},\ a > b\right]$	$\frac{2\,a-2\,b+d}{a-b}$
FullSimplify$\left[\frac{d+\sqrt{4\,a^2-8\,a\,b+4\,b^2}}{a-b},\ a > b\right]$	$2 + \frac{d}{a-b}$
Simplify$\left[\sqrt{(x+y)^2},\ \{x, y\} \in \text{Reals}\right]$	Abs[x + y]
Simplify$\left[\sqrt{(x+y)^2},\ x > 0 \wedge y > 0\right]$	x + y
FunctionExpand$\left[(2\,x - x^2)^a\right]$	$(2 - x)^a\,x^a$

Simplification

Simplify tries various manipulations to make expressions smaller. FullSimplify tries more manipulations, but takes longer.

Simplify[■]
Simplify[■, □]
FullSimplify[■]
FullSimplify[■, □]
FunctionExpand[■]
FunctionExpand[■, □]

The two-argument forms of the simplification functions take assumptions in the second argument, which can be formed from the items in the following palette.

■ > □ | ■ < □
■ ≥ □ | ■ ≤ □
■ == □ | ■ ∧ □
■ ∈ Complexes
■ ∈ Reals
■ ∈ Algebraics
■ ∈ Rationals
■ ∈ Integers
■ ∈ Primes
■ ∈ Booleans

Os operadores de simplificação de expressões algébricas podem incluir um segundo argumento com condições a ter em conta quando se simplifica uma expressão. Em geral estas condições são expressões booleanas (True ou False) envolvendo símbolos que aparecem na expressão a simplificar.

$1 + \sqrt{3} \in$ Algebraics	True
$\pi \in$ Algebraics	False
$1 + i \in$ Complexes	True
$1 + i\,a \in$ Complexes	$1 + i\,a \in$ Complexes
$\frac{4}{3} \in$ Rationals	True
$\frac{1.5}{3} \in$ Reals	True
$\frac{1.5}{3} \in$ Rationals	False
$\frac{15}{30} \in$ Rationals	True
$5 \in$ Integers	True
$5. \in$ Integers	False

■ Uso de Parêntesis em Mathematica

(*termo*)	*Curvos para agrupar termos dentro de expressões...*
(* *frase* *)	*Curvos com * para declarações e comentários ...*
{ *a1, a2, a3* }	*Chavetas só para Listas, Matrizes ou Conjuntos!*
f[*args*]	*Rectos (1 vez) para argumentos de funções ...*
expr[[*índice*]]	*Rectos (2 vezes) para índices localizando partes de ' expr '.*

Os Parêntesis em *Mathematica* têm usos diferentes que em C. Os parêntesis curvos já não são usados para delimitar os argumentos de funções **f(args)**, papel desempenhado sempre por parêntesis rectos **f[args]**.

```
f (a, b)
```

— *Syntax::sntxf : "(" cannot be followed by "a, b)".*

— *f (a, b)*

```
f[a, b]
```

```
f[a, b]
```

Os parêntesis curvos servem mais para delimitar termos dentro de expressões. O esquecimento de parêntesis curvos pode ter sérias consequências ... Por exemplo, escrevendo:

```
x^2/3
```

$$\frac{x^2}{3}$$

porque a operação Power (^) tem maior precedência que a de divisão. Já escrevendo :

```
x^(2/3)
```

$$x^{2/3}$$

Escrevendo em notação bidimensional alguns parêntesis deixam de ser necessários.

```
x^(2/3)
```

$$x^{2/3}$$

Em de células de *Input* podem-se incluir comentários dentro da combinação de símbolos **(* ... *)**, muito à semelhança do C com a sua forma /*...*/. Estas expressões não são avaliadas pelo *Kernel*.

As chavetas **{}** têm também uma função específica, que é a de representar objectos do tipo lista, como conjuntos, vectores ou matrizes, por ex. $\{a_1, a_2, \ldots, a_n\} \equiv \text{List}[a_1, a_2, \ldots, a_n]$

```
x = {a1, a2, a3}; (* x é um vector com três componentes *)
```

```
norm = √(x.x)          (* O produto escalar é Dot[x,x]≡x.x *)
```

$$\sqrt{a_1^2 + a_2^2 + a_3^2}$$

A função **FullForm** mostra a representação interna de cada expressão do *Mathematica*.

```
FullForm[x]
```

```
List[Subscript[a, 1], Subscript[a, 2], Subscript[a, 3]]
```

A função List[] é linear a associativa. É também distributiva para certas operações mas não para outras.

```
α {a1, a2} + β {b1, b2}
```

$$\{\alpha\, a_1 + \beta\, b_1,\ \alpha\, a_2 + \beta\, b_2\}$$

```
{a1, a2} / {b1, b2}
```

$$\left\{\frac{a_1}{b_1},\ \frac{a_2}{b_2}\right\}$$

```
{a1, a2}^{b1, b2}
```

$$\left\{a_1^{b_1},\ a_2^{b_2}\right\}$$

```
Log[{a1, a2}]
```

```
{Log[a1], Log[a2]}
```

```
Equal[{a1, a2}, {b1, b2}]
```

```
{a1, a2} == {b1, b2}
```

Qualquer expressão **expr** em Mathematica é um aglomerado de símbolos e operações. As suas partes componentes podem ser manipuladas usando a função nativa **Part[expr, pos_List]** ou abrevidamente **expr[[pos_list]]** .

```
FullForm[Sin[x + z]]
```

```
Sin[Plus[x, z]]
```

A estrutura seguinte mostra explicítamente as posições dos diversos elementos que compõem a expressão Sin[x + z]

$$\underbrace{\text{Sin}}_{0}\left[\underbrace{\underbrace{x}_{1,1} + \underbrace{z}_{1,2}}_{1}\right]$$

O índice 0 representa a cabeça da expressão, enquanto que o 1 representa o primeiro (e único neste caso) argumento da cabeça da expressão. Este primeiro argumento Plus[x, z] ≡ x + z por sua vez tem dois argumentos 1, 1 e 1, 2 que são x e z. Ou seja:

```
Sin[x + z][[0]]    →  Sin
Sin[x + z][[1]]    →  x + z
Sin[x + z][[1, 1]] →  x
Sin[x + z][[1, 2]] →  z
```

A parte '0' de qualquer 'expr' é a 'cabeça' (**Head[expr]**) da expressão.

```
x = {a1, a2, a3};
```

```
{x[[0]], Head[x]}
```

```
{List, List}
```

Sendo x = {a_1, a_2, a_3} uma lista simbólica, as operações de soma e Sin distribuem−se pelos seus elementos:

```
x + z
```

```
{z + a1, z + a2, z + a3}
```

```
Sin[x + z]
```

```
{Sin[z + a1], Sin[z + a2], Sin[z + a3]}
```

```
{Sin[x + z][[0]], Head[Sin[x + z]]}
```

```
{List, List}
```

Se a função **f[a, b]** estiver definida, então **f[a, b][[0]] = Head[f[a, b]]** é a cabeça da expressão que usámos para definir **f[x_, y_]**.
A cabeça de **f[a,b] = a^2 − b** é assim **f[a, b][[0]] ≡ Head[f[a, b]] = Plus**.

Nunca confundir a função **f[x]** com **f[[x]]**: esta última notação representa a função **Part[f, x]**, onde **f** pode ser qualquer expressão, e x designa um conjunto de índices inteiros especificando partes de **f**.

```
FullForm[f[[z]]]
```

— Part::pspec : Part specification z is neither an integer nor a list of integers.

```
Part[f, z]
```

```
x[[1]]                (* Part[x,i] ≡ x[[i]] *)
```

```
a1
```

```
x[[{1, 3}]]          (* Partes designadas por conjuntos de posições *)
```

```
{a1, a3}
```

```
M = {x, x /. a → b}          (* Matrizes são Listas de Listas (dim=2) *)
```

```
{{a1, a2, a3}, {b1, b2, b3}}
```

```
MatrixForm[M]
```

$$\begin{pmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{pmatrix}$$

```
M[[0]]
M[[1]]
M[[1, 2]]
```

```
List
```

```
{a1, a2, a3}
```

```
a2
```

```
M.{c1, c2, c3} // MatrixForm
```

$$\begin{pmatrix} a_1\, c_1 + a_2\, c_2 + a_3\, c_3 \\ b_1\, c_1 + b_2\, c_2 + b_3\, c_3 \end{pmatrix}$$

```
{c1, c2, c3}.M
```

— *Dot::dotsh : Tensors* $\{c_1, c_2, c_3\}$ *and* $\{\{a_1, a_2, a_3\}, \{b_1, b_2, b_3\}\}$ *have incompatible shapes.*

```
{c1, c2, c3}.{{a1, a2, a3}, {b1, b2, b3}}
```

Computação Simbólica

■ Atribuição e Substituição

Alias	Função	Acção
x = expr	Set[x, expr]	define o símbolo x como sendo expr, o qual pode ser um número ou uma expressão simbólica contendo outras variáveis
x := expr	SetDelayed[x, expr]	define x como a expressão literal de expr só avaliando expr quando x é usado
x =.	Unset[x]	apaga qualquer definição associada ao símbolo x.
	Clear[x]	faz o mesmo que Unset[x] mas deixa Atributos de x
	Remove[x]	Remove x da lista de símbolos Global.
expr /. {x → val_x, ...}	ReplaceAll[expr, {x → val_x, ...}]	substitui em sequência as ocorrências de x, ... em expr por val_x, ... imediatamente.
expr //. {x → val_x, ...}	ReplaceRepeated[expr, {x → val_x, ...}]	substitui repetidamente as ocorrências de x, ... em expr por val_x, ... até expr não mudar mais.
expr /. {x :→ val_x, ...}	ReplaceAll[expr, {x → val_x, ...}]	substitui as ocorrências de x, ... em expr por val_x, ... mas só quando expr é avaliada

Tabela 6

A atribuição de ´valores´ simbólicos ou numéricos (reais ou complexos) bem como a substituição numa dada expressão de símbolos, sub– expressões ou padrões por valores simbólicos diferentes é uma das características do processamento do Mathematica. Note–se que enquanto uma atribui–ção fica registada em memória durante uma sessão de trabalho do Kernel, uma substituição só é válida para a avaliação da expressão a que está aplicada.

□ Atribuição Imediata de uma expressão a um símbolo

x = expr ou *Set[x, expr]*

A atribuição de um nome **x** a uma expressão (simbólica ou numérica) **expr** é realizado pela fun–ção nativa **Set[x, expr]** ou abreviadamente **x = expr**. Esta atribuição é uma associação **Ime–diata** do símbolo **x** à expressão **expr**, sendo que todos os símbolos com valores associados que entram em **expr** são substituídos pelos seus valores antes de se fazer a atribuição. Depois da atribuição imediata ao símbolo **x**, mudanças nos valores dos símbolos em **expr** apenas afectam **x** se à altura da atribuição esses símbolos não tiverem um valor associado. A operação inversa desta atribuição é óbviamente **Unset[x]** que limpa qualquer associação de valores ou expressões com o símbolo **x**.

a passa a ser um símbolo puro.

```
a =.
```

b fica associado com π /2

```
b = π/2;
```

b é substituído pelo seu valor associado antes da atribuição de x.

```
x = Tan[a^2 - b];
FullDefinition[x]
```

```
x = -Cot[a^2]
```

se atribuir um valor a a isso reflecte−se na avaliação final de x

```
a = π;
x
```

```
-Cot[π^2]
```

A definição de x inclui agora o valor associado de a.

```
FullDefinition[x]
```

```
x = -Cot[a^2]

a = π
```

Como b já não entra na definição de x, alterá−lo agora não afecta x.

```
b = π/4;
x
```

```
-Cot[π^2]
```

```
FullDefinition[x]
```

```
x = -Cot[a^2]

a = π
```

□ Atribuição Retardada de uma expressão a um símbolo

x := expr ou *SetDelayed[x , expr]*

SetDelayed[x, expr] ou **x := expr** é a forma de fazer uma atribuição **Retardada**, i.e. uma atribuição a **x** duma expressão **expr** cujos símbolos só são avaliados quando se invoca **x**. Isto permite manter a forma da expressão independente das definições dos seus símbolos à altura da atribuição. Por outro lado, de cada vez que **x** é invocado, **expr** é processada de novo, mesmo que não tenha havido qualquer mudança nos valores dos seus símbolos, o que se torna oneroso especialmente em ciclos repetitivos.

```
a =.
b = π/2;
x := Tan[a^2 - b];

FullDefinition[x]
```

```
x := Tan[a^2 - b]

b = π/2
```

Agora x é sensível a mudanças nas associações de a e de b.

```
a = π;
b = π/4;
x
```

```
-Tan[π/4 - π^2]
```

As novas associações de a e b fazem agora parte da definição de x

```
FullDefinition[x]
```

```
x := Tan[a^2 - b]

a = π

b = π/4
```

Limpa todas as associações aos símbolos x, a e b.

```
Clear[x, a, b]
```

□ Substituição Imediata de um símbolo ou modelo de expressão

expr /. *x* → *val*$_x$ ou *ReplaceAll*[*expr*, *x* → *val*$_x$]

Para alterar o valor de um símbolo ou sub–expressão **x** numa expressão **expr** existe a função **ReplaceAll[expr, x → valor]** ou **expr /. x→valor**. Note que o valor de **x** não é alterado, apenas a sua ocorrência em **expr** o é! Esta substituição é imediata, mas pára no primeiro nível da expressão. Para obter uma substituição de **x** a todos os níveis de **expr** use **ReplaceAll**[expr, x → valor, Infinity].

O símbolo a é substituído por y

```
Tan[a^2] /. a → y
```

```
Tan[y^2]
```

O símbolo a não foi permanentemente alterado.

```
a
```

```
a
```

Se y for uma expressão, também é imediatamente substituída.

```
y = (1 + b)^2;

Tan[a^2] /. a → y
```

```
Tan[(1 + b)^4]
```

A expressão (1+b) é substituída por ζ no resultado anterior.

```
% /. (1 + b) → ζ
```

```
Tan[ζ^4]
```

Note a precedência de **ReplaceAll** em relação a **Set**, e a sua alteração usando parêntesis. No caso de **v** a atribuição é feita antes da substituição.

```
u = {x1, x2} /. x → z
(v = {x1, x2}) /. x → z
```

```
{z1, z2}
```

```
{z1, z2}
```

```
Print["u=", u, "\n", "v=", v]
```

```
u={z1, z2}
v={x1, x2}
```

Para além de símbolos e expressões, **Rule[x, valor]** ou **RuleDelayed[x, valor]** aceitam modelos para **x**. Estes modelos são expressões envolvendo o símbolo _ ou **Blank[]**, que representa qualquer expressão do *Mathematica,* na forma **z_** (modelo designado **z**) ou **z_h** (modelo designado **z** para expressões de cabeça **h**). Por exemplo $x_{i\_}$ representa o modelo para expressões do tipo **Subscript[x, ...]**, onde o índice pode ser qualquer expressão.

No seguinte exemplo **v** é a lista **{x_1, x_2}**, **a** tem o valor $\frac{\pi}{3}$. Na última linha o molde $\mathbf{x_{i\_}}$ ajusta-se a $\mathbf{x_1}$ com **i = 1** e a $\mathbf{x_2}$ com **i = 2**, pelo que estes são substituídos em **v** pelas respectivas expressões $\mathbf{Sin\left[i + \frac{\pi}{3}\right]}$, onde a foi préviamente substituído pelo seu valor.

```
v = {x1, x2};
a = π/3;
v /. x_i_ → Sin[i + a]
```

```
{Sin[1 + π/3], Sin[2 + π/3]}
```

□ Substituição Retardada de um símbolo ou modelo de expressão

expr /. x :⇢ val$_x$ ou *ReplaceAll[expr, x :⇢ val$_x$]*

É possível fazer atribuições de **valor** a símbolos ou expressões **x** e que se efectivam apenas durante a avaliação duma expressão de *Input*. **Rule[x, valor]** ou **x → valor** indica que **x** é o mesmo que **valor** para efeitos da avaliação da expressão de *Input* em que aparece apenas. (Use −> ou ESC − > ESC para produzir →) Nesta atribuição, **valor** é avaliado antes de se substituir a **x**.

```
a = π / 3;
Rule[x, Sin[ξ + a]]
```

$$x \to \text{Sin}\left[\frac{\pi}{3} + \xi\right]$$

A atribuição eventual retartada **RuleDelayed[x, valor]** ou **x :⇢ valor** só difere da anterior no facto de **valor** ser avaliado apenas depois de se substituir a **x**. (Use :> ou ESC :> ESC para produzir :⇢)

```
a = π / 3;
RuleDelayed[x, Sin[ξ + a]]
```

$$x :\to \text{Sin}[\xi + a]$$

Na expressão seguinte, **HoldForm[expr]** mantém **expr** sem a avaliar. A substituição eventual de símbolos em **expr** é feita de acordo com as regras, (como se pode ver ligando o *Tracer* do ciclo de avaliação com o comando **On[]**), dando resultados diferentes conforme a substituição é imediata ou retardada.

```
u = {z1, z2};
```

```
On[ReplaceAll]
```

```
HoldForm[x^b] /. x → u[[1]]
```

— *ReplaceAll::trace : x^b /. $x \to u[[1]]$ −−> x^b /. $x \to z_1$.*

— *ReplaceAll::trace : x^b /. $x \to z_1$ −−> z_1^b.*

$$z_1^b$$

```
HoldForm[x^b] /. x :⇢ u[[1]]
```

— *ReplaceAll::trace : x^b /. $x :\to u[[1]]$ −−> x^b /. $x :\to u[[1]]$.*

— *ReplaceAll::trace : x^b /. $x :\to u[[1]]$ −−> $u[[1]]^b$.*

$$u[[1]]^b$$

```
Off[]
```

□ Substituição Iterativa de um símbolo ou modelo de expressão

expr //. {*x* → *val*$_x$, *y* → *val*$_y$, …} ou *ReplaceRepeated*[*expr*, {*x* → *val*$_x$, *y* → *val*$_y$, …}]

Numa lista de regras de substituição pode existir referência a um símbolo substituível que aparece noutro lugar da lista de substituições. Com **ReplaceAll** essa lista é percorrida uma vez apenas, enquanto que com **ReplaceRepeated[expr, {x → val$_x$, y → val$_y$, …}]** a lista de substituições é percorrida até a expressão não mudar mais. Cuidado com esta forma porque pode originar ciclos infinitos.

```
x + y /. {x → γ, y → x + α}
```

```
x + α + γ
```

```
x + y //. {x → γ, y → x + α}
```

```
α + 2 γ
```

■ Conjuntos, Listas e Matrizes

□ List

Listas (ing. **List**) são colecções ou conjuntos de objectos com índices que podem ser unidimensionais (um só índice) ou multidimensionais (sequências de índices, quando os seus elementos incluirem por sua vez objectos do tipo **List**). Quaisquer símbolos ou expressões separadas por vírgulas (i.e. uma **Sequence**) dentro de chavetas **{a, b, …}** formam uma lista.

(a) – Os elementos de uma lista *list* são referenciados pelo operador **Part[list, i$_1$, i$_2$, …]** (alias **list[[i$_1$, i$_2$, …]]**) ou **Extract[list, positions]**.

```
{a, {b, c}}[[1]] == a
{a, {b, c}}[[2]] == {b, c}
{a, {b, c}}[[2, 1]] == b
{a, {b, c}}[[2, 2]] == c
```

(b) – O número de elementos no primeiro nível de uma lista é dado pelo operador **Length**[**list**], o número de níveis (contando com 0) é dado pelo operador **Depth**[**list**] e as dimensões duma lista por **Dimensions**[**list**]

```
Length[{a, {b, c}}] == 2
Depth[{a, {{b}, c}}] == 4
Dimensions[{{a, b}, {c, d}, {e, f}}] == {3, 2}
```

(c) – **Join**, **Union**, **Intersection** e **Complement** são operadores sobre conjuntos representados por listas.

```
a = {x, u, z};
b = {x, y, z};
Join[a, b] == {x, u, z, x, y, z}
(Union[a, b] ≡ a ⋃ b) == {u, x, y, z}
(Intersection[a, b] ≡ a ⋂ b) == {x, z}
Complement[a, b] == {u}
```

(d) – O operador **Flatten**[**list**] produz uma lista unidimensional de todos os elementos de list. **Flatten**[**list, n**] subtrai **n** ao número de níveis da lista. Em contrapartida o operador **Partition[list, n]** incrementa o número de níveis dividindo o primeiro nível da lista em conjuntos de **n** elementos. Se **Length[list]** não é divisível por n é possível usar **PadRight[list, m, ""]** para adicionar elementos vazios em número suficiente para não cortar elementos de list.

```
Flatten[{a, {{b}, c}}] == {a, b, c}
Flatten[{a, {{b}, c}}, 1] == {a, {b}, c}
Flatten[{a, {{b}, c}}, 2] == {a, b, c}
```

```
Partition[{a, b, c, d, e}, 2] == {{a, b}, {c, d}}
Partition[PadRight[{a, b, c, d, e}, 6, ""], 2] == {{a, b}, {c, d}, {e, ""}}
```

(e) – Os operadores **Sort[list], Reverse[list], RotateLeft[list, n]** e **RotateRight[list, n]** reordenam os elementos de **list**.

```
Sort[{b, d, c, a}] == {a, b, c, d}
Reverse[{b, d, c, a}] == {a, c, d, b}
RotateLeft[{b, d, c, a}, 2] == {c, a, b, d}
RotateRight[{b, d, c, a}, 3] == {d, c, a, b}
```

(f) – Os operadores **Drop[list, n]**, **Append[list, elements]** ou **Prepend[list, elements]** e **Insert[list, elements]** alteram listas.

```
Drop[{a, {b, c}, d}, −1] == {a, {b, c}}
Append[{a, {b, c}, d}, {e, f, g}] == {a, {b, c}, d, {e, f, g}}
(Insert[{a, {b, c}, d}, s[e, f, g], {2, 2}] /. s → Sequence) == {a, {b, e, f, g, c}, d}
```

(g) – Os operadores **Cases[list, pattern]**, **Select[list, criteria]** e **Pick[list, truthpattern]** escolhem os elementos de list que verificam determinados padrões ou critérios.

```
Cases[{a, c, e, d, f, b}, z_ /; OrderedQ[{z, d}]] == {a, c, d, b}
Select[{a, c, e, d, f, b}, OrderedQ[{#1, d}] &] == {a, c, d, b}
Pick[{a, c, e, d, f, b}, {True, False, True, True, False, False}] == {a, e, d}
```

□ RANGE

A função Range gera uma lista unidimensional.

```
Range[10]
```

```
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
```

```
Range[-10, 10]
```

$$\{-10, -9, -8, -7, -6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$$

```
Range[0, 2 π, π / 4]
```

$$\left\{0, \frac{\pi}{4}, \frac{\pi}{2}, \frac{3\pi}{4}, \pi, \frac{5\pi}{4}, \frac{3\pi}{2}, \frac{7\pi}{4}, 2\pi\right\}$$

```
?? Range
```

```
Range[imax] generates the list {1, 2, ... , imax}.
Range[imin, imax] generates the list {imin, ... , imax}.
Range[imin, imax, di] uses step di. More...
```

```
Attributes[Range] = {Listable, Protected}
```

□ Table e Array

As funções nativas **Table** e **Array** geram listas com dimensões variadas. A diferença é que **Table** usa índices explícitos **i, j, ...** para referenciar cada dimensão na sua definição, enquanto **Array** usa índices implícitos **#1, #2, ...**

```
?? Table
```

```
Table[expr, {imax}] generates a list of imax copies of expr.
Table[expr, {i, imax}] generates a list of the values of expr when i runs from 1 to imax.
Table[expr, {i, imin, imax}] starts with i = imin.
Table[expr, {i, imin, imax, di}] uses steps di.
Table[expr, {i, imin, imax}, {j, jmin, jmax}, ... ] gives a nested list.
The list associated with i is outermost. More...
```

```
Attributes[Table] = {HoldAll, Protected}
```

```
TableForm[Table[a_{i,j}, {i, 3}, {j, 2}]]
```

$$\begin{array}{ll} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \\ a_{3,1} & a_{3,2} \end{array}$$

```
TableForm[Array[a_{#1,#2} &, {3, 2}]]
```

$$\begin{array}{ll} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \\ a_{3,1} & a_{3,2} \end{array}$$

```
?? Array
```

Array[f, n] generates a list of length **n**, with elements **f[i]**. **Array[f, {n1, n2, ...}]** generates an **n1** by **n2** by ... array of nested lists, with elements **f[i1, i2, ...]**.
More...

```
i = 1;  Array[If[#1 == #2, i++, 0] &, {3, 3}] // MatrixForm
```

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{pmatrix}$$

Matrizes são simplesmente listas bi–dimensionais, i.e. uma lista cujos elementos são outras listas. Cuidado que o primeiro índice referencia linhas e não colunas ! Note–se o uso de **MatrixForm** para ver a matriz como é tradicional.

```
MatrixForm[Array[{{"coluna "[#2], "linha "[#1]}} &, {3, 2}]]
```

$$\begin{pmatrix} \text{coluna [1]} & \text{coluna [2]} \\ \text{linha [1]} & \text{linha [1]} \\ \text{coluna [1]} & \text{coluna [2]} \\ \text{linha [2]} & \text{linha [2]} \\ \text{coluna [1]} & \text{coluna [2]} \\ \text{linha [3]} & \text{linha [3]} \end{pmatrix}$$

```
MatrixForm[Array[lc[#1, #2] &, {3, 2}]] ≡ Array[lc[#1, #2] &, {3, 2}]
```

$$\begin{pmatrix} lc[1, 1] & lc[1, 2] \\ lc[2, 1] & lc[2, 2] \\ lc[3, 1] & lc[3, 2] \end{pmatrix} \equiv \{\{lc[1, 1], lc[1, 2]\}, \{lc[2, 1], lc[2, 2]\}, \{lc[3, 1], lc[3, 2]\}\}$$

```
<< Statistics`DataManipulation`
```

```
% // Column[#, 1] & // MatrixForm
```

$$\begin{pmatrix} \text{coluna [1]} \\ \text{linha [1]} \\ \text{coluna [1]} \\ \text{linha [2]} \\ \text{coluna [1]} \\ \text{linha [3]} \end{pmatrix}$$

■ Variáveis-Modelo e Expressões Conformes a um modelo.

A variável–modelo _ ou **Blank**[] faz o papel de substituto para qualquer objecto simbólico ou numérico do *Mathematica* mas numa sequência de objectos _ identifica–se com um só. Para designar uma sequência qualquer de objectos existe a variável–modelo __ (dois _ seguidos) ou **BlankSequence[]**, e para designar uma sequência eventualmente vazia de objectos dispomos de ___ (três _ seguidos) ou **BlankNullSequence**[]. A função nativa **MatchQ**[expr, modelo] mostra se uma expressão se conforma ao modelo ou não.

```
2 E^x^2 == 2 * E^x^2
MatchQ[2 E^x^2, _] → True
MatchQ[2 E^x^2, _ * _] → False
MatchQ[2 E^x^2, _ * Power[__]] → True
MatchQ[2 E^x^2, _ * __] → True
MatchQ[2 E^x^2, Times[___]] → True
```

As variáveis–modelo do género **Blank** podem restringir o seu campo de acção declarando o tipo de operação que pretendem imitar. Assim **_Plus** ou **Blank[Plus]** apenas se identifica com somas, **_Times** com produtos, etc.

```
MatchQ[b^2, _Power] → True
MatchQ[c/2, _Times] → True
MatchQ[a + b^2 + c/2, _Plus] → True
MatchQ[a + b^2 + c/2, _Plus + _Power] → True
MatchQ[a + b^2 + c/2, _Plus + _Times] → True
MatchQ[a + b^2 + c/2, _Plus + ___Power + _Times] → True
MatchQ[a + b^2 + c/2, _Plus + _Power + _Times] → False
```

Note que **Plus** ou **Times** implícitamente envolvem dois ou mais argumentos, por isso $a + b^2 + \frac{c}{2} = \left(a + \frac{c}{2}\right) + b^2 = (a + b^2) + \frac{1}{2} * c$ pode ser representado por **_Plus + _Power** ou por **_Plus + _Times**, mas não por **_Plus + _Power + _Times**.

(a) – Uma expressão–modelo (ing. **Pattern**) é qualquer expressão envolvendo uma ou mais variáveis–modelo. *Em geral é útil dar–lhe um nome, por isso* **z_** ou **Pattern**[z, Blank[]] representa uma expressão–modelo com o nome **z**. A variável–modelo **z_Plus** ou Pattern[z,Blank[Plus]] tem o nome **z** durante os testes e representa qualquer expressão do tipo **Plus**.

```
MatchQ[a + b, z_]                  ⟶   True
MatchQ[a + b, z_Plus]              ⟶   True
MatchQ[(a + b)^2, z_Plus]          ⟶   False
MatchQ[(a + b)^2, z_Power]         ⟶   True
MatchQ[1/(a+b)^2, z_Plus^n_]       ⟶   True
```

(b) – Uma utilização frequente de expressões–modelo envolve a substituição por outras expressões de partes duma expressão que sejam conformes ao modelo. Por exemplo a expressão modelo **{a_, b_}** ajusta–se a **{x, y}** mas não a **{x, y, z}**.

```
f[{x, y}, {x, y, z}] /. {a_, b_} → {b, a}
```

```
f[{y, x}, {x, y, z}]
```

```
b + c.a /. z_ /; MatchQ[Head[z], Dot] :→ z[[2]]
```

```
a + b
```

```
c.a + b /. (z_Dot?(#[[1]] == c  &)) :→ 2
```

```
2 + b
```

(c) – O modelo **patt?test** é uma forma de seleccionar elementos **y** de expressões **expr** que além de verificarem **MatchQ[y, patt] == True**, também verificam **test[y] == True**.

```
x^2 – Sin[y] /. _Sin?(#[[1]] == y &) :→ 1
```

```
–1 + x^2
```

```
Head[Sin[y][[1]]] == Symbol
```

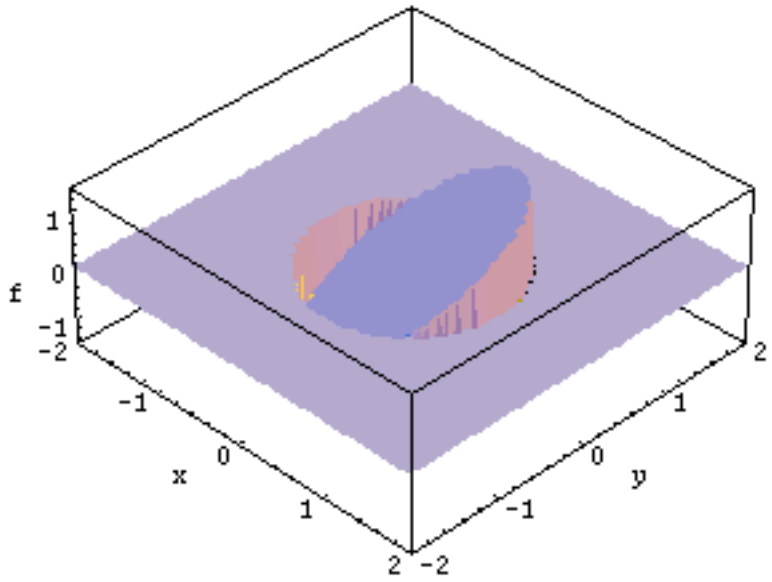
```
True
```

■ Estruturas Condicionais e Testes

Estruturas condicionais são expressões que incluem operadores que usam testes para definir o resultado final. Por exemplo, para manipular expressões é necessário frequentemente aplicar operadores a partes da expressão que passem determinados testes. As funções nativas do Mathematica que seleccionam e operam em partes de uma expressão usam frequentemente estas estruturas condicionais. São exemplos frequentes as funções como **Cases, Position, Select** que procuram dentro de uma expressão todas as ocorrências de uma sub–expressão que se conforme a um molde dado. Outra utilização frequente é na definição do domínio de aplicação de uma função, impondo condições nos argumentos desta que são testados na altura da avaliação e determinam qual das várias definições da função se deve aplicar (i.e. aquela que testar **True** nas condições sobre os argumentos). Por exemplo a seguinte função tem um valor **x + y** dentro do círculo de raio unitário no plano e zero fora.

```
f[x_, y_] := Condition[x + y, x^2 + y^2 < 1]
f[x_, y_] := 0
```

```
Plot3D[f[x, y], {x, –2, 2}, {y, –2, 2}, PlotPoints → 90,
    Mesh → False, ViewPoint → 5 {1, – 1, 1}, AxesLabel → {x, y, f}];
```



As funções nativas mais comuns que usam estruturas condicionais são os operadores:

(a) – **Cases[expr, pattern]** dá as ocorrências de sub–expressões de **expr** que sejam da forma de **pattern**.

(b) – **Position[expr, pattern]** dá as posições de sub–expressões de **expr** que sejam da forma de **pattern**.

```
Cases[x + f[z, y] + g[x], _[__]]
```

```
{f[z, y], g[x]}
```

```
MatchQ[#, _[__]] & /@ (List @@ (x + f[z, y]))
```

```
{False, True}
```

```
Position[x^2 + f[z, y^3], _f]
Position[x^2 + f[z, y^3], _Power]
Position[x^2 + f[z, y^3], z]
```

```
{{2}}
{{1}, {2, 2}}
{{2, 1}}
```

(c) – **If [test, value_if _true, value_if _false (, value_if _undecided)]** dá o resultado correspondente ao valor do **test** ser **True**, **False** ou **Indeterminado**. Este último argumento é opcional.

(d) – **Switch [expr, $patt_1$, val_1, $patt_2$, val_2, ...]** dá o primeiro dos valores val_i para o qual se verifica que **MatchQ[expr, $patt_i$] == True**

```
Switch[z^3, x_^2, √x, x_Power, y^(1/3)]
```

```
y^(1/3)
```

(e) – **Which [test$_1$, val$_1$, test$_2$, val$_2$, ...]** dá o primeiro val$_i$ para o qual **test$_i$ == True**

```
Which[2 > 3, ah, 3 > 2, b]
```

```
b
```

Uma lista de testes disponíveis para as diversas estruturas nativas do Mathematica pode obter–se enumerando todos os casos de símbolos terminados em Q.

```
Cases[Names["*"], z_ /; StringMatchQ[z, "*@Q"]]
```

ArgumentCountQ	InverseEllipticNomeQ	MemberQ	StringFreeQ
ArrayQ	LegendreQ	NameQ	StringMatchQ
AtomQ	LetterQ	NumberQ	StringQ
DigitQ	LinkConnectedQ	NumericQ	SyntaxQ
EllipticNomeQ	LinkReadyQ	OddQ	TensorQ
EvenQ	ListQ	OptionQ	TrueQ
ExactNumberQ	LowerCaseQ	OrderedQ	UnsameQ
FreeQ	MachineNumberQ	PartitionsQ	UpperCaseQ
InexactNumberQ	MatchLocalNameQ	PolynomialQ	ValueQ
IntegerQ	MatchQ	PrimeQ	VectorQ
IntervalMemberQ	MatrixQ	SameQ	

A forma destas estruturas condicionais varia conforme se trate de:

(a) – comparação com expressões booleanas estáticas (por exemplo **Equal** (==), **SameQ** (≡) e respectivas negações **Unequal** (≠), **UnsameQ** (≢)) ou em geral qualquer função cujo resultado seja True ou False.

```
Select[f[x^3, y, 1/y], (Head[#] == Power) &]
```

$$f\left[x^3, \frac{1}{y}\right]$$

(b) – comparações com modelos de expressão **patt** condicionados a um teste **test** na forma **patt /; test** (ou **Condition[patt, test]**)

```
Cases[{x, x^2, 1/x}, z_ /; Not[FreeQ[z, Power]]]
```

$$\left\{x^2, \frac{1}{x}\right\}$$

```
FullForm[1/x]
```

```
Power[x, -1]
```

(c) – ou comparações via **patt?test** (ou **PatternTest[patt, test]**) com modelos de expressão **patt** construídos a partir do modelo elementar _ ou Blank[].

```
Cases[1/x + x + x^2, z_?AtomQ]
```

```
{x}
```

```
NumberQ[#] & /@ {π, N[π], 3, 4/5, ∞, i}
```

```
{False, True, True, True, False, True}
```

```
Cases[{π, N[π], 3, 4/5, ∞, i}, z_?NumberQ]
```

```
{3.14159, 3, 4/5, i}
```

```
NumericQ[#] & /@ {π, N[π], 3, 4/5, ∞, i}
```

```
{True, True, True, True, False, True}
```

```
Cases[{π, N[π], 3, 4/5, ∞, i}, z_?NumericQ]
```

```
{π, 3.14159, 3, 4/5, i}
```

(d) – Finalmente, comparações directas entre expressões **expr** e modelos **patt** através de testes do tipo **MatchQ[expr, patt]**. Para strings deve usar–se antes **StringMatchQ** e modelos envolvendo * (zero ou mais carateres) ou "@" (um ou mais caracteres excluindo maiúsculas).

```
MatchQ[#, _^2] & /@ {x, x^2, 1/x}
```

```
{False, True, False}
```

```
Cases[{x, x^2, 1/x}, z_ /; MatchQ[z, _^2]]
```

```
{x^2}
```

■ Ciclos

Os ciclos são controlados por operadores **Do**, **For**, **While**, **Break** e **Continue**.

```
? Do
```

Do[expr, {imax}] evaluates expr imax times.

```
x = 1;
Do[Print[x = x / 2.], {8}];
```

```
0.5
0.25
0.125
0.0625
0.03125
0.015625
0.0078125
0.00390625
```

Do[expr, {i, imax}] evaluates **expr** with the variable **i** successively taking on the values 1 through **imax** (in steps of 1).

```
Do[Print[x^i], {i, 3}]
```

```
x
x^2
x^3
```

Do[expr, {i, imin, imax}] starts with **i = imin**.

```
Do[Print[f[i]], {i, 2, 4}]
```

```
f[2]
f[3]
f[4]
```

Do[expr, {i, imin, imax, di}] uses steps **di**.

Do[expr, {i, imin, imax}, {j, jmin, jmax}, ...] evaluates **expr** looping over different values of **j**, etc. for each **i**.

```
matriz = {};
Do[AppendTo[matriz, a_{i,j}], {i, 0, 2}, {j, 1, 3}];
Print[matriz]
```

$\{a_{0,1}, a_{0,2}, a_{0,3}, a_{1,1}, a_{1,2}, a_{1,3}, a_{2,1}, a_{2,2}, a_{2,3}\}$

? For

For[start, test, incr, body] executes **start**, then repeatedly evaluates **body** and **incr** until **test** fails to give **True**.

```
Clear[x, f];
For[j = 1, j ≤ 4, j++, Print[# == (# /. x → 2.)] &@Nest[1/(1 + #) &, x, j]]
```

$\frac{1}{1+x} == 0.5$

$\frac{1}{1+\frac{1}{1+x}} == 0.666667$

$\frac{1}{1+\frac{1}{1+\frac{1}{1+x}}} == 0.6$

$\frac{1}{1+\frac{1}{1+\frac{1}{1+\frac{1}{1+x}}}} == 0.625$

? While

While[test, body] evaluates test, then body, repetitively, until test first fails to give True.

? Break

Break[ ] exits the nearest enclosing Do, For or While.

More...

? Continue

Continue[ ] exits to the nearest enclosing Do, For or While in a procedural program.

More...

Funções em *Mathematica*

■ Definição de Função

O conceito de função no **Mathematica** é mais geral que apenas o de função numérica com argumentos numéricos reais ou complexos. De facto no **Mathematica** podem–se definir funções ou operadores com argumentos quaisquer, sejam expressões ou segmentos de texto (ing. **Strings**) ou mesmo gráficos e células de Notebooks.

- Os argumentos de funções são especificados entre parêntesis rectos [...].
- As funções nativas do MATHEMATICA começam sempre por uma letra Maiúscula.
- A potenciação é representada pelo símbolo ^, o produto por * e a divisão por / : a^b, a*b, a/b
- Um espaço entre dois símbolos ou números é interpretado como um produto : a*b = a b
- A forma normal de representar potências e divisões é bidimensional : a^b = a^b; a/b = $\frac{a}{b}$
- É admissível usar notação bidimensional na definição de funções.
- Números em notação científica podem ser usados : 0.025 ou 2.5*10^–2 ou $2.5\,10^{-2}$
- O símbolo E está reservado como a base do números neperianos E = e = 2.71828
- O símbolo I está reservado como o complexo puro unitário I = $i = \sqrt{-1}$
- O símbolo π está reservado como o irracional π = Pi = 3.14159

A sintaxe de definição e invocação de funções é bastante simples mas requer algum cuidado dada a flexibilidade do Mathematica fazer atribuições imediatas ou retardadas às próprias expressões que definem as funções.

Em geral **f[x_1, x_2, ...]** representa uma função **f** definida pelo utilizador, e que depende de variáveis **x_1, x_2, ...** Normalmente pretende–se que estas variáveis só sejam substituídas por valores na altura da invocação da função, por exemplo **f[π, a, ...]**, por isso na altura da definição de f as variáveis **x_1, x_2, ...** devem ser substituídas por **Padrões** (ing. **Patterns**) que são símbolos quaisquer seguidos de um ou mais caracteres de sublinhado _ (designado **Blank[]**), ou seja **x1_, x2_, ...**

Por exemplo, na definição **f[x_, y_] = Tanh[$x^3 + \frac{A}{y^2}$]** os padrões **x_, y_** representam os argumentos 1 e 2 de f, expressões arbitrárias a que se dá o nome **x, y** para efeitos da definição de **f**. Qualquer valor introduzido como 1º argumento de **f** substitui **x** na definição, e o 2º argumento substitui **y**.

$$f[x\_, y\_] = \text{Tanh}\left[x^3 + \frac{A}{y^2}\right]$$

$$\text{Tanh}\left[x^3 + \frac{A}{y^2}\right]$$

Esta definição é associada globalmente ao símbolo f.

```
? f
```

```
Global`f
```

$$f[x\_, y\_] = \text{Tanh}\left[x^3 + \frac{A}{y^2}\right]$$

Com esta definição,

```
f[a, b]
```

$$\text{Tanh}\left[a^3 + \frac{A}{b^2}\right]$$

Os argumentos **x** e **y** de **f** podem ser qualquer expressão do *Mathematica*, incluíndo texto.

```
f["hello", "bye"]
```

$$\text{Tanh}\left[\text{hello}^3 + \frac{A}{\text{bye}^2}\right]$$

```
f[Cos[π], Sqrt[A/π]]
```

$$-\text{Tanh}[1 - \pi]$$

A avaliação numérica explícita pode ser invocada com o operador N.

```
N[f[Cos[π], Sqrt[A/π]]]
```

```
0.972778
```

A declaração do tipo de argumento também é facultativa, já que a representação interna das expressões do Mathematica determina à partida o seu tipo através do operador **Head**. O operador **FullForm** dá explícitamente a representação unidimensional da expressão enviada para avaliação pelo Kernel.

4	FullForm[4]	Head[4]	
4.	FullForm[4.]	Head[4.]	
$\frac{4}{3}$	FullForm[$\frac{4}{3}$]	Head[$\frac{4}{3}$]	
π	FullForm[π]	Head[π]	
∞	FullForm[∞]	Head[∞]	
$-\infty$	FullForm[$-\infty$]	Head[$-\infty$]	
i	FullForm[i]	Head[i]	
4 + 2 I	FullForm[4 + 2 I]	Head[4 + 2 I]	
a + 2 I	FullForm[a + 2 I]	Head[a + 2 I]	
"4+2 I"	FullForm["4+2 I"]	Head["4+2 I"]	
a^2	FullForm[a^2]	Head[a^2]	// TableForm
a^2	FullForm[a^2]	Head[a^2]	
x_1	FullForm[x_1]	Head[x_1]	
x_1^3	FullForm[x_1^3]	Head[x_1^3]	
x_1^3	FullForm[x_1^3]	Head[x_1^3]	
$\overset{a}{x}$	FullForm[$\overset{a}{x}$]	Head[$\overset{a}{x}$]	
$\underset{c}{\overset{1}{x}}$	FullForm[$\underset{c}{\overset{1}{x}}$]	Head[$\underset{c}{\overset{1}{x}}$]	
z_	FullForm[z_]	Head[z_]	
x__	FullForm[x__]	Head[x__]	
x_?NumberQ	FullForm[x_?NumberQ]	Head[x_?NumberQ]	

4	4	Integer
4.	4.‘	Real
$\frac{4}{3}$	Rational[4, 3]	Rational
π	Pi	Symbol
∞	DirectedInfinity[1]	DirectedInfinity
$-\infty$	DirectedInfinity[−1]	DirectedInfinity
i	Complex[0, 1]	Complex
$4+2\,i$	Complex[4, 2]	Complex
$2\,i+a$	Plus[Complex[0, 2], a]	Plus
4+2 I	"4+2 I"	String
a^2	Power[a, 2]	Power
a^2	Superscript[a, 2]	Superscript
x_1	Subscript[x, 1]	Subscript
x_1^3	Power[Subscript[x, 1], 3]	Power
x_1^3	Subsuperscript[x, 1, 3]	Subsuperscript
$\overset{a}{x}$	Overscript[x, a]	Overscript
$\underset{c}{\overset{1}{x}}$	Underoverscript[x, c, 1]	Underoverscript
z_	Pattern[z, Blank[]]	Pattern
x__	Pattern[x, BlankSequence[]]	Pattern
x_?NumberQ	PatternTest[Pattern[x, Blank[]], NumberQ]	PatternTest

Note que algumas expressões bi−dimensionais são idênticas mas têm avaliações diferentes: usando **HoldForm** para evitar a formatação automática do Output em notação **StandardForm** bi−dimensional, pode−se ver que **Superscript** e **Power** têem formatações iguais, mas resultados distintos.

```
(FullForm[Superscript[4, 2]] == Superscript[4, 2]) ≠
 (HoldForm[StandardForm[Power[4, 2]]] == Power[4, 2])
```

$$\left(\text{Superscript}[4,\,2] == 4^2\right) \neq \left(4^2 == 16\right)$$

■ Operadores e Funções Simbólicas

Eis alguns **aliases** para operadores, padrões e funções nativas do **Mathematica .**

```
alias          = (Alias[]) /. System‘Private‘ValueList :> List;
reverserules = Map[#1[[{2, 1}]] &, alias, 1] /. (z_ :> y_) → ({z, y});
sortedalias  = (GridBox[#,
                RowLines −> True, ColumnLines −> True, ColumnAlignments −> Left,
                GridFrame −> True] & /@ Partition[Sort[reverserules], 21]) // DisplayForm
```

AddTo	+=	Less	<	Rule	->
Alternatives	|	LessEqual	<=	RuleDelayed	:>
And	&&	List	{}	SameQ	===
Apply	@@	Map	/@	Set	=
Blank	_	MapAll	//@	SetDelayed	:=
BlankNullSequence	___	MessageName	::	Slot	#
BlankSequence	__	NonCommutativeMultiply	**	SlotSequence	##
CompoundExpression	;	Optional	_.	String	"
Condition	/;	Or	||	StringJoin	<>
Decrement	--	Out	%	Subtract	-
Divide	/	Part	[[ ]]	SubtractFrom	-=
DivideBy	/=	PatternTest	?	TagSet	/:
Dot	.	Plus	+	Times	*
Equal	==	Power	^	TimesBy	*=
Factorial2	!!	Put	>>	UnAlias	:: =.
Function	&	PutAppend	>>>	Unequal	!=
Get	<<	Repeated	..	UnsameQ	=!=
Greater	>	RepeatedNull	...	Unset	=.
GreaterEqual	>=	ReplaceAll	/.	UpSet	^=
Increment	++	ReplaceRepeated	//.	UpSetDelayed	^:=

Tabela 7

■ FUNÇÕES MATEMÁTICAS COMUNS

As funções matemáticas mais comuns estão definidas no *Mathematica* e podem ser invocadas usando o nome usualmente convencionado a começar por uma maíuscula. A seguinte tabela mostra alguns exemplos, juntamente com algumas notações bi–dimensionais que o FrontEnd entende e traduz automáticamente

Sqrt[*x*]	*raiz quadrada* $\left(\sqrt{x}\right)$
Exp[*x*]	*exponencial* e^x
Log[*x*]	*logaritmo neperiano* $\log_e x$
Log[*b* , *x*]	*logaritmo de base b* $\log_b x$
Sin[*x*] , *Cos*[*x*] , *Tan*[*x*]	*funções trigonométricas* (*argumentos em radianos*)
ArcSin[*x*] , *ArcCos*[*x*] , *ArcTan*[*x*]	*funções trigonométricas inversas*
n !	*factorial* (*produto de inteiros 1,2,...,n*)
Abs[*x*]	*valor absoluto*
Round[*x*]	*inteiro mais próximo de x*
Mod[*n* , *m*]	*n modulo m* (*resto da divisão inteira de n por m*)
Random[ ]	*número pseudo–random entre 0 e 1*
Max[*x* , *y* , ...], *Min*[*x* , *y* , ...]	*maximum, minimum de x , y , ...*
FactorInteger[*n*]	*factores primos de n*

Para além destas muitas outras existem que podem ser invocadas directamente ou após carregar um módulo auxiliar (**Package**) apropriado. Com base nestas funções nativas o utilizador pode

construir as suas próprias funções com valores reais, complexos, vectoriais ou tensoriais, simbólicos ou de texto.

Sin[x] é uma função 'nativa' com um argumento x, que pode ser qualquer expressão do Mathematica. A menos que a função tenha regras explícitas para isso, a avaliação só dá um resultado numérico *explícito* se o argumento for um número real não racional, ou se o solicitarmos explícitamente usando o operador N[expr]

$\text{Sin}[\frac{3}{5}]$

$\text{Sin}[\frac{3}{5}]$

$\text{Sin}[\frac{3}{5}\pi]$

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

$\text{Sin}[\frac{3.}{5}]$

0.564642

$\text{N}[\text{Sin}[\frac{3}{5}]]$

0.564642

■ Formas alternativas de escrever funções

No Mathematica é possível escrever de várias maneiras equivalentes a aplicação de uma função com designação (ing. **Head**) f aos seus argumentos. Note que

Prefix[f[expr]]	⟺	f @ expr	Sin @ x	Sin[x]
Postfix[f[expr]]	⟺	expr // f	x // Sin	Sin[x]
Infix[g[x_, y_], ⊡]	⟺	x ⊡ y	x.y	Dot[x, y]
Apply[f, expr]	⟺	f @@ expr	f @@ {x, y}	f[x, y]
Map[f, expr]	⟺	f /@ expr	f /@ {x, y}	{f[x], f[y]}
ReplaceAll[expr, Head[expr] → f]	⟺	expr /. Head[expr] → f	x + y /. Plus :→ f	f[x, y]

Uma última forma é fazer Sin[2 + x y] /. z_Times → f, o que dá Sin[2 + f[x, y]], porque Head[x y] → Times, por isso x y verifica o modelo: z_Times (z_h expressão de nome z com 'cabeça' h – ver pág. 572) A expressão x y é substituída por isso pela expressão f[x y]. É necessário muito

cuidado com esta forma de aplicar funções, porque as substituições podem acontecer onde não se pretende.

Por exemplo:

```
Sin[x y – 2] /. z_Times :⇢ (f @@ z)
```

```
f[–1, Sin[2 – x y]]
```

```
Sin[x y – 2] // FullForm
```

```
Times[–1, Sin[Plus[2, Times[–1, x, y]]]]
```

Uma maneira de resolver isto é saber de antemão o 'modelo' da expressão 'x y' e substituir então por f

```
Sin[x y – 2] /. Times[–1, a_Symbol, b_Symbol] :⇢ (–f[a, b])
```

```
–Sin[2 – f[x, y]]
```

Se f[x, y] já se definiu anteriormente, para ver todas as definições associadas a f, usa–se ?f (ou Definition[f]).

Se f[x, y] envolvesse expressões que foram também definidas pelo utilizador, então ??f (ou FullDefinition[f]) mostra as definições de todas as expressões que entram directa ou indirecta–mente na definição de f.

```
Definition[f]
```

$$f[x\_, y\_] = \text{Tanh}\left[x^3 + \frac{A}{y^2}\right]$$

```
x = 3;          y = a;          z = b;
FullDefinition[f]
```

$$f[x\_, y\_] = \text{Tanh}\left[x^3 + \frac{A}{y^2}\right]$$

$$x = 3$$

$$y = a$$

Funções com vários argumentos f[var1, var2, …] podem–se escrever com regras semelhantes, tendo em conta que var1, var2, … é uma sequência de expressões e não uma expressão só.

```
f @ Sequence[x, y]
```

```
f[x, y]
```

```
f @@ {x, y}          (*usando' Apply' sobre uma lista*)
```

```
f[x, y]
```

```
f @ {x, y}           (*não funciona assim*)
```

```
f[{x, y}]
```

```
f @ Sequence[x, y]
```

```
f[x, y]
```

```
f @ {x, y} /. List :→ Sequence          (*cuidado com esta forma*)
```

```
f[x, y]
```

```
f @ {x, {y, x}} /. List :→ Sequence
```

```
f[x, y, x]
```

```
(*mas*)  f @@ {x, {y, x}}
```

```
f[x, {y, x}]
```

■ Funções puras

def & e *Function*[{*x, y, …*}, *def*]

Em muitos casos torna–se desnessário dar um nome a uma função.(Por exemplo, quando a usa apenas uma vez ou só temporáriamente para simplificar ou esclarecer um cálculo.) Para isso o *Mathematica* disponibiliza funções puras.

■ **def &** define uma função pura que toma valores assim que se aplicar a uma sequência de valores para as variáveis através de **def &[vars]**. Os argumentos da função são referidos na definição def por #1 (ou só #), #2, etc. conforme a sua posição sequencial.

```
(#2^2 – #1) &[a, b]
```

```
–a + b^2
```

```
(Sin[#1]^3 &)[a + b]
```

```
Sin[a + b]^3
```

```
(#1[[1]]^3 &) @ (a + b)
```

```
a^3
```

```
{{a, a}, {g, s}, {c, s}} // (GridBox[#, ColumnLines → True] &)
```

```
GridBox[{{a, a}, {g, s}, {c, s}}, ColumnLines → True]
```

```
% // DisplayForm
```

a	a
g	s
c	s

▪ **Function[x, def]** define uma função pura com uma variável referida por **x**, que é substituído pelo valor $\mathbf{val_x}$ quando a função é avaliada com **Function[x, def]**[$\mathrm{val_x}$].

A derivação seguinte não dá o resultado esperado porque a substituição não é feita: usando **FullForm** vê-se porquê.

```
∂x f[x] /. f[x] → Sqrt[x^2 + 1]
% // FullForm
```

```
f'[x]
```

```
Derivative[1][f][x]
```

A substituição de **f** por uma função pura permite concluir o cálculo da derivada.

```
∂x f[x] /. f → Function[{x}, Sqrt[x^2 + 1]]
```

$$\frac{x}{\sqrt{1+x^2}}$$

```
f'[x]
```

▪ **Function[{$\mathbf{x_1}$, $\mathbf{x_2}$, ...}, def]** define uma função pura com variáveis múltiplas referidas sucessivamente por $\mathbf{x_1}$, $\mathbf{x_2}$, ... que são substituídas pelos valores $\mathrm{val_{x_1}}$, $\mathrm{val_{x_2}}$, ... quando a função é avaliada com **Function[{$\mathbf{x_1}$, $\mathbf{x_2}$, ...}, def]**[$\mathbf{val_{x_1}}$, $\mathbf{val_{x_2}}$, ...]

```
f = Function[{x, y}, x^2 - y];
f[a, b]
```

```
a^2 - b
```

Note que **Attributes[Function] = {HoldAll, Protected}**, ou seja nada dentro de **Function** é avaliado ou simplificado na altura da sua definição. Os símbolos x_1, x_2, ... que aparecem no primeiro argumento, e quaisquer símbolos que apareçam adicionalmente em **def** no segundo

argumento, são estritamente locais em **Function**, só sendo avaliados na altura da invocação da função

```
a = π;
f = Function[{x, y}, x^2 – a y]
f[X, Y]
```

```
Function[{x, y}, x^2 – a y]
```

```
X^2 – π Y
```

■ Atributos e Propriedades de Funções

O comportamento das funções nativas do Mathematica (e das funções definidas pelo utilizador que as usam) relativamente aos seus argumentos é controlado por atributos que podem ser opcionalmente alterados (desde que a função não tenha o atributo Locked).

A maioria destes atributos diz respeito ao comportamento da função relativamente aos seus argumentos: **Plus** e **Times** são óbviamente funções de múltiplos argumentos para as quais a ordem destes é irrelevante (x + y = y + x e x y = y x), e portanto têm o atributo **Orderless**. Qualquer das operações matemáticas vulgares é igualmente **Listable** ({a, b, c}^n = {a^n, b^n, c^n} ou √{a, b, c} = {√a, √b, √c}), enquanto que as operações lógicas como **And** e **Or** têm atributos **HoldHall**. Para determinar os atributos de uma função basta fazer

```
Attributes[Plus]
```

```
{Flat, Listable, NumericFunction, OneIdentity, Orderless, Protected}
```

```
Attributes[Times]
```

```
{Flat, Listable, NumericFunction, OneIdentity, Orderless, Protected}
```

```
Attributes[Sin]
```

```
{Listable, NumericFunction, Protected}
```

Para alterar alguns destes atributos usa–se ClearAttributes[function, {attr$_1$, attr$_2$ …}] para eliminar atributos ou SetAttributes[function, {attr$_1$, attr$_2$ …}] para adicionar. Por exemplo, o atributo Orderless de Plus verifica

```
b + a == a + b
```

```
True
```

Fazendo Plus não–comutativo nos seus argumentos significa que b + a já não é necessáriamente o mesmo que a + b. (Óbviamente é desejável repôr a propriedade comutativa de Plus para o normal funcionamento.)

```
Unprotect[Plus];
ClearAttributes[Plus, Orderless]
b + a == a + b
SetAttributes[Plus, Orderless]
Protect[Plus];
```

```
b + a == a + b
```

Orderless	orderless, commutative function (arguments are sorted into standard order)
Flat	flat, associative function (arguments are "flattened out")
OneIdentity	$f[f[a]]$, etc. are equivalent to a for pattern matching
Listable	f is automatically threaded over lists that appear as arguments (f[{a, b}] becomes {f[a], f[b]})
Constant	all derivatives of f are zero
NumericFunction	f is assumed to have a numerical value when its arguments are numeric quantities
Protected	values of f cannot be changed
Locked	attributes of f cannot be changed
ReadProtected	values of f cannot be read
HoldFirst	the first argument of f is not evaluated
HoldRest	all but the first argument of f is not evaluated
HoldAll	none of the arguments of f are evaluated
HoldAllComplete	the arguments of f are treated as completely inert
NHoldFirst	the first argument of f is not affected by N
NHoldRest	all but the first argument of f is not affected by N
NHoldAll	none of the arguments of f are affected by N
SequenceHold	Sequence objects appearing in the arguments of f are not flattened out
Temporary	f is a local variable, removed when no longer used
Stub	Needs is automatically called if f is ever explicitly input

The Mathematica Book: sect. 2.6.3

O seguinte código mostra exemplos de funções nativas que têm um dos atributos da lista de cima. Use–a para perceber a utilidade dos atributos em geral.

```
withAttr[attr_] := Select[Names["*"], MemberQ[Attributes[#], attr] &] // TableForm
```

```
withAttr[Orderless]
```

```
ArithmeticGeometricMean
DiracDelta
DiscreteDelta
GCD
KroneckerDelta
LCM
Max
Min
Multinomial
Plus
Times
UnitStep
Xor
```

Quase todas as funções nativas do *Mathematica* têm o atributo **Protected**, mas alterações às suas definições ainda se podem fazer usando **Unprotect[]**, ao contrário da funções nativas com atributo **Locked**, que não podem ser alteradas.

Um exemplo de função protegida é a função trigonométrica nativa tangente **Tan[x]**. A expansão trigonométrica **TrigExpand** de **Tan[x + y]** dá um resultado que pode não ser o mais pretendido, sendo necessário realizar uma série de operações para atingir a forma requerida:

```
TrigExpand[Tan[x + y]]
```

$$\frac{\text{Cos[y]}\,\text{Sin[x]} + \text{Cos[x]}\,\text{Sin[y]}}{\text{Cos[x]}\,\text{Cos[y]} - \text{Sin[x]}\,\text{Sin[y]}}$$

```
% /. Sin[z_] :→ tan[z] Cos[z]     (* Note que se escreveu tan[z] e não Tan[z]. Porquê? *)
```

$$\frac{\text{Cos[x]}\,\text{Cos[y]}\,\text{tan[x]} + \text{Cos[x]}\,\text{Cos[y]}\,\text{tan[y]}}{\text{Cos[x]}\,\text{Cos[y]} - \text{Cos[x]}\,\text{Cos[y]}\,\text{tan[x]}\,\text{tan[y]}}$$

```
% // Simplify
```

$$\frac{\text{tan[x]} + \text{tan[y]}}{1 - \text{tan[x]}\,\text{tan[y]}}$$

```
% /. tan :→ Tan
```

$$\frac{\text{Tan[x]} + \text{Tan[y]}}{1 - \text{Tan[x]}\,\text{Tan[y]}}$$

A função **Tan** pode ter alguma vantagem em ser alterada da seguinte forma:

```
Unprotect[Tan];
Tan[x_ + y_] := (Tan[x] + Tan[y]) / (1 - Tan[x] Tan[y])
Protect[Tan];
```

Agora tem-se automáticamente

```
Tan[a + b/c]
```

$$\frac{\text{Tan}[a] + \text{Tan}\left[\frac{b}{c}\right]}{1 - \text{Tan}[a]\,\text{Tan}\left[\frac{b}{c}\right]}$$

■ Argumentos facultativos e Opções em funções

Para definir uma função que depende de algum parâmetro que típicamente toma um valor pré-definido, mas que eventualmente gostaríamos de alterar, é possível usar o modelo z_: zval para adicionar um argumento facultativo, com um valor zval quando omisso, na atribuição **f[x1_, x2_, ..., xn_, z_: zval] := def**. Nesta forma, uma avaliação envolvendo **f[x_1, x_2, ..., x_n]** é equivalente a invocar **f[x_1, x_2, ..., x_n, zval]**.

Uma forma alternativa de especificar valores pré-definidos para argumentos de **f** é através da função **Default[f, k] := kval** onde **kval** indica o valor do k-ésimo argumento quando omisso de **f**, sempre que a forma **zk_.** é usada nessa posição para indicar que esse argumento é facultativo, i.e. **f[x_1, ..., zk_.] := def**

```
$Line = 0;
```

```
ClearAll[f, x]
```

```
f[x_, n_: 1] := Tan[x + n]
```

```
f[y]
```

```
Tan[1 + y]
```

```
f[y, 1] == f[y]
```

```
True
```

```
f[y, 2]
```

```
Tan[2 + y]
```

```
ClearAll[f];
Default[f, 2] = 3;
Default[f, 3] = 1;
f[x_, z_., n_.] := Tan[x + n + π/z]
```

```
?? f
```

```
Global`f

f[x_, z_., n_.] := Tan[x + n + π/z]

f /: Default[f, 2] = 3

f /: Default[f, 3] = 1
```

```
f[y]
```

$$\text{Tan}\left[1 + \frac{\pi}{3} + y\right]$$

```
f[y, x]
```

$$\text{Tan}\left[1 + \frac{\pi}{x} + y\right]$$

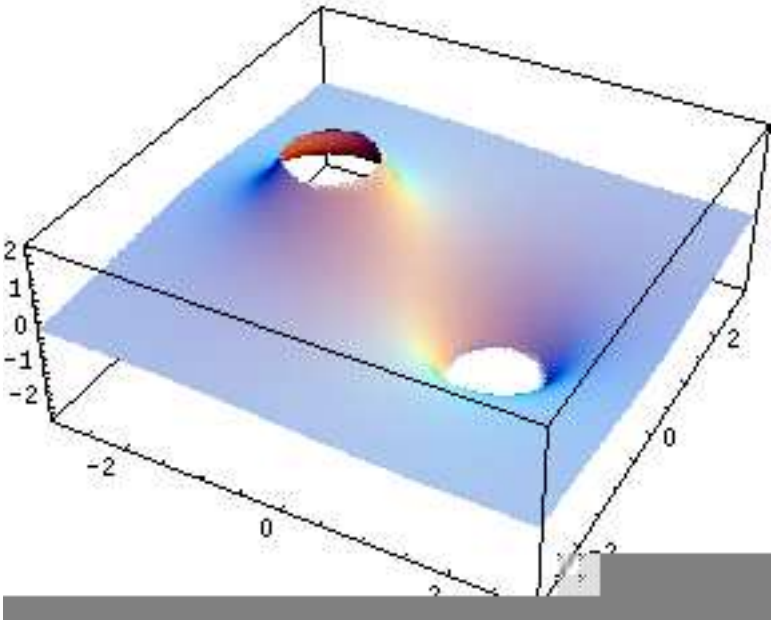
```
f[y, x, s]
```

$$\text{Tan}\left[s + \frac{\pi}{x} + y\right]$$

Muitas funções nativas admitem argumentos facultativos em forma de **Rule[option, value]** ou **option → value**, que permitem passar parâmetros que alteram o comportamento típico da função. Os casos mais frequentes de utilização prendem-se com as opções de funções gráficas (como **Plot** ou **Plot3D**) que ajustam o número de pontos de amostragem ou o domínio de valores expostos.

$$V[r\_, q1\_, q2\_] := \frac{Q}{\sqrt{(r - q1).(r - q1)}} - \frac{Q}{\sqrt{(r - q2).(r - q2)}}$$

```
Plot3D[V[{x, y}, {-1, 0}, {1, 0}] /. Q → 2, {x, -3, 3},
    {y, -3, 3}, PlotPoints → 150, ClipFill → None, Mesh → False];
```



■ Funções definidas por Módulos

As funções nativas **Block[{vars}, expr$_1$; expr$_2$; ...]** e **Module[{vars}, expr$_1$; expr$_2$; ...]** permitem dar um nome a um grupo de definições **expr$_1$; expr$_2$; ...** que juntamente constituem uma rotina que deve ser executada quando se invoca esse nome. O resultado é o **Output** da última expressão avaliada na rotina.

As variáveis **vars** são em ambos os casos locais (i.e. não são afectadas nem afectam definições exteriores de símbolos com o mesmo nome). A diferença entre **Block** e **Module** consiste apenas em que a primeira guarda os valores de variáveis globais que tenham o mesmo nome que as suas variáveis locais, e depois de terminada a rotina volta a restaurar os valores dessas variáveis globais, enquanto que **Module** trabalha com variáveis novas **vars$nnn** diferentes cada vez que **Module** é invocado, não se dando assim ao trabalho de gerir conflitos de nomes.

```
x = 2;          (* x tem um valor global *)
```

```
f[y_] := Block[{t, x},
    z = y;
    Print["t=", t, "      ", "x=", x, "      ", "z=", z];
    t = y^2]
```

```
f[a]          (* Dentro de Block, x não tem valor; t=y^2 é local *)
```

```
t=t     x=x     z=a
```

```
a^2
```

```
(* Globalmente apenas z mudou de valor*)

Print["t=", t, "      ", "x=", x, "      ", "z=", z]
```

```
t=t     x=2     z=a
```

```
f[y_] := Module[{t, x},
        z = y;
        Print["t=", t, "       ", "x=", x, "       ", "z=", z];
        t = y^2];
```

```
f[a]   (* Em Module novos símbolos substituem x e t*)
```

```
t=t$347      x=x$347      z=a
```

```
a^2
```

```
Print["t=", t, "       ",
  "t$" <> ToString[$ModuleNumber – 1] <> "=",
  "t$" <> ToString[$ModuleNumber – 1] // ToExpression, "       ",
  "x=", x, "       ", "z=", z]
```

```
t=t      t$15=t$15      x=2      z=a
```

Gráficos, Animação e Som

O Mathematica dispõe de capacidades gráficas para gerar gráficos de funções a 2 D e 3 D, de densidade e de curvas de nível, bem como gráficos de listas discretas de valores. Pode–se ainda fazer animação de sequências de gráficos. Os gráficos são gerados num código .APS semelhante a Encapsulated PostScript mas mais compacto. A vantagem deste código é que é redimensionável sem perca de qualidade (ao contrário dos gráficos de tipo Bitmap que têm uma resolução fixa e denotam pixelização se aumentados). A desvantagem é ser pouco compacto e ter de ser interpre–tado para a sua apresentação (ing. **rendering**) o que se pode tornar pesado em termos de tempo e recursos de processador. O menu Cell ▸ ConvertTo ▸ Bitmap converte qualquer célula selec–cionada num gráfico de tipo Bitmap, o que aplicado às células gráficas representa uma redução muito significativa no tamanho final do Notebook para efeitos de armazenamento.

■ Graphics e Show

A função Graphics é usada para envolver um conjunto de instruções de desenho bi–dimensional designadas Primitivas Gráficas. (Para desenho a 3D usar–se–ia a função **Graphics3D** com primitivas semelhantes mas agora com coordenadas tri–dimensionais…) As seguintes primitivas gráficas são válidas para o desenho 2D (há mais...)

Point[{x, y}]	ponto na posição {x, y}
Line[{ {x_1, y_1}, …}]	linha definida por lista de posições {x_i, y_i}
Circle[{x, y}, r]	circunferência de raio r e centro {x, y}
Disk[{x, y}, r]	círculo cheio de raio r e centro {x, y}
Polygon[{ {x_1, y_1}, …}]	polígono cheio de vértices {x_i, y_i}
Text[expr, {x, y}]	texto centrado em {x, y}
PostScript["string"]	código Postscript dado por string

A maneira de visualizar uma expressão do tipo Graphics ou **Graphics3D** é usando a função Show[{graphics}, options]. A função Show permite combinar uma lista de gráficos num só, criando uma sequência de opções escolhidas de entre as opções dos gráficos que se quer mostrar juntos.

Internamente os gráficos são armazenados como qualquer outra expressão simbólica do *Mathematica, por isso podem ser referidos posteriormente por um nome que se lhe associe (existe sempre o de* **Out[n]**, onde n é número da expressão de **Input** que gerou o gráfico). Uma vez gerado, um gráfico pode ser apresentado outra vez em qualquer altura numa sessão (sem necessidade de re−avaliar a expressão que lhe deu origem), só ou em sobreposição com outros gráficos usando o comando Show[{$graph_1$, $graph_2$, …}, options].

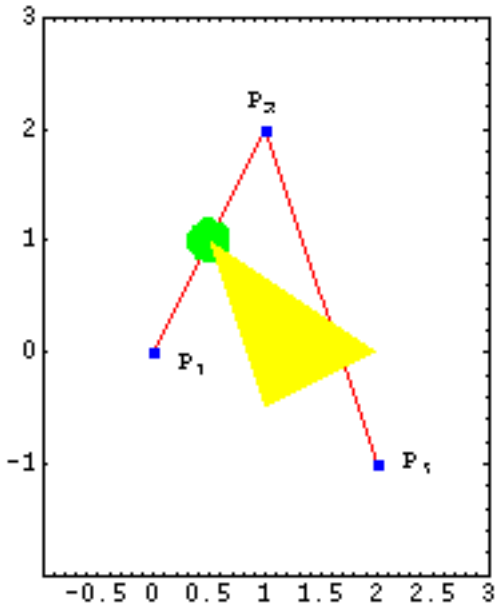
Embora haja vários tipos de gráficos gerados por diversos comandos do *Mathematica*, por exemplo Graphics, GraphicsArray, ContourGraphics, DensityGraphics, **SurfaceGraphics**, todos podem ser convertidos ao tipo **Graphics** ou Graphics3D conforme sejam 2 D ou 3 D.

Note−se que as células de tipo gráfico são distintas das células de tipo O**utput**, as quais são inscritas no Notebook após as respectivas células de graficos. A interpretação e apresentação de gráficos num Notebook (ing. **rendering**) é uma tarefa do FrontEnd enquanto ao Kernel compete calcular os elementos gráficos que envia para o FrontEnd. Para gráficos complexos é frequente ver o FrontEnd começar a apresentar o gráfico enquanto o Kernel ainda se encontra a enviar dados deste para o FrontEnd.

Os comandos gráficos do Mathematica podem−se classificar em dois tipos gerais:

(a) – Aqueles que desenham pontos (**Point[coords]**), linhas (**Line[{pt1, pt2, ...}]**), polígonos (**Polygon[{pt1, pt2, ...}]**), texto (**Text[expr, coords]**) e outras primitivas gráficas dadas explícitamente pelo utilizador ou geradas por código. São da forma geral **Graphics[{primitivas}, opções]** ou **Graphics3D[{primitivas}, opções]**. (consulte Help ▸ Help Browser ▸ Graphics Primitives para uma lista completa) Para ver estes gráficos (e de facto para ver quaisquer objectos gráficos) tem que se usar o comando **Show**[Graphics[{primitivas}, opções de Graphics], opções de Show].

```
pt1 = {0, 0};
pt2 = {1, 2};
pt3 = {2, −1};
gr0 = Show[Graphics[{
        Red, Line[{pt1, pt2, pt3}],
        Blue, PointSize[.03], Point[pt1], Point[pt2], Point[pt3],
        Black, Text["P1", pt1, {−2, 1}], Text["P2", pt2, {0, −2}], Text["P3", pt3, {−2, 0}],
        Green, Disk[1/2 pt2, .2],
        Yellow, Polygon[{{2, 0} + pt1, 1/2 pt2, 1/2 pt3, {2, 0} + pt1}]
            }, PlotRange → {{−1, 3}, {−2, 3}}],
      AspectRatio → 5/4, Frame → True]
```



- Graphics -

(b) – Aqueles que representam o comportamento de uma ou mais funções dependentes de variáveis com valores em intervalos finitos, contínuos ou discretos. A sua sintaxe é sempre na forma de plotfunction[{funções de x}, {x, $\min_x$, $\max_x$}, opções] ou ainda plotfunction3D[{funções de x e y}, {x, $\min_x$, $\max_x$}, {y, $\min_y$, $\max_y$}, opções] onde x e y são símbolos arbitrários. Para funções discretas dadas por listas ou matrizes de valores usa–se **plotlist[list, opções]** ou **plotarray[array, opções]**.

O nome das funções **plotfunction, plotfunction3D, plotlist** ou **plotarray** pode ser um de entre os comandos nativos:

Plot	**ListPlot**	
Plot3D	**ListPlot3D**	
ParametricPlot		
ParametricPlot3D		
DensityPlot	**ListDensityPlot**	**ArrayPlot**
ContourPlot	**ListContourPlot**	

ou definido por um dos módulos gráficos adicionais que estendem ou definem novas funcionalidades gráficas deste tipo, como por exemplo os módulos:

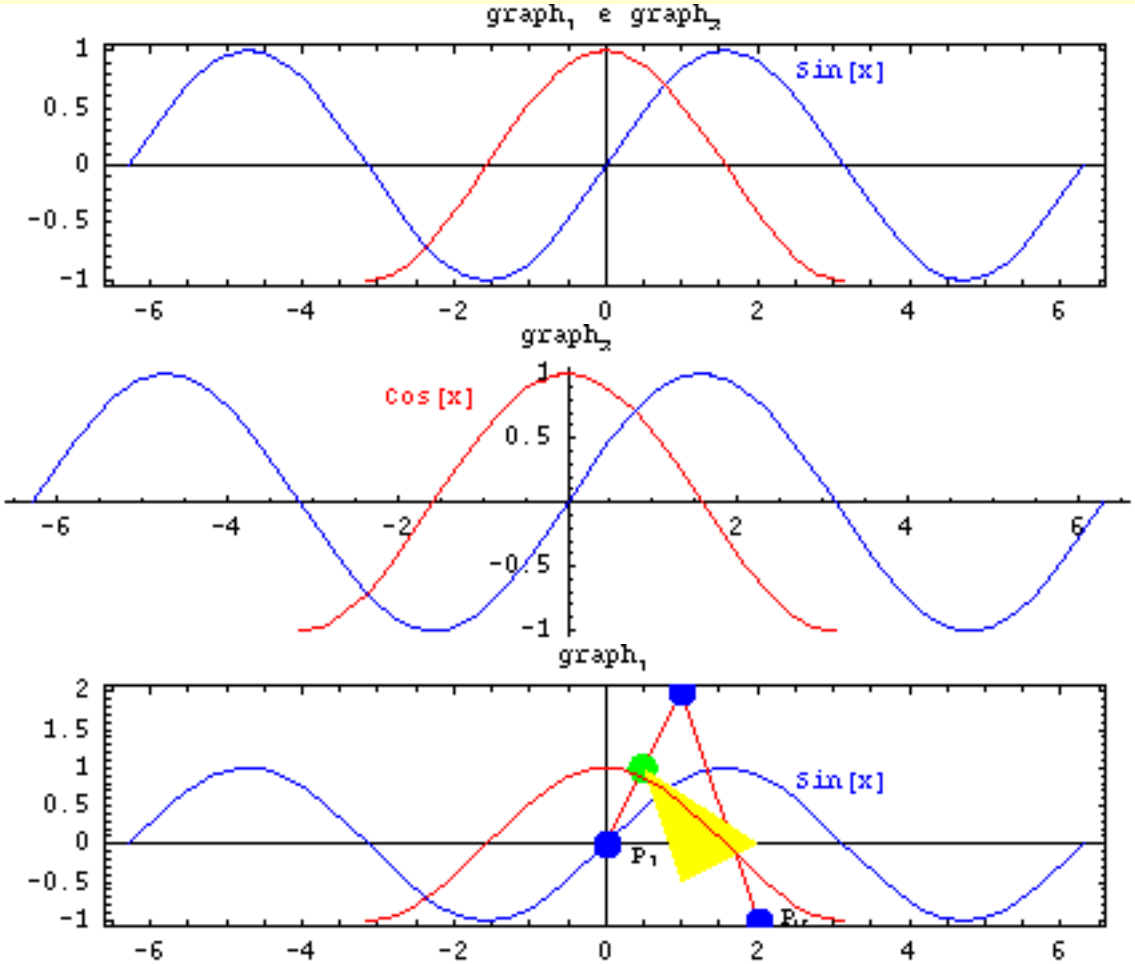
Graphics‘Graphics‘	**Graphics‘Graphics3D‘**	**Graphics‘InequalityGraphics‘**
Graphics‘FilledPlot‘	**Graphics‘ContourPlot3D‘**	**Graphics‘ImplicitPlot**
Graphics‘MultipleListPlot‘	**Graphics‘PlotField‘**	**Graphics‘PlotField3D‘**

□ EPILOG

Mais uma vez estes tipos de gráficos podem ser sobrepostos usando **Show** desde que sejam da mesma dimensionalidade (2 D ou 3 D), mas em qualquer caso é sempre possível especificar primitivas gráficas num gráfico de tipo (b) usando a opção Epilog → {lista de primitivas gráficas 2 D} que desenha sobre a janela bidimensional do gráfico 2 D ou 3 D (ou **Prolog** → {lista de primitivas gráficas 2 D} que fica subjacente ao gráfico).

A desvantagem da opção Epilog (ou **Prolog**) é que pode não ser preservada quando o gráfico em questão é sobreposto a outros com a função **Show**[{graph$_1$, graph$_2$, ...}, options]. De facto **Show** não pode respeitar simultâneamente todas as opções dos vários gráficos, pelo que em caso de conflito usa as opções do primeiro gráfico da lista {graph$_1$, graph$_2$, ...} que as especifica, ou usa opções que forem declaradas explícitamente na **options** de **Show**.

```
gr1 = Plot[Sin[x], {x, -2 π, 2 π}, PlotLabel → "graph1", Frame → True, PlotStyle → Blue,
    Epilog → {Text["Sin[x]", {π, 0.8}]}, DisplayFunction → Identity];
gr2 = Plot[Cos[x], {x, -π, π}, PlotLabel → "graph2", Frame → False, PlotStyle → Red,
    Epilog → {Text["Cos[x]", {-π/2, 0.8}]}, DisplayFunction → Identity];
Show[{gr1, gr2}, DisplayFunction → $DisplayFunction,
    AspectRatio → 3/(4 π), PlotLabel → "graph1 e graph2"];
Show[{gr2, gr1}, DisplayFunction → $DisplayFunction, AspectRatio → 3/(4 π)];
Show[{gr1, gr0, gr2}, DisplayFunction → $DisplayFunction, AspectRatio → 3/(4 π)];
```



Note que a linguagem PostScript não suporta transparências, pelo que é necessário recorrer a sobreposições de elementos gráficos pela ordem correcta para garantir a sua visibilidade. De facto a característica de ter um canal **alpha** para descrição de cores é uma propriedade de descrições Bitmap de imagens, enquanto o PostScript é uma descrição vectorial das imagens (podendo contudo conter imagens Bitmap inseridas no código).

Nos comandos anteriores a opção **AspectRatio** → $\frac{\text{altura}}{\text{comprimento}}$ é estipulada de forma a garantir a equidimensionalidade dos intervalos na horizontal e na vertical. Só assim se consegue que o disco não fique distorcido.

Embora existam funções para alterar o estilo do texto em gráficos, os comandos **Text** reflectem as alterações à propriedades do texto inserido como String no seu primeiro argumento. Note que todos os gráficos usam a variável de sistema $DefaultFont para o tipo de letra de qualquer texto que não esteja explícitamente formatado com outro estilo. Em geral $DefaultFont = {" Courier ", 10} designado o nome e tamanho do tipo de letra (ing. **Font**) usado por norma nos gráficos. Esta variável pode ser alterada para outro tipo e tamanho, por exemplo $DefaultFont = {" Times – Bold ", 9}.

□ A Opção DisplayFunction

De todas as opções gráficas existe uma designada DisplayFunction que permite controlar para onde é encaminhado o resultado gráfico. As possibilidades são:

DisplayFunction → (Display[{"stdout"}, #1] &)	gráfico é mostrado no FrontEnd
DisplayFunction → Identity	gráfico é gerado mas não é mostrado
DisplayFunction → (Display["channel", #, "fmt"] &)	gráfico é reencaminhado para "channel" no formato "fmt"

A variável de sistema **$DisplayFunction** armazena informação sobre o canal para que deve enviar o resultado de cálculos de gráficos. Por norma **$DisplayFunction** envia os resultados gráficos para $Display = {" stdout "} num formato que o FrontEnd sabe interpretar, mas é possí–vel modificar **$DisplayFunction** para enviar os gráficos para outros destinos, em particular documentos gráficos noutro formato ou comandos exteriores.

A opção **DisplayFunction** → Identity garante que o FrontEnd não desenha o gráfico embora o Kernel o tenha gerado. Quando se pretender vizualizar estes gráficos com **Show** é necessário explicitar a opção DisplayFunction → > $DisplayFunction, onde a variável de sistema $DisplayFunction = Display[$Display, #1] & representa a função que normalmente redirecciona os elementos gráficos para o FrontEnd (onde $Display = {" stdout "}).

□ Manuseamento de Célula Gráficas

Os gráficos do Mathematica são redimensionáveis usando a moldura que aparece à sua volta quando são seleccionados (i.e. quando se pressiona o cursor sobre o gráfico). Nessa moldura há oito pontos (nos cantos e no meio das arestas) em cima dos quais o cursor muda de forma, indi–cando o tipo de redimensionamento que proporcionam. A vantagem de ter os gráficos descritos numa linguagem vectorial é que o redimensionamento da imagem não afecta a resolução, ao contrário do que acontece numa imagem do tipo Bitmap.

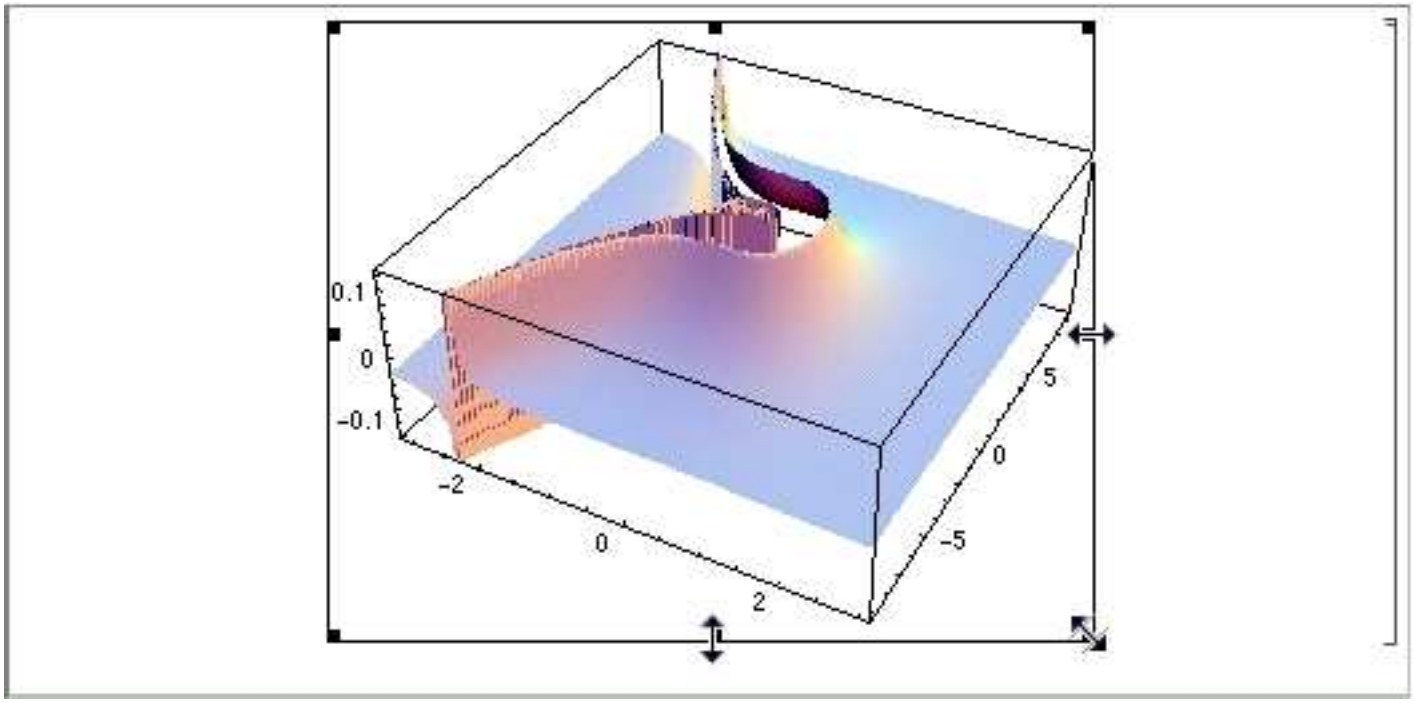


Figura 10

□ SHOW[GRAPHICS] E SHOW[GRAPHICSARRAY]

O *Mathematica* pode ainda importar imagens de documentos externos usando o comando **Import["*file*", *Type*]** convertendo–os automáticamente em objectos do tipo **Graphics**, onde " *file* " representa uma imagem do tipo ***Type*** reconhecível pelo *Mathematica*.

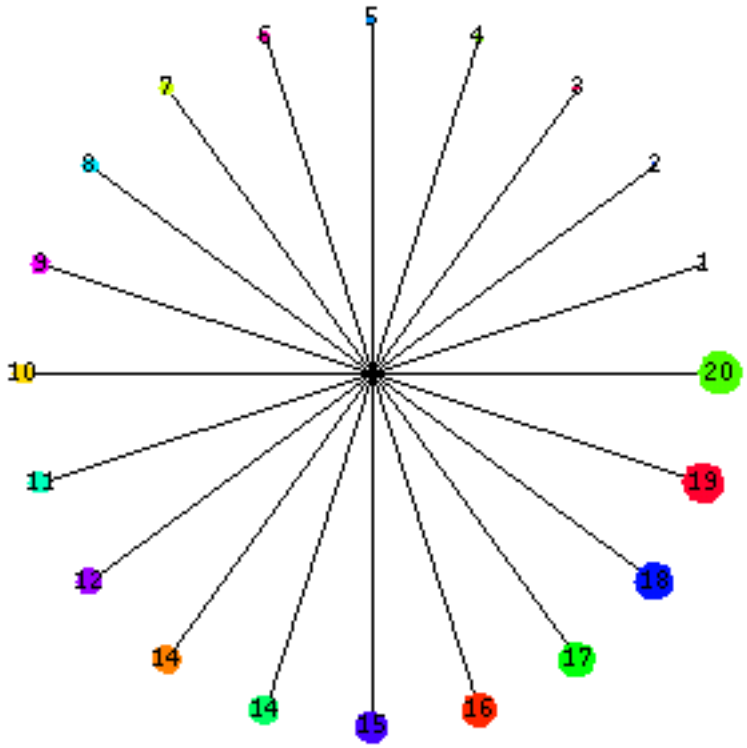
O comando Show[GraphicsArray[matriz de gráficos]] permite combinar e apresentar uma matriz de m × n gráficos num quadro de m linhas e n colunas numa só célula gráfica.

```
g1 = Import["01–10–05_1452.jpg", "JPEG"];
g2 = Import["01–10–05_1453.jpg", "JPEG"];
Show[GraphicsArray[{g1, g2}, Frame → True]]
```

- GraphicsArray -

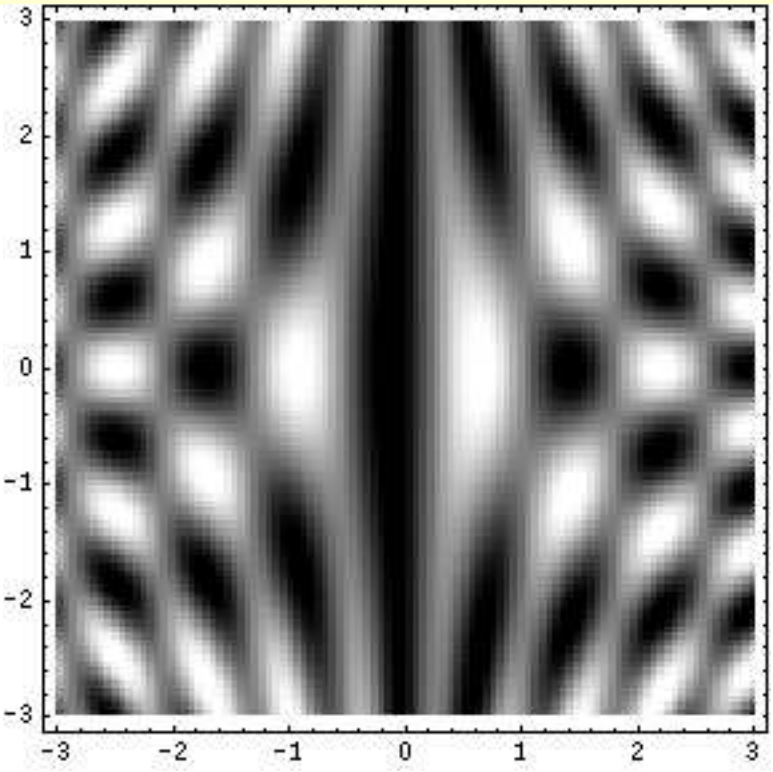
Por exemplo, a seguinte instrução usa uma expressão Graphics para gerar uma lista de linhas e discos dependentes dum parâmetro que se faz variar de 0 a 2π em passos de $\frac{\pi}{10}$:

```
Show[{Graphics[{Line[{{0, 0}, {Cos[#], Sin[#]}}], {Hue[#], Disk[{Cos[#], Sin[#]}, #/100]},
      Text[Ceiling[10 #1/π], {Cos[#], Sin[#]}]}, AspectRatio → 1] & /@ Range[0, 2π, .1π]}];
```



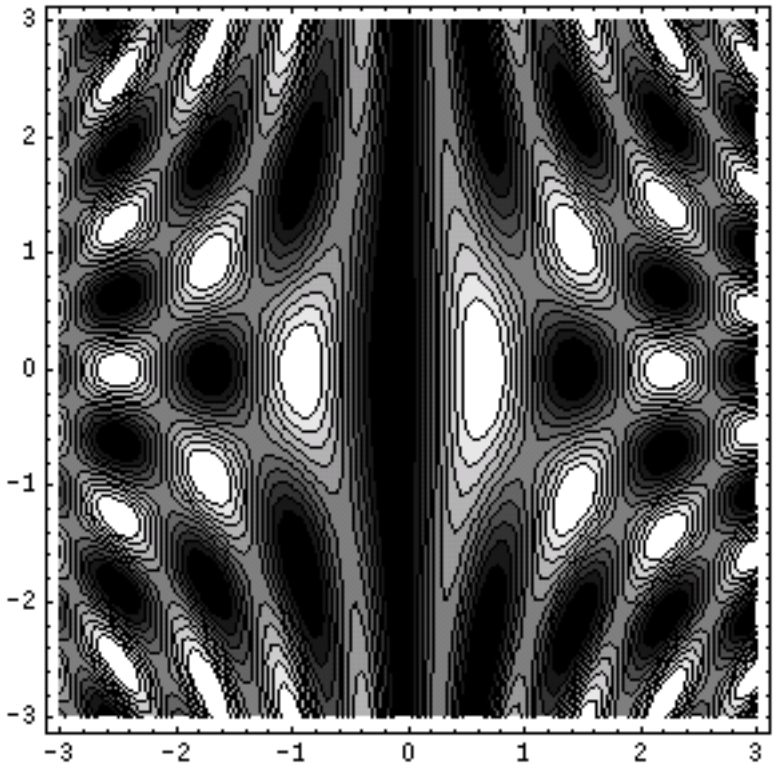
Outras funções como DensityPlot e ContourPlot que criam expressões de tipo DensityGraphics e ContourGraphics podem–se combinar com Graphics em Show.

```
DensityPlot[Sin[4 x – 1] Cos[2 y x], {x, –3, 3}, {y, –3, 3},
  Mesh → False, PlotPoints → 100, PlotRange → {–0.9, 0.9}]
```



- DensityGraphics -

```
ContourPlot[Sin[4 x – 1] Cos[2 y x], {x, –3, 3}, {y, –3, 3}, PlotPoints → 100, PlotRange → {–0.9, 0.9}]
```



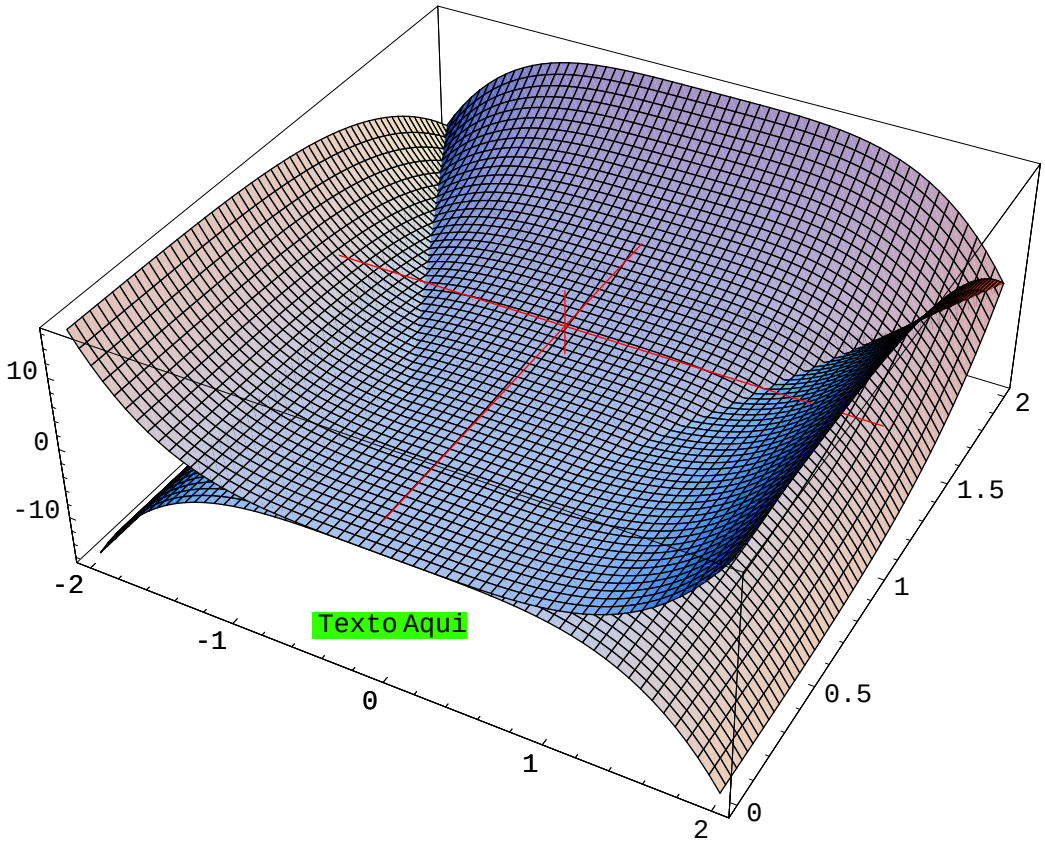
- ContourGraphics -

De forma semelhante se pode usar Graphics3D com o seu conjunto de primitivas gráficas próprio:

Point[{x, y, z}]	ponto na posição cartesiana {x, y, z}
Line[{ {x_1, y_1, z_1}, ...}]	linha unindo posições cartesianas {x_i, y_i, z_i}
Polygon[{ {x_1, y_1, z_1}, ...}]	polígono cheio com vértices {x_i, y_i, z_i}
Text[expr, {x, y, z}]	Expressão escrita na posição indicada

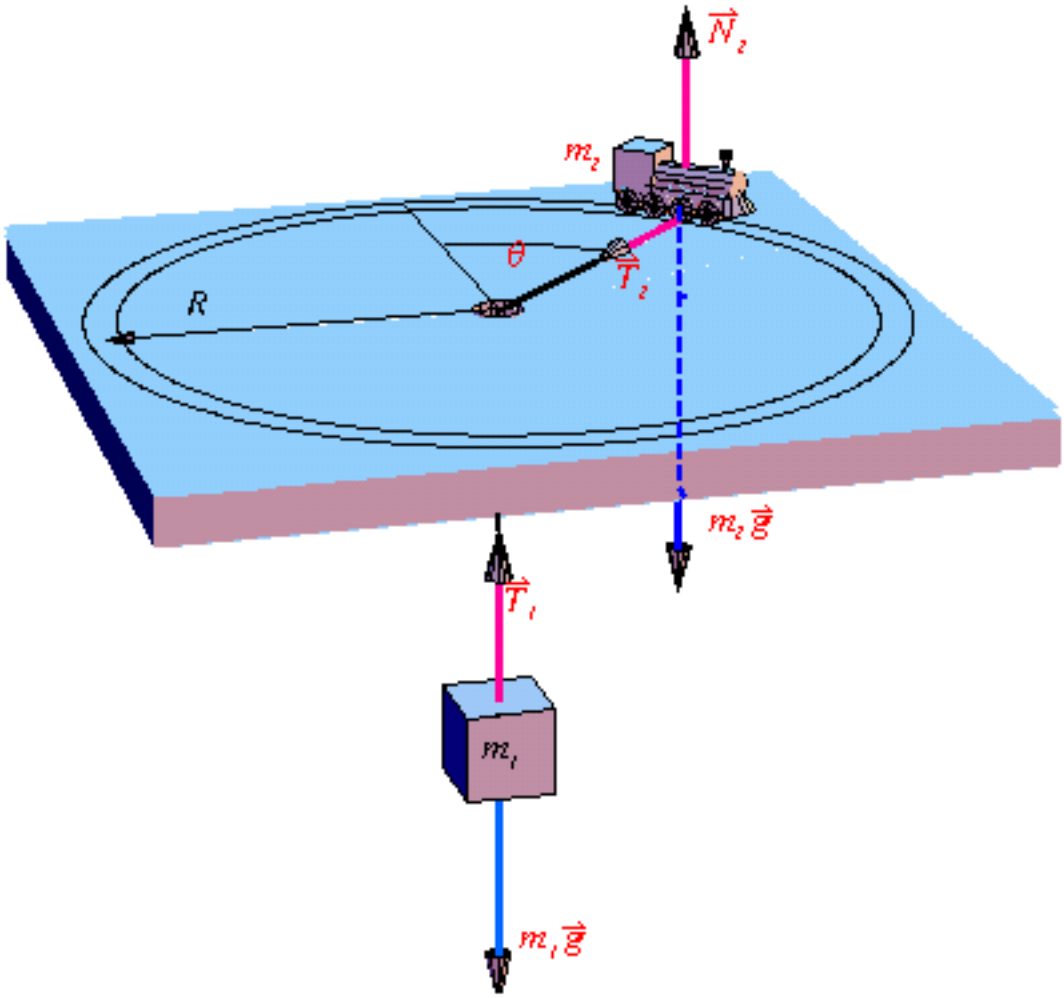
No exemplo seguinte juntamos o SurfaceGraphics anteriormente gerado com o nome gr23 e um Graphics3D contendo apenas texto e linhas.

```
Show[{gr23, Graphics3D[{Text["Texto Aqui", {0, 0, −10}, Background → Hue[.3]], Hue[1],
      Line[{{0, 1, −10}, {0, 1, 10}}], Line[{{−2, 1, 5}, {2, 1, 5}}], Line[{{0, 0, 5}, {0, 2, 5}}]}]},
 DisplayFunction → $DisplayFunction]
```



- Graphics3D -

Com alguma experiência é possível construir diferentes elementos gráficos separadamente e juntá–los num gráfico final, como no exemplo seguinte de um objecto **Graphics3D**. A vantagem desta abordagem é que a correcção dos elementos gráficos mais complexos pode ser feita separada–mente.



■ Funções que geram Gráficos 2D

Plot

Plot[f[x], {x, x_{min}, x_{max}}, *opções*] é o comando básico para gerar gráficos de funções escalares reais f[x], com $x \in [x_{min}, x_{max}]$ onde x pode ser qualquer símbolo e onde opções é uma sequência ou lista (possívelmente nula) de regras propr → valor separadas por vírgula. O conjunto de opções disponíveis para Plot pode ser obtido executando o comando Options[Plot]. As opções em geral são todas omitidas, e apenas as que forem explícitamente especificadas alteram a respectiva propriedade para o gráfico em questão.

```
Options[Plot] // Partition[#, 15] & // Transpose // TableForm
```

AspectRatio → $\frac{1}{\text{GoldenRatio}}$	FrameStyle → Automatic
Axes → Automatic	FrameTicks → Automatic
AxesLabel → None	GridLines → None
AxesOrigin → Automatic	ImageSize → Automatic
AxesStyle → Automatic	MaxBend → 10.
Background → Automatic	PlotDivision → 30.
ColorOutput → Automatic	PlotLabel → None
Compiled → True	PlotPoints → 25
DefaultColor → Automatic	PlotRange → Automatic
DefaultFont :→ $DefaultFont	PlotRegion → Automatic
DisplayFunction :→ $DisplayFunction	PlotStyle → Automatic
Epilog → {}	Prolog → {}
FormatType :→ $FormatType	RotateLabel → True
Frame → False	TextStyle :→ $TextStyle
FrameLabel → None	Ticks → Automatic

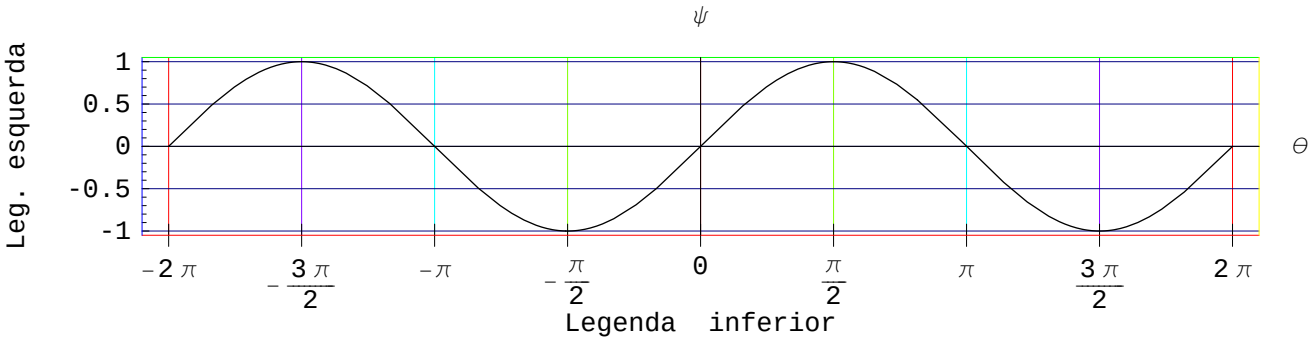
O utilizador pode modificar explícitamente uma ou mais destas regras alterando o valor de cada propriedade, mas enquanto alguns valores como o de PlotPoints → 25 têm significado evidente, já outras como as que usam o valor Automatic devem ser alteradas com conhecimento do formato adequado. Use o Help Browser ... para obter informação sobre os formatos admitidos para cada propriedade.

Se pretender uma alteração mais permanente de alguma opção de **Plot**, deve usar o comando SetOptions[Plot, propriedade → valor] que se manterá válido para todos os posteriores comandos Plot durante toda a sessão do Kernel até ser outra vez alterado. Por exemplo **SetOptions[Plot, PlotPoints → 50]** altera a resolução de todos os comandos **Plot** posteriores para dividirem o seu domínio em pelo menos 50 pontos.

Uma forma de memorizar estas alterações para outras sessões de trabalho no mesmo Notebook consiste em catalogar de **Initialization Cell** as células onde as alterações pretendidas são escritas.

O resultado de Plot é um objecto do tipo Graphics[lines, options] que pode ser atribuído a um símbolo para futura utilização.

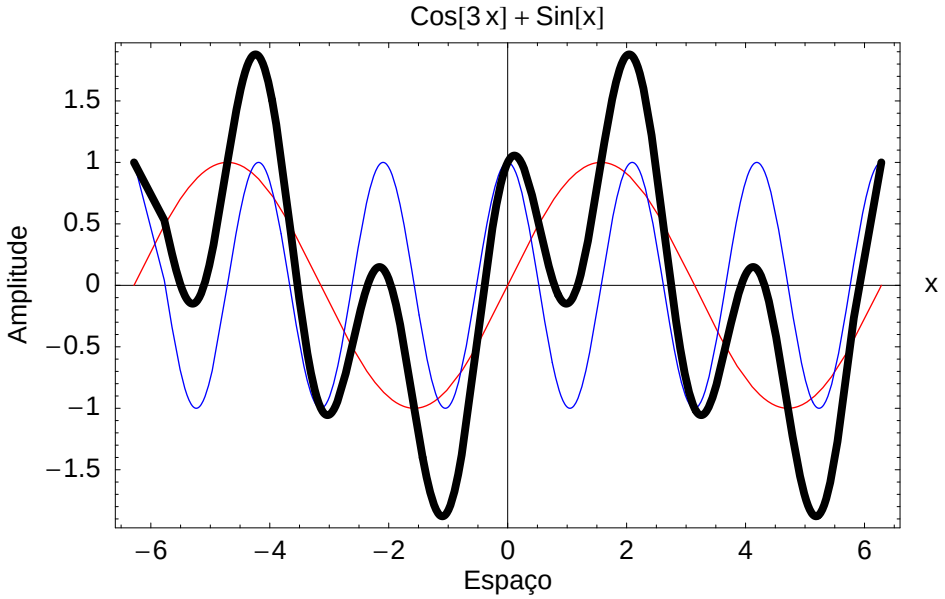
```
gr = Plot[Sin[x], {x, -2 π, 2 π},
    AspectRatio → 2/(4 π), Frame → True, AxesLabel → {"θ", "ψ"},
    FrameLabel → {"Legenda  inferior", "Leg. esquerda\n", None, None},
    FrameStyle → {{Red}, {Blue}, {Green}, {Yellow}},
    FrameTicks → {{# π/2, # π/2, .02} & /@ Range[-4, 4], Automatic, None, None},
    GridLines → {{# π/2, {Hue[Abs@#/4.]}} & /@ Range[-4, 4], Range[-1, 1, .5]}]
```



Plot gera uma expressão de tipo Graphics, e por isso é possível sobrepôr gráficos de funções em domínios diferentes fazendo diferentes atribuições gr[i] = Plot[f_i[x], {x, … }] e mostrando–os juntos com Show[{gr[1], …, gr[n]}]. Se os domínios forem comuns, e se em vez de f[x] estiver uma lista de funções reais {f_1[x], f_2[x], …}, todas dependentes do parâmetro x, a função Plot[{f_1[x], f_2[x], …}, {x, x_{min}, x_{max}}] sobrepõe os gráficos das diferentes funções f_i[x] no mesmo domínio {x, x_{min}, x_{max}}.

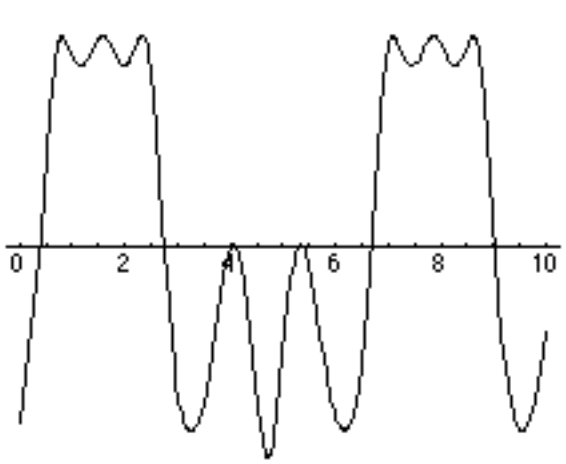
```
$DefaultFont = {"Helvetica", 10};
```

```
Plot[{Sin[x], Cos[3 x], Cos[3 x] + Sin[x]}, {x, -2 π, 2 π},
                    PlotStyle → {Red, Blue, {Thickness[0.01‘], Black}},
                    Frame → True,
                    PlotLabel → Cos[3 x] + Sin[x],
                    AxesLabel → {x, None},
                    FrameLabel → {Espaço, Amplitude}
          ];
```

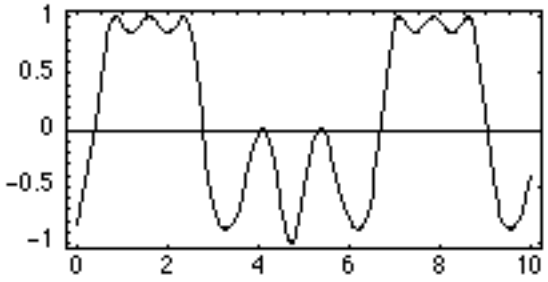


□ Opções de Plot

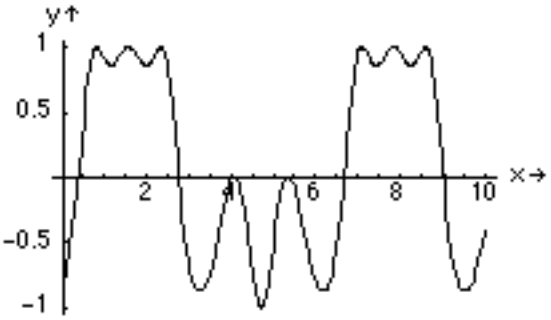
Os seguintes gráficos mostram o efeito de usar diferentes opções em Plot, evidenciadas pelo texto de fundo amarelo.



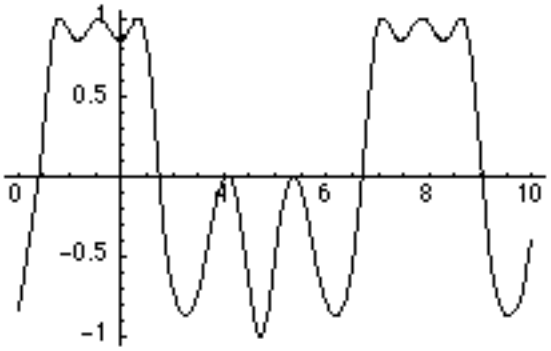
AspectRatio→ 0.8
Axes→{True, False}



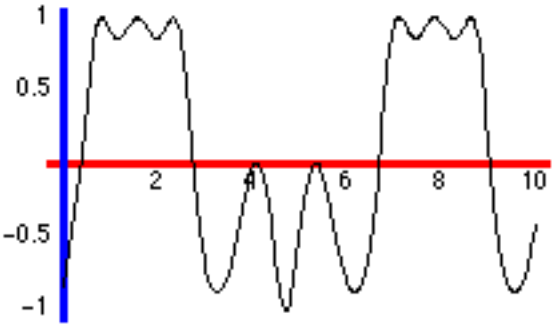
AspectRatio→ 0.5
Frame → True



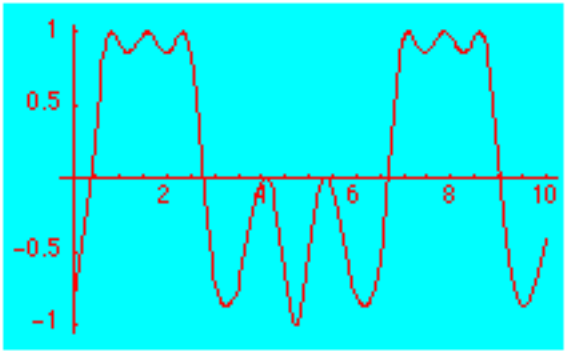
AxesLabel→{x→, y↑}



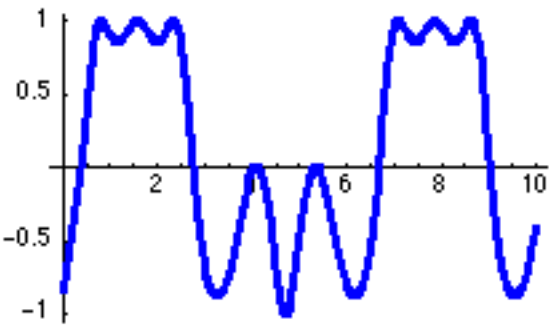
AxesOrigin→{2, 0}



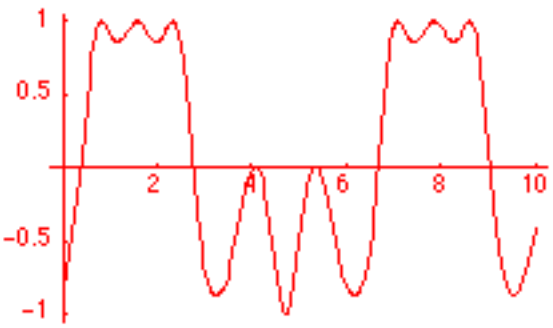
AxesStyle→ $\begin{pmatrix}\text{Thickness[0.02]} & \text{Red}\\ \text{Thickness[0.02]} & \text{Blue}\end{pmatrix}$



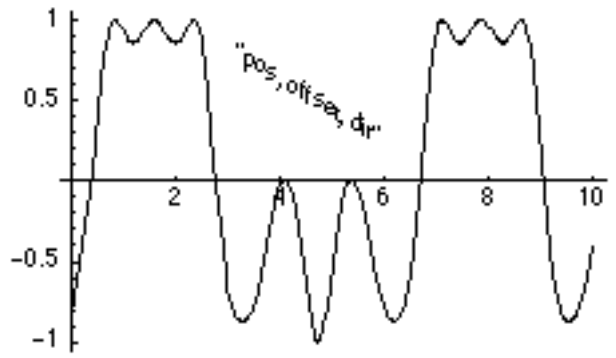
Background→ RGBColor[0, 1, 1]



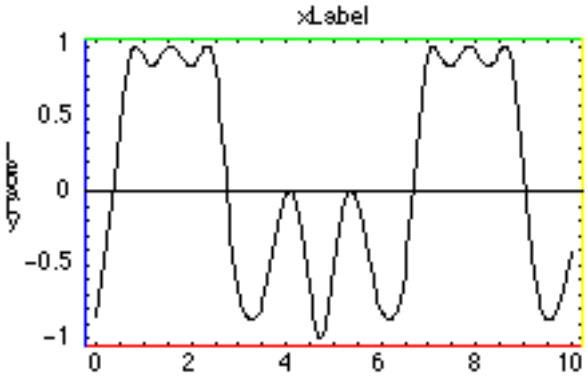
PlotStyle→ {Thickness[0.02], Blue}



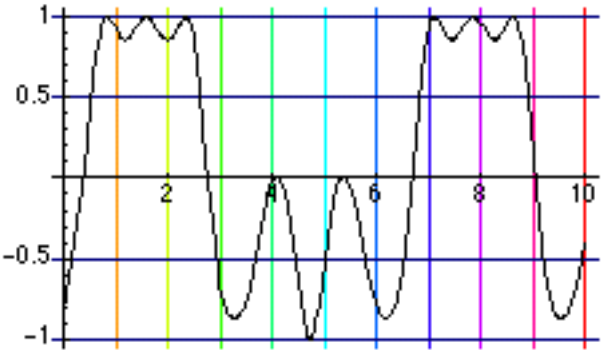
DefaultColor→ Red



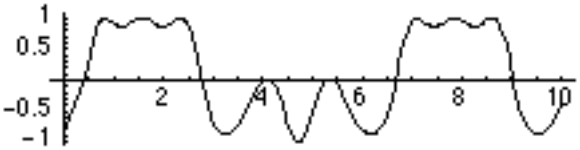
Epilog → Text["pos, offset, dir", {3, $\frac{1}{2}$}, {-1, 0}, {$\frac{1}{2}$, $-\frac{3}{10}$}]



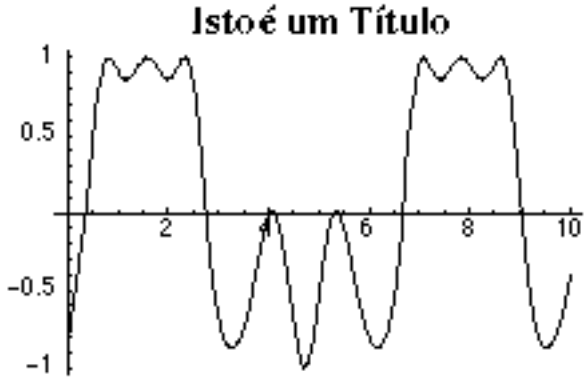
Frame → True
FrameLabel → {None, yLabel, xLabel, None}
FrameStyle → {{Red}, {Blue}, {Green}, {Yellow}}



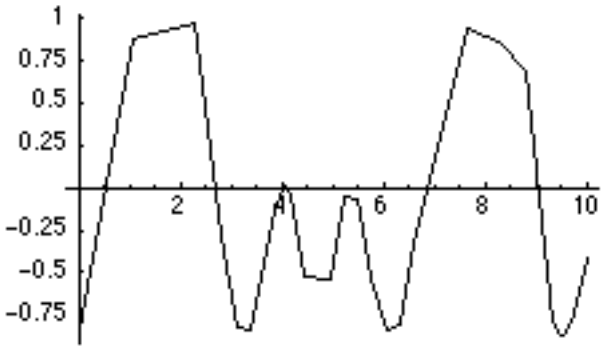
GridLines → {Array[{#1, {Hue[$\frac{\#1}{10}$]}}&, 10], Automatic}



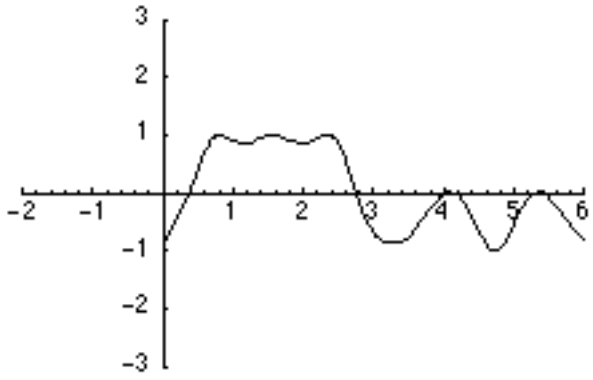
ImageSize → {200, 50}
AspectRatio → $\frac{1}{4}$



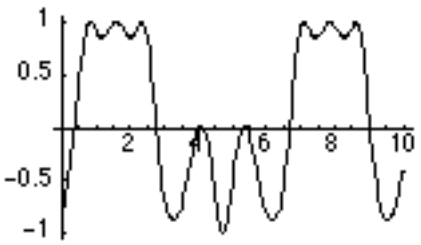
PlotLabel → Istoé um Título



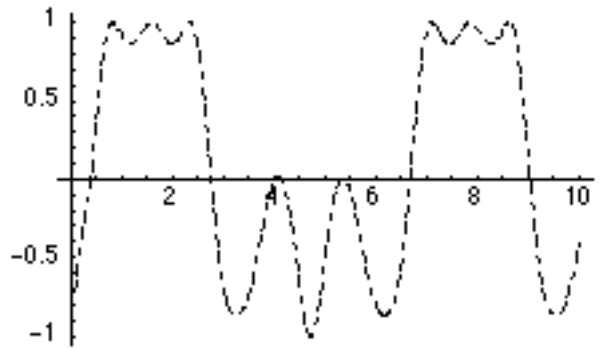
PlotPoints → 10
MaxBend → 100



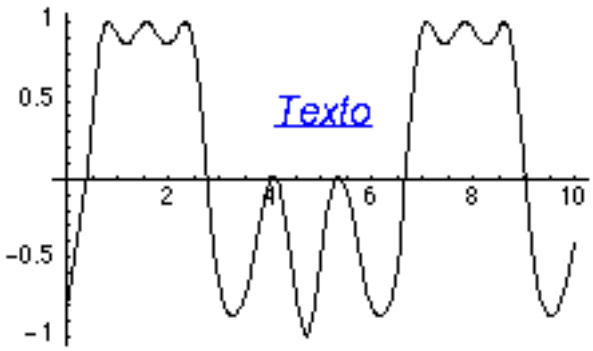
PlotRange → {{-2, 6}, {-3, 3}}



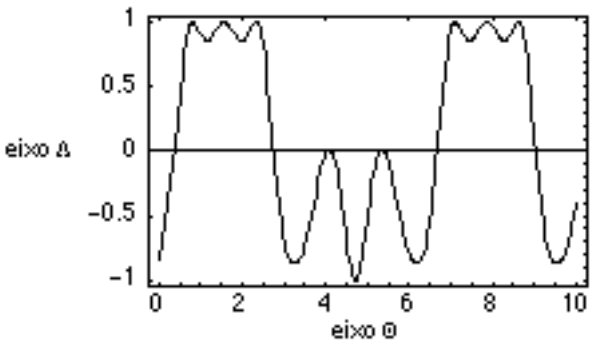
PlotRegion → {{0.3, 1}, {0.3, 1}}



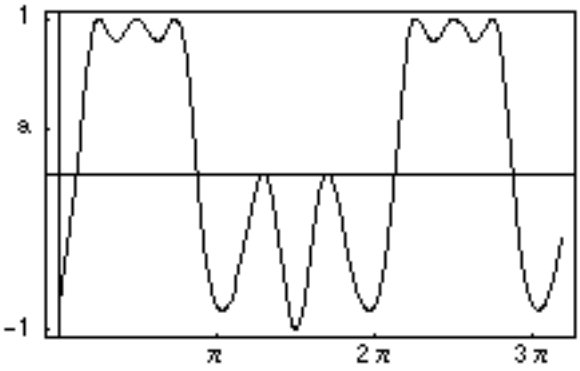
PlotStyle→ Dashing[{0.02, 0.01, 0.02}]



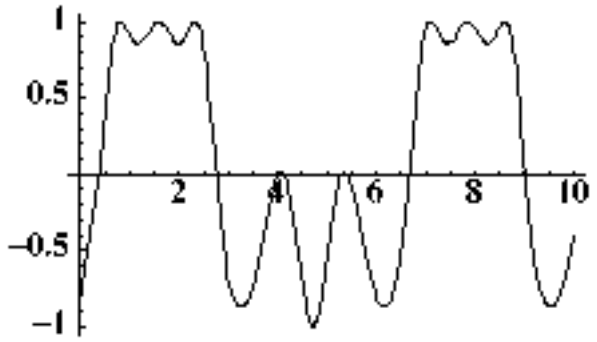
Prolog → Text[Texto, {4, 0.4}, {-1, 0}]



Frame → True
FrameLabel → {eixo @, eixo A}
RotateLabel → False



Frame → True
Axes → True
FrameTicks → {Array[#1 π&, 3], {-1, {0.3, a}, 1}, {}, {}}

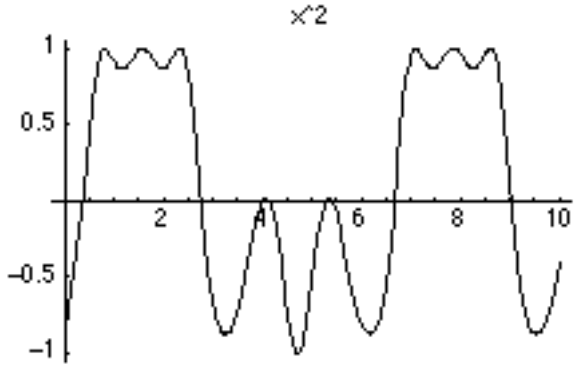


DefaultFont :→{Times-Bold, 12.}

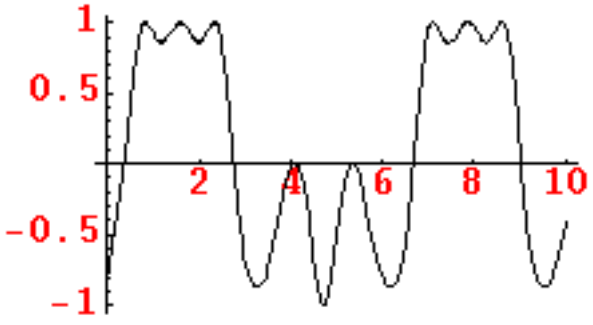
DisplayFunction→Display["file.gif",#]&)

Este gráfico está em file.gif

Axes → None



PlotLabel → x²
FormatType :→ InputForm



TextStyle:→{Section, FontColor → Red}

O código para gerar este conjunto de gráficos está aqui.

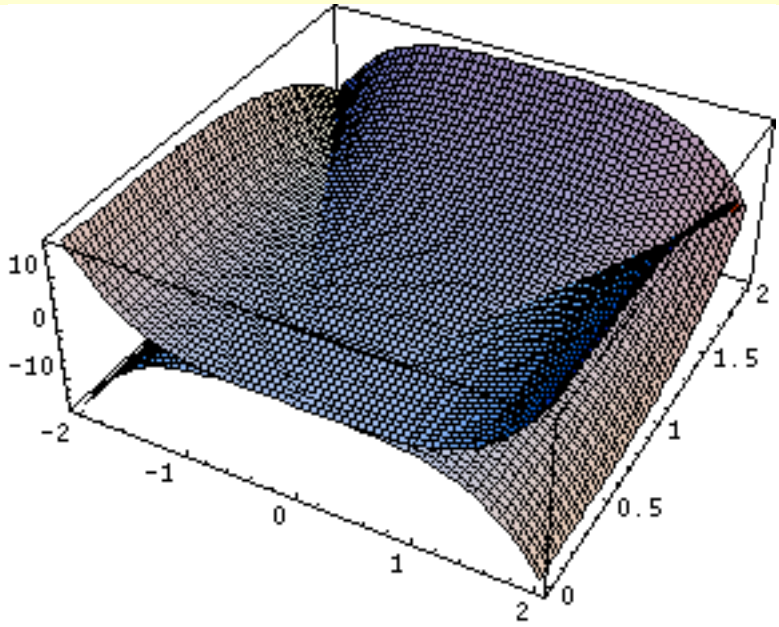
■ Plot3D

Com Plot3D o gráfico de uma função real de dois parâmetros pode ser visto, sendo o resultado uma expressão de cabeça SurfaceGraphics. Ao contrário de Plot, esta não pode mostrar várias funções ao mesmo tempo, mas é possível combinar vários SurfaceGraphics num só usando Show[{ lista de gráficos}, opções]

```
gr2 = Plot3D[x^4 - y^4, {x, -2, 2}, {y, 0, 2}, PlotPoints → 60, DisplayFunction → Identity];
```

```
gr3 = Plot3D[ - (x^4 - y^4), {x, -2, 2}, {y, 0, 2}, PlotPoints → 60, DisplayFunction → Identity];
```

```
gr23 = Show[{gr2, gr3}, DisplayFunction → $DisplayFunction];
```



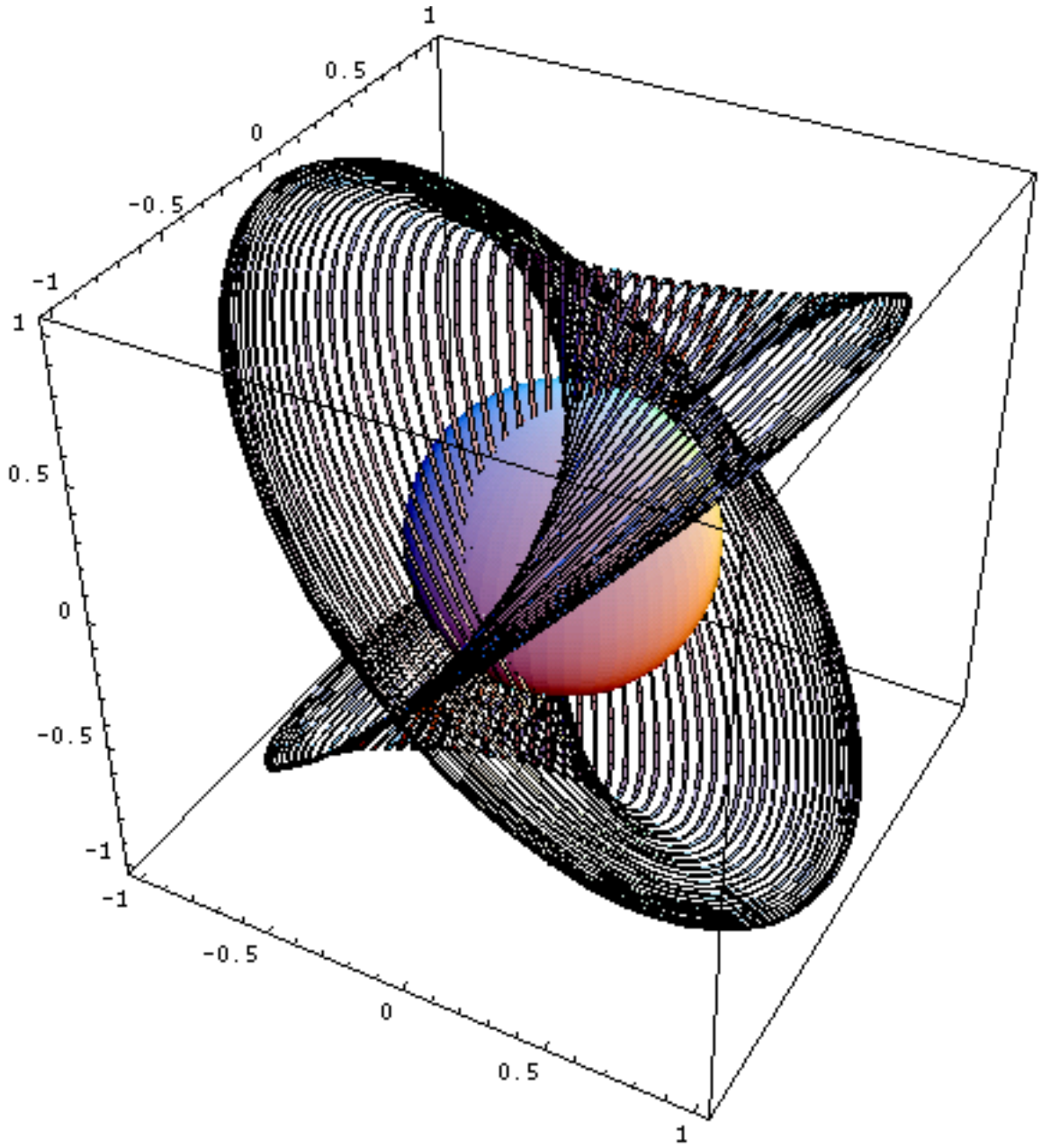
■ ParametricPlot e ParametricPlot3D

Para obter gráficos de representações paramétricas de curvas e superfícies podem–se usar as funções nativas ParametricPlot e ParametricPlot3D.

```
ParametricPlot[{Sin[θ], Cos[6 θ + π / 6]}, {θ, 0, # / 30 π},
        PlotPoints → 60, PlotRange → {{−1, 1}, {−1, 1}}] & /@ Range[60];
```

```
gr0 = ParametricPlot3D[{Sin[y] Cos[x], Sin[x] Sin[y], Cos[x]},
                        {x, 0, 2 Pi}, {y, 0, 2 Pi},
                        BoxRatios → {1, 1, 1},
                        DisplayFunction → Identity, PlotPoints → 60];
gr1 = ParametricPlot3D[.5 {Sin[y] Cos[x], Sin[x] Sin[y], Cos[y]},
                        {x, 0, 2 π}, {y, 0, π},
                        BoxRatios → {1, 1, 1},
                        DisplayFunction → Identity, PlotPoints → 60];
```

```
Show[{gr1 /. Polygon[z_] :→ {EdgeForm[], Polygon[z]},
      gr0 /. Polygon[z_] :→ Polygon[z + .43 (Reverse[z] − z) + (RotateRight[Reverse[z], 2] − z) 0]},
      DisplayFunction → $DisplayFunction];
```



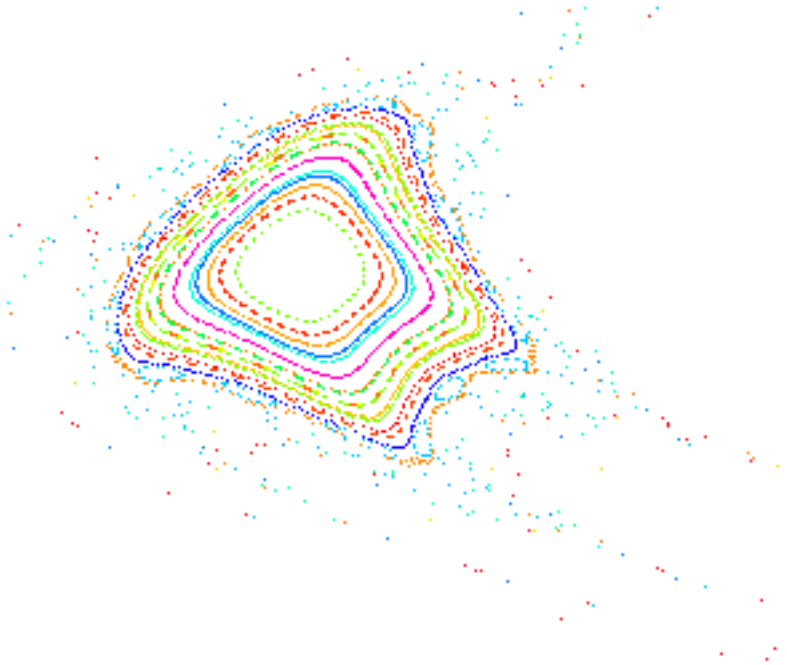
■ ListPlot e ListPlot3D

A representação gráfica de conjuntos discretos também é possível usando as funções ListPlot e ListPlot3D

```
henon[n_, m_, α_] := Block[{},
    Xmin = -4.;          Xmax = 4.;
    Ymin = -3.;          Ymax = 3.;
    points = {};         OVF = 10^10;
    a = Cos[2. π α];         b = Sin[2. π α];

    For[k = 0, k ≤ m, k++,          xo = Random[];
      yo = Random[]; AppendTo[points, Hue[Random[]]];
      For[j = 0, j ≤ n, j++,          x1 = a xo + b yo + b xo^2;          y1 = -b xo + a yo + a xo^2;
        If[x1^2 < OVF && y1^2 < OVF,          xo = x1;          yo = y1;
                                  AppendTo[points, Point[{x1, y1}]], Break]
          ]
        ]
      ]
```

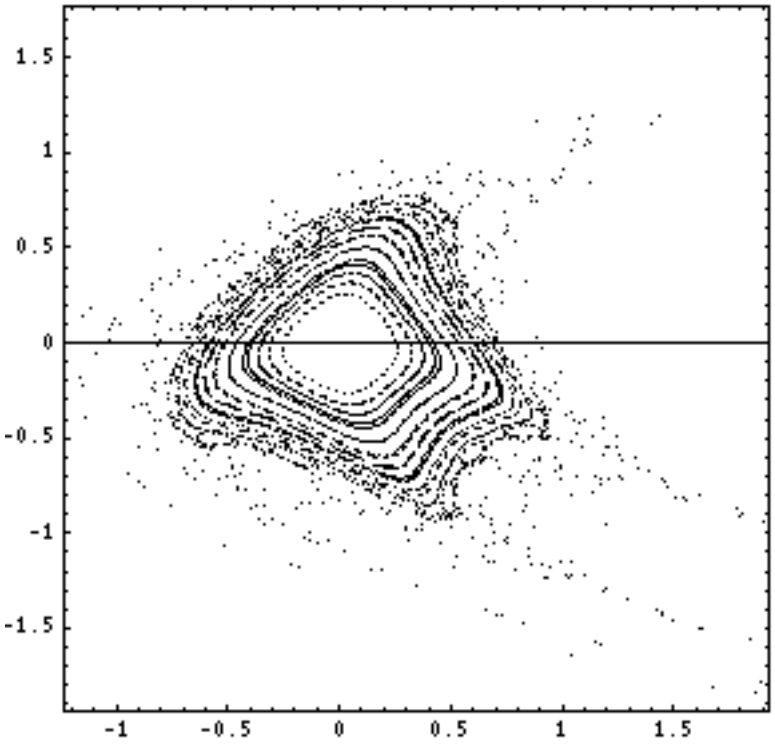
```
henon[2^8, 34, .245104]
```

```
Show[Graphics[{PointSize[0.006], points}], AspectRatio → 1]
```



```
pts = Cases[points, z_Point] /. z_Point :→ z[[1]];
```

```
ListPlot[pts, PlotJoined → False, PlotRange → Automatic,
    PlotStyle → {PointSize[0.003]}, AspectRatio → 1];
```



■ Animação de Gráficos

A forma mais rápida de fazer animações é gerar os "frames", i.e. uma sequência de gráficos (tendo o cuidado de especificar a opção PlotRange → $\{\text{range}_x, \text{range}_y, \text{range}_z\}$ igual para todos) e, depois de seleccionar o conjunto de células gráficas, usar o menu Cell → Animate Selected Graphics. Para fazer tudo isto duma vez só é necessário escrever os frames da animação num grupo fechado de células donde apenas a primeira é visível, seleccionar o conjunto e dar a ordem de animação usando SelectionAnimate.

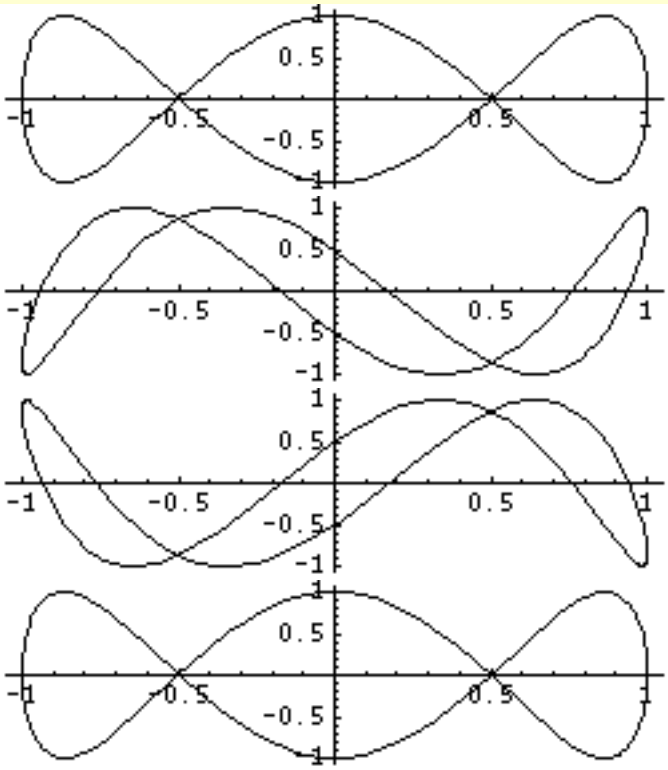
□ Escrever células criando um grupo fechado

Como é que se escreve um grupo de células fechado? O comando **CellGroup–Data[{cell1,cell2,...},Closed]** permite escrever num *Notebook* um conjunto de células agrupa–das com a primeira apenas visível.

A seguir use DisplayFunction para especificar o que fazer com as strings de Postscript geradas pelo *Mathematica* . Neste caso, basta acumular estas strings numa lista de células do tipo **Cell[–GraphicsData["PostScript", display_string],"Graphics"]** para depois escrever todos os frames da animação num grupo fechado de células no **SelectedNotebook[]**.

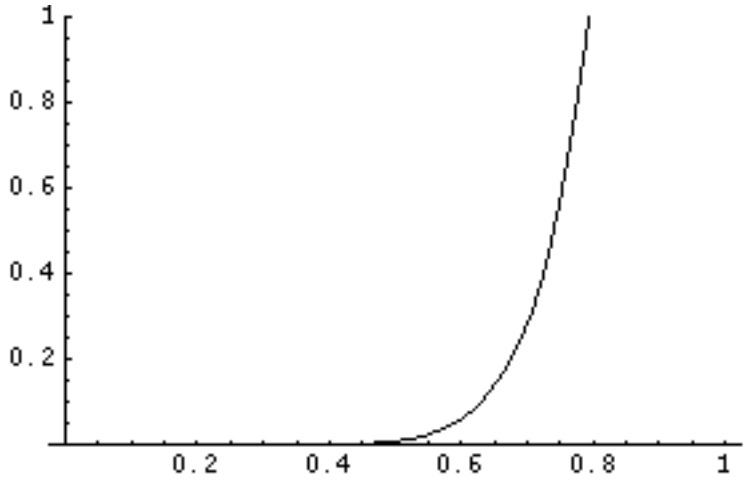
O comando **SelectionMove** para o grupo de células que acabaram de ser geradas dentro de um **CellGroup** garante que todas os frames da animação ficam seleccionados. Finalmente, use o **SelectionAnimate[]** para animar o que está seleccionado.

```
nb = SelectedNotebook[];
celllist = {};
Table[
    ParametricPlot[{Sin[x − 3 t], Cos[3 x − 15 t]}, {x, 0, 2 π},
      AspectRatio → 72 / 272, DisplayFunction → (AppendTo[celllist,
            Cell[ GraphicsData["PostScript", DisplayString[#]],
              "Graphics", ImageSize → {272, 72}]] &)],
    {t, 0, 2π/3, 2π/9}];
NotebookWrite[nb, Cell[CellGroupData[celllist, Closed]]];
SelectionMove[nb, Previous, CellGroup];
SelectionAnimate[nb, 4, AnimationDisplayTime → .3];
```



Uma outra opção envolve escrever os sucessivos "frames" de uma animação numa célula do actual *Notebook* de trabalho (cuja referência é nb = EvaluationNotebook[]) referenciada por uma propriedade CellTags → meutag. Esta célula pode ser criada usando a função nativa CellPrint, e cada "frame" é para lá redireccionado usando uma expressão adequada para a DisplayFunction.

```
Module[{nb = EvaluationNotebook[], Target, tar},
  meutag = ToString[tag] <> ToString[$SessionID];
  CellPrint[Cell["", "Graphics", CellTags → meutag]];
  Do[Pause[.1];
    Plot[a ∗ x^a, {x, 0, 1}, PlotRange → {0, 1},
      DisplayFunction → (NotebookFind[nb, meutag, All, CellTags];
          NotebookWrite[nb, Cell[GraphicsData["PostScript", DisplayString[#]],
              "Graphics", CellTags → meutag]] &)], {a, 0, 10, 1}]]
```



■ Gráficos e Som

Uma função oscilatória pode ser ouvida usando a função Play[f[t], {t, t_o, t_f}] e pressionando duas vezes o canto superior direito da célula.

```
Play[Sin[20 t] Sin[23 t] Sin[10^4 t (1 + 3/10 e^(-5 Sin[Sqrt[t]]) Sin[123.4 t])], {t, 0, 4}];
```

Cálculo Diferencial

■ Derivação Simbólica

(a) – As operações de cálculo diferencial simbólico também estão definidas no *Mathematica*.

```
{∂x ArcSin[k x − v t]   ∂t ArcSin[k x − v t]   ∂x,t ArcSin[k x − v t]
   Dt[f[x[t]], t]        ∂x (∂y (1/y A[x, y]))        D[x^y, x, y]
 TableForm
```
// FullSimplify // TraditionalForm //

$$\begin{pmatrix} \frac{k}{\sqrt{1-(t\,v-k\,x)^2}} & -\frac{v}{\sqrt{1-(t\,v-k\,x)^2}} & \frac{k\,v\,(t\,v-k\,x)}{(1-(t\,v-k\,x)^2)^{3/2}} \\ f'(x(t))\,x'(t) & \frac{y\,A^{(1,1)}(x,y)-A^{(1,0)}(x,y)}{y^2} & x^{y-1}\,(y\log(x)+1) \end{pmatrix}$$

(b) – D[f[x], x] representa a 1ª derivada de f[x], algo que em notação normal se costuma escrever $\partial_x f[x]$.

(c) – O símbolo ∂, ou \[Partial], obtém–se com a combinação ESC pd ESC ou :pd:.

(d) – A menos que se use com o operador HoldForm, a notação $\partial_x f[x]$ é imediatamente substituída por D[f[x], x], a qual é posteriormente avaliada como Derivative[1][f][x].

```
({ToString[#1][HoldForm[∂x f[x]]], "⟶", #1[∂x f[x]]} &) /@
    {FullForm, StandardForm, TraditionalForm} // TableForm
```

FullForm[$\partial_x f[x]$]	⟶	Derivative[1][f][x]
StandardForm[$\partial_x f[x]$]	⟶	$f'[x]$
TraditionalForm[$\partial_x f[x]$]	⟶	$f'(x)$

```
({ToString[#1]["HoldForm" @ HoldForm[∂x f[x]]], "⟶", #1[HoldForm[∂x f[x]]]} &) /@
    {FullForm, StandardForm, TraditionalForm} // TableForm
```

FullForm[HoldForm[$\partial_x f[x]$]]	⟶	HoldForm[D[f[x], x]]
StandardForm[HoldForm[$\partial_x f[x]$]]	⟶	$\partial_x f[x]$
TraditionalForm[HoldForm[$\partial_x f[x]$]]	⟶	$\frac{\partial f(x)}{\partial x}$

(e) – Para funções com mais de uma variável, as definições estendem–se de forma natural a D[f[x, y, …], x_1, x_2, …, x_n] ou $\partial_{x_1, x_2, \ldots, x_n} f[x, y, \ldots]$

```
({ToString[#1][HoldForm[∂x,y,x f[x, y]]], "⟶", #1[∂x,y,x f[x, y]]} &) /@
    {FullForm, TraditionalForm} // TableForm
```

FullForm[$\partial_{x,y,x} f[x, y]$]	⟶	Derivative[2, 1][f][x, y]
TraditionalForm[$\partial_{x,y,x} f[x, y]$]	⟶	$f^{(2,1)}(x, y)$

```
f[x_, z_] := a[x] + b[x, z]/x
∂y,y f[y, z]
```

$$\frac{2\,b[y,\,a]}{y^3} + a''[y] - \frac{2\,b^{(1,0)}[y,\,a]}{y^2} + \frac{b^{(2,0)}[y,\,a]}{y}$$

```
({ToString[#1]["HoldForm" @ HoldForm[∂x,x f[x, y]]],
        "⟶", #1[HoldForm[∂x,x f[x]]]} &) /@
    {FullForm, StandardForm, TraditionalForm} // TableForm
```

FullForm[HoldForm[$\partial_{x,x} f[x, y]$]]	⟶	HoldForm[D[f[x], x, x]]
StandardForm[HoldForm[$\partial_{x,x} f[x, y]$]]	⟶	$\partial_{x,x} f[x]$
TraditionalForm[HoldForm[$\partial_{x,x} f[x, y]$]]	⟶	$\frac{\partial^2 f(x)}{\partial x\,\partial x}$

⚠ A menos que se indique explícitamente a dependência de f em x com a notação f[x], a derivação simbólica D[expr, x] trata como constante qualquer símbolo que não esteja pré–definido como uma função de x. Assim, não se deve usar $\partial_x f$ em expressões (mesmo que a intenção fosse substituir posteriormente f por f[x]) porque na avaliação $\partial_x f \to 0$. Pode–se no entanto usar HoldForm[$\partial_x f$], desde que depois se faça ReleaseHold no fim para obter uma expressão avaliada completamente.

```
∂x f /. f → a[x]
```

```
0
```

```
HoldForm[∂x f] /. f → a[x] // ReleaseHold
```

```
a'[x]
```

```
∂x f[x] /. f :→ Function[{t}, a[t]]
```

```
a'[x]
```

Enquanto que $\partial_x f$ dá em geral derivadas parciais de f, o operador Dt[expr, τ] determina a derivada total de expr, assumindo que todos os símbolos de expr dependem implícitamente de τ. (É possível dar a opção Constants → { a_1, … } para indicar quais os símbolos que devem ser considerados meros parâmetros.)

```
ClearAll[f]
```

```
Dt[f[x], t] // Simplify // TraditionalForm
```

$$\frac{dx}{dt} f'(x)$$

```
Dt[f[x, t], t] // TraditionalForm
```

$$f^{(0,1)}(x, t) + \frac{dx}{dt} f^{(1,0)}(x, t)$$

```
z =.
```

```
Dt[f[x, y, z, t], t] /. Derivative[n__][f_][x__] :⇢ (rl = (k → {n}.{x}); HoldForm[∂k f] /. rl) //
  TraditionalForm
```

$$\frac{\partial f}{\partial t} + \frac{dx}{dt}\frac{\partial f}{\partial x} + \frac{dy}{dt}\frac{\partial f}{\partial y} + \frac{dz}{dt}\frac{\partial f}{\partial z}$$

```
ReleaseHold[%]
```

0

```
Format[Derivative[z__][f_][x__], TraditionalForm] :=
  (HF["∂"ξ/ζ f] /. ζ → (DisplayForm@RowBox@Flatten@((Table[#1, {j, #2}] & @@ ##) & /@
                    Transpose[{{x}, {z}}])) /. ξ → If[Tr[{z}] == 1, "", Tr[{z}]])
```

```
Dt[f[x, y, z, t], t] // TraditionalForm
```

$$\partial_t f + \frac{dz}{dt}(\partial_z f) + \frac{dy}{dt}(\partial_y f) + \frac{dx}{dt}(\partial_x f)$$

■ INTEGRAÇÃO

A primitivação e integração estão implementadas através de Integrate[f[x], x], ou $\int f[x]\,dx$.

O símbolo $\int$ ou \[Integral] pode–se introduzir escrevendo ESC int ESC.

d ou \[DifferentialD] (ESC dd ESC) é um operador tipo *prefixo* que requer sempre um argumento, tal como ∂ aliás.

O símbolo que segue d (e.g. dx) determina a variável de integração (i.e. x).

Para efeitos de integração x deve ser um símbolo e não uma expressão, a menos que seja simboli–zada (via Symbolize).

```
d\[Xi]
```

```
DifferentialD[ξ]
```

```
∫Sin[π - x^2] dx
```

$$\sqrt{\frac{\pi}{2}}\,\text{FresnelS}\left[\sqrt{\frac{2}{\pi}}\,x\right]$$

```
∫_a0^a1 Sin[π - x^2] dx
```

$$-\sqrt{\frac{\pi}{2}}\,\text{FresnelS}\left[\sqrt{\frac{2}{\pi}}\,a_0\right] + \sqrt{\frac{\pi}{2}}\,\text{FresnelS}\left[\sqrt{\frac{2}{\pi}}\,a_1\right]$$

```
Dt[∫_c[x,t]^d[x,t] f[ξ] dξ, t];
```

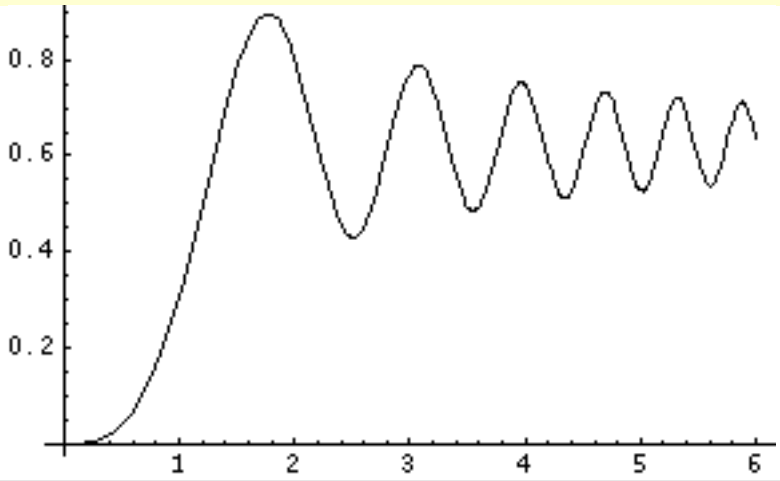
$$-f[c[x,t]]\left(c^{(0,1)}[x,t] + Dt[x,t]\,c^{(1,0)}[x,t]\right) + f[d[x,t]]\left(d^{(0,1)}[x,t] + Dt[x,t]\,d^{(1,0)}[x,t]\right)$$

```
D[∫_c[x,t]^d[x,t] f[ξ] dξ, t]
```

$$-f[c[x,t]]\,c^{(0,1)}[x,t] + f[d[x,t]]\,d^{(0,1)}[x,t]$$

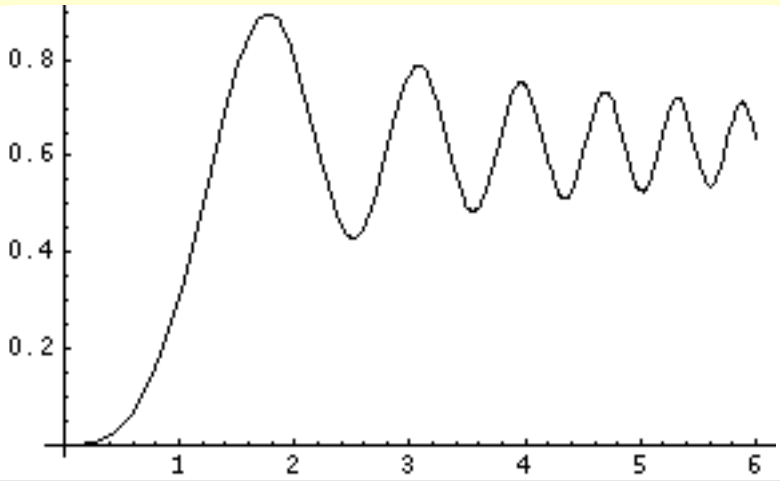
Para funções demasiado complexas para que o *Mathematica* encontre primitivas, NIntegrate[f[x], {x, x_0, x_1}] dá uma aproximação numérica do integral num intervalo [x_0, x_1].

```
Plot[∫_0^x Sin[π - ξ^2] dξ, {x, 0, 6}] // Timing
```




```
{15.05 Second, - Graphics -}
```

```
Plot[∫_0^x Sin[π - ξ^2] dξ // Release, {x, 0, 6}] // Timing
```

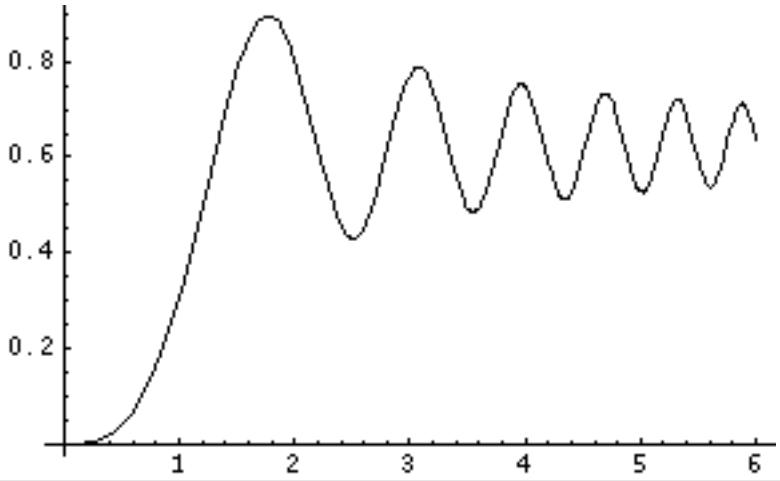



```
{5.43 Second, - Graphics -}
```

```
F = Compile[{x}, ∫_0^x Sin[π - ξ^2] dξ // Release]
```

```
CompiledFunction[{x}, √(π/2) FresnelS[√(2/π) x], -CompiledCode-]
```

```
Plot[F[x], {x, 0, 6}] // Timing
```



{5.38 Second, - Graphics -}

■ Solução de Equações Diferenciais

Para a solução de equações diferenciais ordinárias DSolve[F[y[t], y'[t], ..., $y^{(n)}$[t]], y[t], t] procura encontar soluções para a função y[t] da variável independente t.

□ Inicial.

```
<< Utilities`Notation`

$Post := (If[Head[#] === List, TableForm[#], #] &)
```

```
Symbolize[x_o]
```

□ Exemplo I

A seguinte equação linear não–homogénea de coeficientes dependentes do tempo apenas tem uma solução genérica.

```
Eqn1 = x'[t] == a[t] x[t] + b[t];
```

```
InitCond1 =          (x[0] == x_o);
```

```
sol1 =          DSolve[Eqn1 && InitCond1, x[t], t];
```

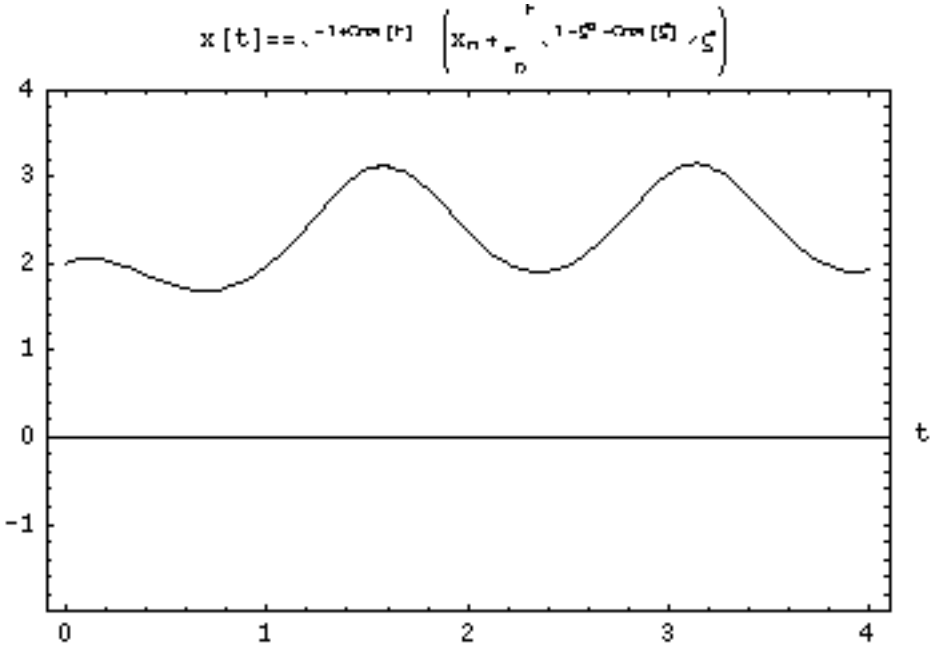
$$x[t] \to e^{\int_0^t a[\tau]\,d\tau}\, x_o + e^{\int_{\tau_o}^t a[\tau]\,d\tau} \int_0^t e^{-\int_{\tau_o}^{\zeta} a[\tau]\,d\tau}\, b[\zeta]\,d\zeta$$

```
sol1 = sol1 /.
      Thread[(Cases[sol1, z_ /; StringMatchQ[ToString[z], "K$*"], {0, ∞}] // Union) → {τ, τ_o, ζ, ζ_o}] //
    At[1, 1, 2, (# // Distribute // Collect[#, _Power] &) /. (-∫_s_^0 z_ dx_ + ∫_s_^t z_ dx_ :→ ∫_0^t z dx &)]
```

A verificação é simples:

x'[t] == (D[x[t] /. sol1⟦1, 1⟧, t] // Simplify) /. Reverse[sol1⟦1, 1⟧]

$x'[t] == b[t] + a[t]\,x[t]$



sys = $\{a[t\_] \to -\text{Sin}[4\,t],\ b[t\_] \to e^{-t^2},\ \tau_o \to 0\}$;

$\text{Plot}\Big[x[t] /. sol1⟦1, 1⟧ /. sys /. x_o \to 2 // \text{Release}, \{t, 0, 4\}, \text{PlotRange} \to \{-2, 4\}, \text{Frame} \to \text{True},$
$\text{AxesLabel} \to \{t, x\}, \text{PlotLabel} \to x[t] == e^{-1+\text{Cos}[t]}\left(x_o + \int_0^t e^{1-\zeta^2-\text{Cos}[\zeta]}\,d\zeta\right)\Big]$;

□ EXEMPLO II

Esta é uma equação não–linear por causa do termo $x[t]^2$ mas é separável

Eqn2 = $\left(x'[t] == a[t]\,x[t]^2\right)$;

InitCond2 = $(x[0] == x_o)$;

sol2 = DSolve[Eqn2 && InitCond2, x[t], t];

sol2 /. Thread[(Cases[sol2, z_ /; StringMatchQ[ToString[z], "K$*"], {0, ∞}] // Union) → {τ, τ_o}];
$\%$ // At$\Big[1, 1, 2, 2, 1, \text{Collect}[\#, x_o] /. \left(\int_{s_}^0 z_\,dx_ - \int_{s_}^t z_\,dx_\right) :\to -\int_0^t z\,dx$ &$\Big]$

$x[t] \to \dfrac{x_o}{1 - x_o \int_0^t a[\tau]\,d\tau}$

□ EXEMPLO III

Esta é a equação de movimento de um oscilador amortecido com frequência natural ω e amortecimento λ.

```
Eqn3=          x''[t]+2 λ x'[t]+ω^2 x[t]==0;
```

```
InitCond3 =          {x[0] == x_0, x'[0] == v_0};
```

```
sol3 = DSolve[Eqn3 && And @@ InitCond3, x[t], t]
```

A solução dá o movimento x[t] dum oscilador amortecido. Quando $\lambda \geq \omega$ o movimento deixa de ser oscilatório.

$$x[t] \to e^{-t\lambda}\left(x_0 \operatorname{Cos}\left[t\sqrt{-\lambda^2+\omega^2}\right] + \frac{(v_0+x_0\,\lambda)\operatorname{Sin}\left[t\sqrt{-\lambda^2+\omega^2}\right]}{\sqrt{-\lambda^2+\omega^2}}\right)$$

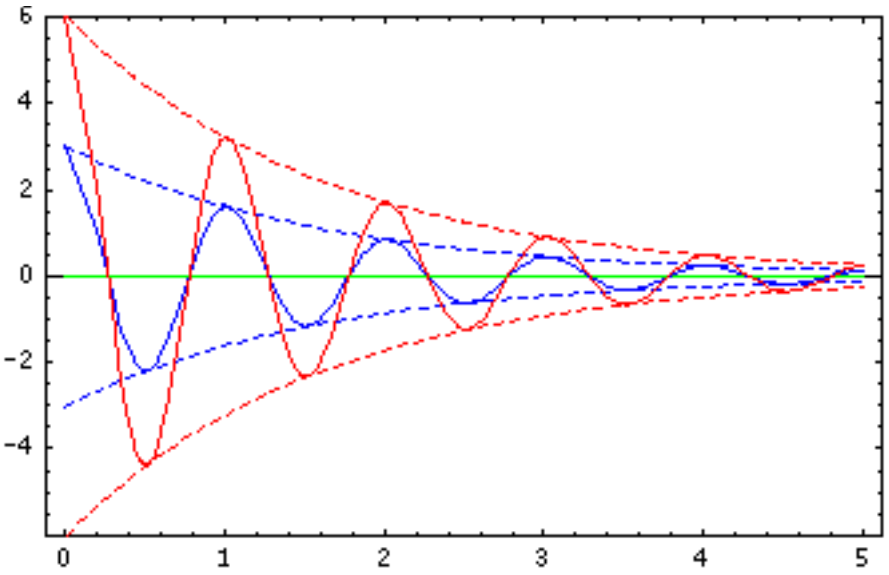
```
sol3 = DSolve[Eqn3 && And @@ InitCond3, x[t], t] //
    At[1, 1, 2, ((# e^(t λ) / HoldForm[e^(t λ)] // ExpToTrig // Simplify // Collect[#, {x_0, v_0}] &) /. Sqrt[z_] :> i Sqrt[-z] /
        (Sqrt[z_])^-1 :> (i Sqrt[-z])^-1 // Collect[Apart[#1], _HoldForm] & //
      At[2, 2, Simplify]) &] // ReleaseHold;
```

Se mantivermos $\lambda < \omega$ poderemos ainda simplificar esta solução para

$$x[t] \to e^{-t\lambda}\sqrt{x_0^2 + \frac{(v_0+x_0\,\lambda)^2}{-\lambda^2+\omega^2}}\;\operatorname{Sign}[x_0]\operatorname{Sin}\left[t\sqrt{-\lambda^2+\omega^2} + \operatorname{ArcTan}\left[\frac{x_0\sqrt{-\lambda^2+\omega^2}}{v_0+x_0\,\lambda}\right]\right]$$

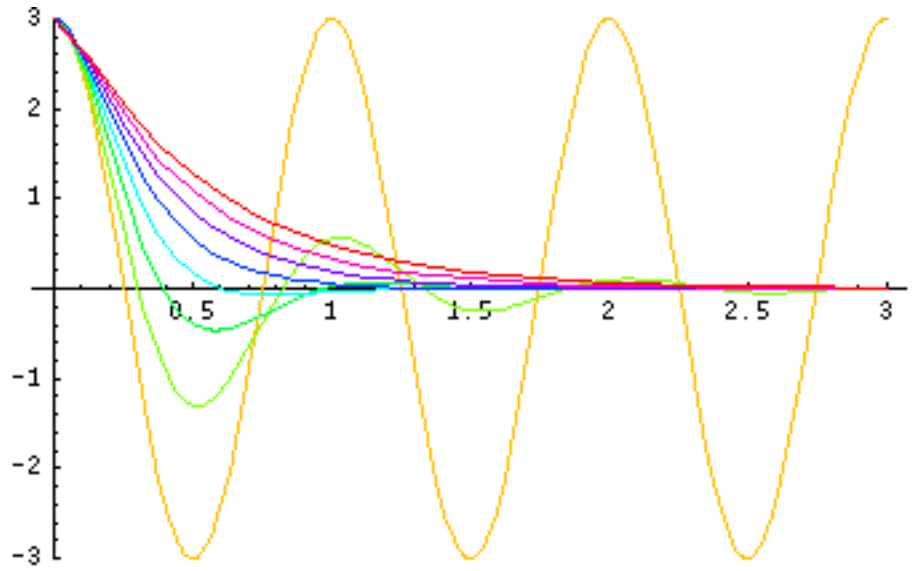
```
sol3 // At[1, 1, 2,
  # /. z_ Cos[θ_] + y_ Sin[θ_] :> (Sign[z] Sqrt[z^2 + (y^2 // Simplify)] (Sin[θ + ArcTan[z/y]] // Simplify)) &]
```

□ Posição inicial



```
Plot[#, {t, 0, 5}, PlotStyle →
    ({Hue[#/3], {Dashing[.01 {1, 1, 1}], Hue[#/3]}, {Dashing[.01 {1, 1, 1}], Hue[#/3]}} & /@
      Range[6] // Flatten[#, 1] &), PlotRange → {-6, 6}, Frame → True] & @@
  Table[{x[t], {#, -#} & @@ ℯ^(-t λ) Sqrt[x_o^2 - (v_o + x_o λ)^2/(λ^2 - ω^2)]} //. {sol3[[1, 1]], v_o → 0, λ → ω/10, ω → 2 π},
    {x_o, 0, 6, 3}];
```

□ AMORTECIMENTO




```
Plot[# // Release, {t, 0, 3}, PlotStyle → ({Hue[#/8]} & /@ Range[8] // Flatten), PlotRange
    → {-3, 3}] & @@ Table[Re[x[t]] //. {sol3[[1, 1]], v_o → 0, x_o → 3, ω → 2 π}, {λ, 0, 4 π, 4.1 π/8}];
```

Quando não existem formas analíticas para a resolução de equações diferenciais pode–se usar a forma numérica NDSolve[F[y[t], y'[t], …, $y^{(n)}$[t]], y[t], {t, t_0, t_1}] para aproximar a solução verdadeira y[t], no intervalo da variável independente [t_0, t_1], por via de uma InterpolatingFunction.

□ EXEMPLO IV

A seguinte equação é não–linear e não–separável, pelo que uma solução analítica deve ser inexistente.

```
Eqns4 = x'(t) == t^2 - 3 t - (t + 1) |x(t)| + 1;
```

```
DSolve[Eqns4 && x[0] == x_o, x[t], t]
```

— *Solve::ifun : Inverse functions are being used by Solve, so some solutions may not be found.*

— *InverseFunction::ifun : Inverse functions are being used. Values may be lost for multivalued inverses.*

— *InverseFunction::ifun : Inverse functions are being used. Values may be lost for multivalued inverses.*

— *Solve::tdep : The equations appear to involve the variables to be solved for in an essentially non−algebraic way.*

```
$Aborted
```

Uma solução numérica é mais fácil, embora possam aparecer singularidades antes de acabar o tempo

```
NDSolve[Eqns4 && x[0] == 1, x[t], {t, 0, 5}]
```

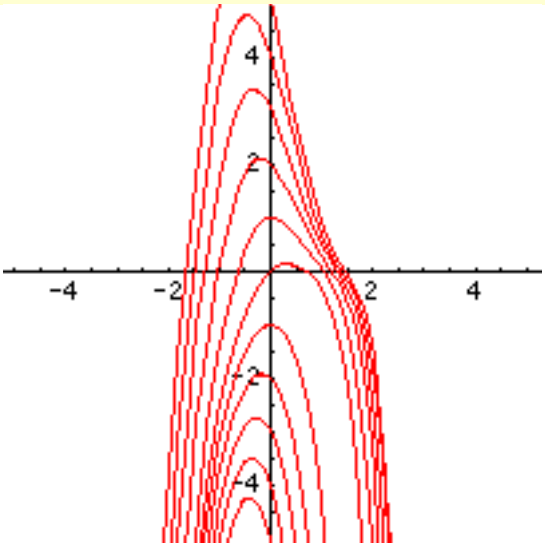
```
{{x[t] → InterpolatingFunction[{{0., 5.}}, <>][t]}}
```

```
Options[NDSolve] // Sort // PadRight[#, Length[#] + 2, ""] &;
% // Partition[#, (Length[#])/2] & // #^T & // TableForm
```

AccuracyGoal → Automatic	NormFunction → Automatic
Compiled → True	PrecisionGoal → Automatic
DependentVariables → Automatic	SolveDelayed → False
EvaluationMonitor → None	StartingStepSize → Automatic
MaxStepFraction → $\frac{1}{10}$	StepMonitor → None
MaxSteps → Automatic	WorkingPrecision → MachinePrecision
MaxStepSize → Automatic	
Method → Automatic	

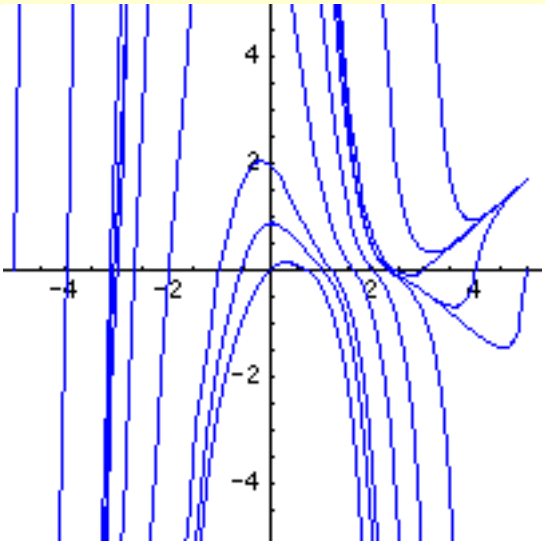
```
soltbl1 = (x[t] /. NDSolve[Eqns4 && x[0] == #, x[t], {t, −5, 5}] & /@ Range[−5, 5, 1]);
```

```
gr1 = Plot[#, {t, −5, 5}, PlotRange → {−5, 5}, AspectRatio → 1, PlotStyle → Red] & @ soltbl1;
```

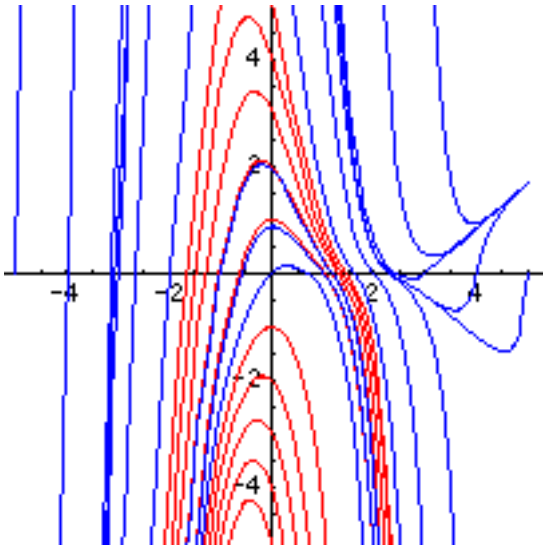



```
soltbl2 = (x[t] /. NDSolve[Eqns4 && x[#] == 0, x[t], {t, −5, 5}] & /@ Range[−5, 5, 1]);
```

```
gr2 = Plot[#, {t, −5, 5}, PlotRange → {−5, 5}, AspectRatio → 1, PlotStyle → Blue] & @ soltbl2;
```




```
Show[{gr1, gr2}, DisplayFunction → $DisplayFunction];
```



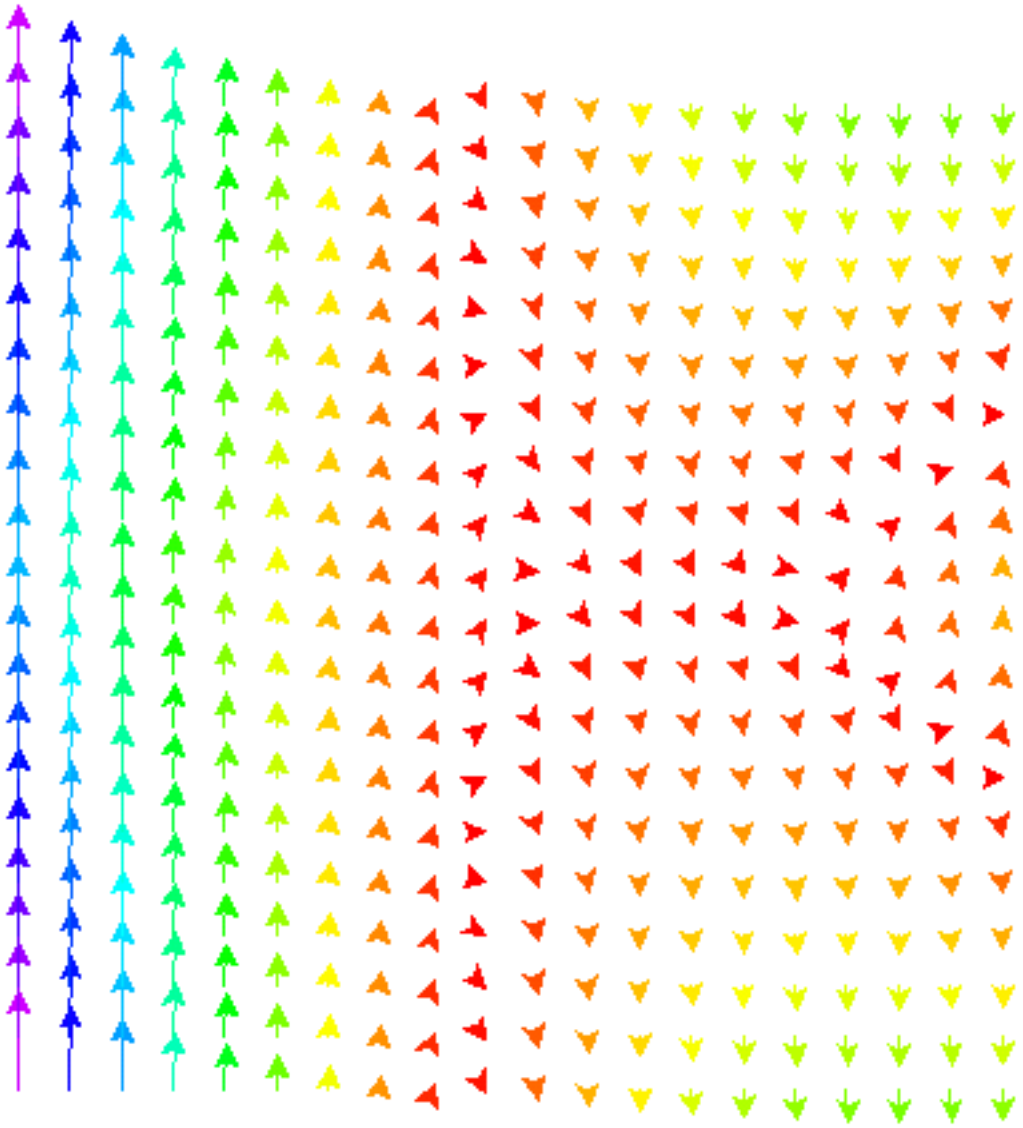
Note que uma equação diferencial de 1ª ordem $\frac{dx}{dt} = f(x, t)$ de facto pode ser entendida como um campo de direcções, representado pelo versor $\vec{e}_f = \left\{\frac{1}{\sqrt{f(x,t)^2+1}}, \frac{f(x,t)}{\sqrt{f(x,t)^2+1}}\right\}$

```
<< Graphics`PlotField`
```

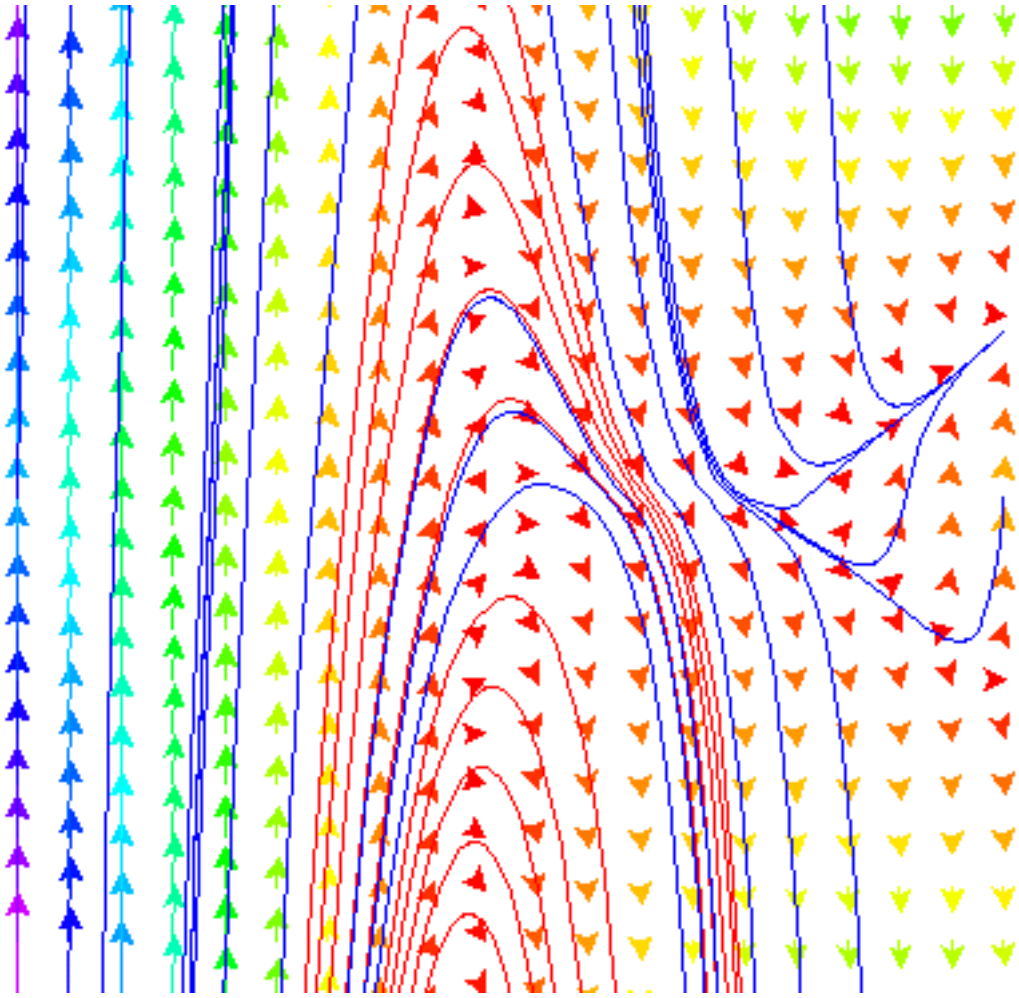
```
Options[PlotVectorField] // Sort // PadRight[#, Length[#] + 2, ""] & // Partition[#, 13] & // #^T & //
  TableForm
```

AspectRatio → Automatic	FrameStyle → Automatic	PlotRegion → Automatic
AspectRatio → $\frac{1}{\text{GoldenRatio}}$	FrameTicks → Automatic	Prolog → {}
Axes → False	GridLines → None	RotateLabel → True
AxesLabel → None	HeadCenter → 1	ScaleFactor → Automatic
AxesOrigin → Automatic	HeadLength → 0.02	ScaleFunction → None
AxesStyle → Automatic	HeadScaling → Automatic	Ticks → Automatic
Background → Automatic	HeadShape → Automatic	ZeroShape → Automatic
ColorFunction → None	HeadWidth → 0.5	DefaultFont :→ $DefaultFont
ColorOutput → Automatic	ImageSize → Automatic	DisplayFunction :→ $DisplayFunctio
DefaultColor → Automatic	MaxArrowLength → None	FormatType :→ $FormatType
Epilog → {}	PlotLabel → None	TextStyle :→ $TextStyle
Frame → False	PlotPoints → Automatic	
FrameLabel → None	PlotRange → All	

```
gr3 = PlotVectorField[{1, -(1 + t) Abs[x] + 1 - 3 t + t^2},
        {t, -5, 5}, {x, -5, 5}, ColorFunction → (Hue[.8 #] &), ScaleFactor → 1,
        HeadWidth → 1, HeadLength → 0.02`, HeadCenter → .8, PlotPoints → 20];
```



```
Show[{gr3, gr1, gr2}, Cases[Options[gr1], z_Rule /; ! FreeQ[z, PlotRange]][[1]],
    DisplayFunction → $DisplayFunction];
```



Exemplos de Cálculo Numérico

Equações algébricas, Raízes.

```
<< Utilities`Notation`
```

```
Symbolize[y_o]
```

```
Symbolize[x_o]
```

```
Symbolize[y_o, WorkingForm → TraditionalForm]
```

— *Symbolize::boxSymbolExists : Warning: The box structure attempting to be symbolized has a similar or identical symbol already defined, possibly overriding previously symbolized box structure.*

```
Notation[Composition[g, f] ⟹ (g ∘ f), WorkingForm → TraditionalForm]
```

UpdateNotationsInNotebook[]

```
ti_z_ := Sequence @@ {FontFamily → Times, FontSlant → Italic, FontSize → z}
tb_z_ := Sequence @@ {FontFamily → Times, FontWeight → Bold, FontSize → z}
```

■ Solução de equações Algébricas

Para a solução de equações algébricas o *Mathematica* dispõe de algoritmos Solve[*eqns*, *vars*, *elims*]e Reduce[*eqns*, *vars*, *elims*].

O resultado de Solve é sempre uma lista de Regras de Substituição (eventualmente vazia) na forma {var → sol, …}.

Pelo contrário, Reduce procura reduzir ao máximo o número de equações independentes, e especi–fica em forma de relação lógica (AND && e OR ||)as condições de validade para essas equações.

```
eqns := {a x – b y == 0, b x – a == 2 y}
```

```
Solve[eqns, {x, y}]
```

$$\left\{\left\{x \to -\frac{a\,b}{2\,a - b^2},\ y \to -\frac{a^2}{2\,a - b^2}\right\}\right\}$$

```
Reduce[(And @@ eqns), {x, y}]
```

$$a == 0 \&\& b == 0 \&\& y == 0 \,||\, x == \frac{a\,b}{-2\,a + b^2} \&\& y == -\frac{a^2}{2\,a - b^2} \&\& 2\,a - b^2 \neq 0$$

■ Zeros de Funções

Excepto para funções polinomiais até ordem 5, a obtenção de zeros duma função y = f(x) só pode ser efectuada usando métodos iterativos: a partir dum valor x_0 estimado próximo dum zero z_r de f, é preciso encontrar uma função de iteração $\mathcal{F}$ tal que z_r seja um ponto fixo de $\mathcal{F}$:

$$x_{i+1} = \mathcal{F}(x_i) = \mathcal{F}(\mathcal{F}(x_{i-1})) = \ldots = \mathcal{F}^{i+1}(x_0)$$

$$z_r = \mathcal{F}(z_r) = \lim_{i\to\infty} \mathcal{F}^i(x_0)$$

O sucesso da determinação do zero z_r depende de:

(a) – Determinação de uma função de iteração $\mathcal{F}$ adequada,

(b) – Condições de convergência da sequência $\{x_i\}_{i=0,\ldots,\infty}$,

(c) – Rapidez de convergência dessa sequência.

□ Exercício

Uma sequência convergente $\{x_i\}_{i=0,\ldots,\infty}$ para um limite z_r designa–se ´Convergente de Grau p´ se

$$\lim_{i\to\infty}\left(\frac{|x_{i+1} - z_r|}{|x_i - z_r|^p}\right) = C \neq 0$$

(a) – Mostre que, para o caso em que $p \in \mathbb{N}$, $\mathcal{F}$ é um método de ordem p se se anularem todas as derivadas $\mathcal{F}^{(n)}[z_r]$ de ordem n < p !

$$x_{i+1} = \mathcal{F}(x_i) \approx \mathcal{F}(z_r) + \frac{\mathcal{F}^p[z_r]\,(x_i - z_r)^p}{p!} + O\left(|x_i - z_r|^{p+1}\right)$$

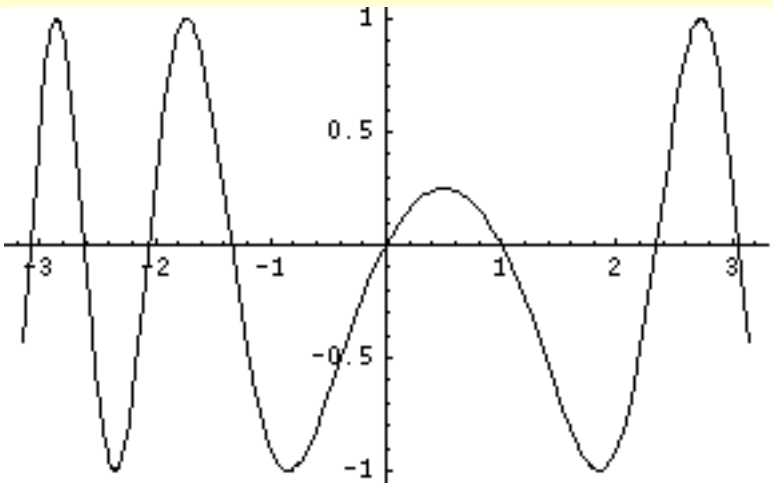
(b) – Mostre que a ordem do método da secante (posição falsa) é p = $\frac{1+\sqrt{5}}{2}$.

Métodos iterativos $\mathcal{F}$ em que a estimativa inicial x_0 é arbitrária e conduza sempre ao zero z_r mais próximo são difíceis de encontrar. O método de Laguerre é um desses métodos, e a convergência é de grau 3 no caso de raízes simples (para raízes degeneradas a convergência é apenas linear):

$$x_{i+1} = x_i - \frac{i\,f(x_i)}{f'(x_i) \pm \sqrt{(i-1)^2\,f'(x_i)^2 - i\,(i+1)\,f(x_i)\,f''(x_i)}}$$

□ APLICAÇÃO

```
f[x_] := Sin[x - x^2]
```



```
data := {}

x0 = x1 = N[1.85, 20];

For[i = 1, i < 9, i++, AppendTo[data, {
SequenceForm["x_ =",
        SetAccuracy[x0 = x0 - i f[x0] / (f'[x0] - Sqrt[(i - 1)^2 f'[x0]^2 - i (i + 1) f[x0] f''[x0]]), 20]],
SequenceForm["ϵ =", (x0 - z_) / f[x0]],

SequenceForm["x+ =",
        SetAccuracy[x1 = x1 - i f[x1] / (f'[x1] + Sqrt[(i - 1)^2 f'[x1]^2 - i (i + 1) f[x1] f''[x1]]), 20]],
SequenceForm["ϵ =", (x1 - z+) / f[x1]]}]];

z_ =.; z+ =.;
Thread[r[{z_ == x, z+ == x}, SetAccuracy[{FindRoot[f[x] == 0, {x, x0}], FindRoot[f[x] == 0, {x, x1}]},
        20]]] /. r :→ ReplaceAll /. Equal :→ Set;

StylePrint[TableForm[data], "DisplayFormula"];

{SetAccuracy[{FindRoot[f[x] == 0, {x, x0}], FindRoot[f[x] == 0, {x, x1}]}, 20]} //
  StylePrint[TableForm[#], "DisplayFormula"] &
```

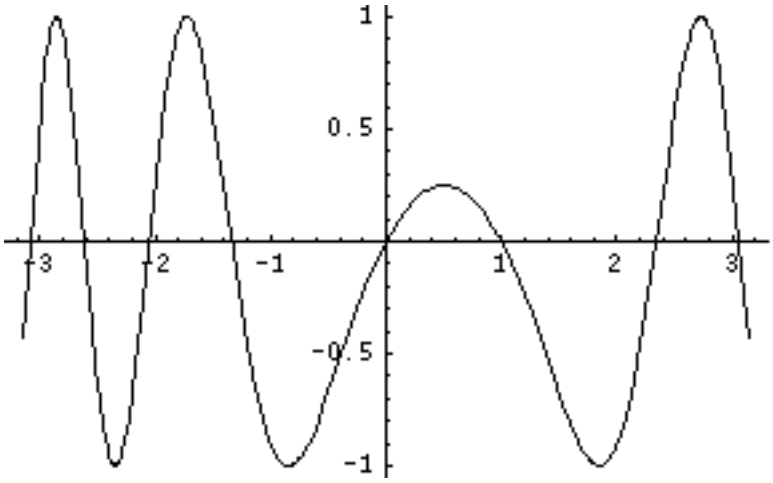
x_ =1.5878540539885905858	ϵ =−0.731468	x+ =2.11151524919767119570	ϵ =0.322471
x_ =1.27106668536269040715	ϵ =−0.802525	x+ =2.2736435927340865604	ϵ =0.279409
x_ =0.9786079916928005495	ϵ =−1.02193	x+ =2.33817206748095207658	ϵ =0.271761
x_ =0.99969042120596263157	ϵ =−1.00031	x+ =2.3416252792236487146	ϵ =0.271499
x_ =0.99999995205795078057	ϵ =−1.	x+ =2.3416277185106708636	ϵ =0.271454
x_ =0.9999999999999990008	ϵ =−1.	x+ =2.3416277185114782178	ϵ =0.
x_ =1.0000000000000000000	ϵ =Indeterminate	x+ =2.3416277185114782178	ϵ =0.
x_ =1.0000000000000000000	ϵ =Indeterminate	x+ =2.3416277185114782178	ϵ =0.

$x \to 1.0000000000000000000$ $x \to 2.3416277185114782178$

□ NOTAS :

```
Off[Power::"infy", ∞::"indet"]
```

```
(Plot[Sin[x - x^2], {x, -π, π}])
```



(a) – Use **Thread[f[a_1, a_2]]** para distribuir f pelos elementos de listas a_1 e a_2

```
?? Thread
```

```
Thread[f[args]] "threads" f over any lists that appear in args. Thread[f[
    args], h] threads f over any objects with head h that appear in args.
    Thread[f[args], h, n] threads f over objects with head h that appear
    in the first n args. Thread[f[args], h, −n] threads over the last n args.
    Thread[f[args], h, {m, n}] threads over arguments m through n. More…
```

```
Attributes[Thread] = {Protected}
```

```
Thread[r[{a, a}, {a → v, a → d}]]
```

```
{r[a, a → v], r[a, a → d]}
```

```
% /. r → ReplaceAll
```

```
{v, d}
```

(b) – Use **SetAccuracy[expr,m]** para exibir cálculos de precisão arbitrária m.

(c) – Use **N[expr,n]** para usar precisão n nos cálculos numéricos.

```
?? SetAccuracy
```

```
SetAccuracy[expr, n] yields a version of expr in which all
    numbers have been set to have an accuracy of n digits. More…
```

```
Attributes[SetAccuracy] = {Listable, Protected}
```

■ Método Genérico

Designando por x_r um valor do argumento de $y = f(x)$ para o qual $f(x_r) = 0$, desde que $f'(x) \neq 0$ a função $f(x)$ é invertível numa vizinhança de x_r, e podemos escrever $x_r = g(0)$, onde $x = g(y) \equiv g(f(x))$ nessa vizinhança. Se $y_0 \approx 0$, podemos desenvolver $g(0)$ na forma

$$g(0) == \sum_{n=0}^{\infty} \frac{(-y_o)^n}{n!}\, g^{(n)}(y_o)$$

Por exemplo, até à ordem 3:

```
g[y] ≃ Series[g[y], {y, y₀, 3}]
```

(1.1) $$g(y) \simeq g(y_o) + g'(y_o)\,(y-y_o) + \frac{1}{2}\,g''(y_o)\,(y-y_o)^2 + \frac{1}{6}\,g^{(3)}(y_o)\,(y-y_o)^3 + O\left((y-y_o)^4\right)$$

Embora se desconheça a função x = g(y), as suas derivadas podem ser recursivamente deduzidas das de *f(x)*. Dado que g(y) ≡ g∘f(x) = x, se derivarmos sucessivamente em ordem a x obtemos

```
TableForm[(HoldForm[(g∘f)^(#1)[x]] == Composition[g, f]^(#1)[x] == (#1 &)^(#1)[x] &) /@ Range[0, 3] /.
    f[x] → y] // TraditionalForm
```

$$(g\circ f)^{(0)}(x) == g(y) == x$$
$$(g\circ f)'(x) == f'(x)\,g'(y) == 1$$
$$(g\circ f)''(x) == g''(y)\,f'(x)^2 + g'(y)\,f''(x) == 0$$
$$(g\circ f)^{(3)}(x) == g^{(3)}(y)\,f'(x)^3 + 3\,f''(x)\,g''(y)\,f'(x) + g'(y)\,f^{(3)}(x) == 0$$

Resolvendo em ordem às derivadas de g(y)

```
(diffg = (S[(Composition[g, f])^(#1)[x] == (#1 &)^(#1)[x], g^(#1)[y]] &) /@ Range[3] /. f[x] → y /.
        S :→ Solve /. Rule :→ Equal // Flatten) // TableForm // TraditionalForm
```

$$g'(y) == \frac{1}{f'(x)}$$
$$g''(y) == -\frac{g'(y)\,f''(x)}{f'(x)^2}$$
$$g^{(3)}(y) == -\frac{3\,f'(x)\,f''(x)\,g''(y) + g'(y)\,f^{(3)}(x)}{f'(x)^3}$$

```
(diffgr = Reduce[diffg, (g^(#1)[y] &) /@ Range[3]] /. And :→ List /. Equal :→ Rule /. y → y_) //
  TableForm // TraditionalForm
```

$$g'(y) == \frac{1}{f'(x)}$$
$$g''(y) == -\frac{f''(x)}{f'(x)^3} \qquad f'(x) \neq 0 \;\&\&\; y == f(x)$$
$$g^{(3)}(y) == \frac{3\,f''(x)^2 - f'(x)\,f^{(3)}(x)}{f'(x)^5}$$

Substituindo em (1.1) estas expressões, obtemos para $y_0 = f(x_0)$,

```
At[2, HoldForm[#1] /. Plus → List &][
  g[y] == Normal[Series[g[y], {y, y₀, 3}]] /. Drop[diffgr, -1] /. y → 0 /. x :→ x₀ /. g[y₀] :→ x₀  /.
    y₀ :→ f[x₀] /. g[0] → x₁] /. List → Plus
```

$$x_1 == x_o - \frac{f(x_o)}{f'(x_o)} - \frac{f''(x_o)\, f(x_o)^2}{2\, f'(x_o)^3} - \frac{\left(3\, f''(x_o)^2 - f'(x_o)\, f^{(3)}(x_o)\right) f(x_o)^3}{6\, f'(x_o)^5}$$

Nesta expressão, os dois primeiros termos representam uma aproximação de Newton–Raphson

■ Métodos iterativos: convergência de ordem quadrática, cúbica, etc

```
g[y] ≃ Series[g[y], {y, y₀, 3}];
```

```
{Composition[g, f]^(#1)[x] == (#1 &)^(#1)[x]} & /@ Range[4] /. f[x] → y // TableForm // TraditionalForm
```

$$f'(x)\,g'(y) = 1$$
$$g''(y)\,f'(x)^2 + g'(y)\,f''(x) = 0$$
$$g^{(3)}(y)\,f'(x)^3 + 3\,f''(x)\,g''(y)\,f'(x) + g'(y)\,f^{(3)}(x) = 0$$
$$g^{(4)}(y)\,f'(x)^4 + 6\,f''(x)\,g^{(3)}(y)\,f'(x)^2 + 4\,g''(y)\,f^{(3)}(x)\,f'(x) + 3\,f''(x)^2\,g''(y) + g'(y)\,f^{(4)}(x) = 0$$

```
(diffg = (s[(Composition[g, f])^(#1)[x] == (#1 &)^(#1)[x], g^(#1)[y]] &) /@ Range[4] /. f[x] → y /. s :→ Solve /.
      Rule :→ Equal // Flatten) // TableForm // TraditionalForm
```

$$g'(y) == \frac{1}{f'(x)}$$
$$g''(y) == -\frac{g'(y)\,f''(x)}{f'(x)^2}$$
$$g^{(3)}(y) == \frac{-3\,f'(x)\,f''(x)\,g''(y) - g'(y)\,f^{(3)}(x)}{f'(x)^3}$$
$$g^{(4)}(y) == \frac{-6\,f''(x)\,g^{(3)}(y)\,f'(x)^2 - 4\,g''(y)\,f^{(3)}(x)\,f'(x) - 3\,f''(x)^2\,g''(y) - g'(y)\,f^{(4)}(x)}{f'(x)^4}$$

```
(diffgr = Reduce[diffg, (g^(#1)[y] &) /@ Range[4]] /. And :→ List /. Equal :→ RuleDelayed /. y :→ y__) //
   TableForm // TraditionalForm
```

$$g'(y\_\_) :\to \frac{1}{f'(x)}$$
$$g''(y\_\_) :\to -\frac{f''(x)}{f'(x)^3}$$
$$g^{(3)}(y\_\_) :\to \frac{3\,f''(x)^2 - f'(x)\,f^{(3)}(x)}{f'(x)^5}$$
$$g^{(4)}(y\_\_) :\to \frac{-15\,f''(x)^3 + 10\,f'(x)\,f^{(3)}(x)\,f''(x) - f'(x)^2\,f^{(4)}(x)}{f'(x)^7}$$
$$f'(x) \neq 0$$

```
g[y] ≃ Normal[Series[g[y], {y, y₀, 3}]] //. Drop[diffgr, -1] /. x → x₀ /. g[y₀] → x₀ /. g[y] → x /.
  y₀ → f[x₀];
```

$$x \simeq x_O + \frac{1}{f'(x_O)}(y - f(x_O)) - \frac{f''(x_O)}{2\,f'(x_O)^3}(y - f(x_O))^2 + \frac{\left(3\,f''(x_O)^2 - f'(x_O)\,f^{(3)}(x_O)\right)}{6\,f'(x_O)^5}(y - f(x_O))^3$$

O significado dos sucessivos termos desta expressão polinomial em y é ilustrada gráficamente como aproximações lineares, quadráticas, cúbicas etc. à curva f(x) no ponto $P_0 = \{x_0, f(x_0)\}$, excepto que estas curvas são agora parametrizadas por y, de forma que encontrar os pontos x_i onde elas cortam o eixo dos x é tão simples como calcular o valor da correspondente sub–expres–

são de (1.1) em y = 0. A aproximação de primeira ordem é a base do método de Newton–Raphson para o cálculo de raízes.

■ Cálculo de zeros: Significado Gráfico

```
X[n_, x₀ : _] := Normal[Series[g[y], {y, y₀, n}]] //. Drop[diffgr, -1] /. x → x₀ //. {g[y₀] → x₀, g[y] → x, y₀ → f[x₀]}

Xr[n_, x₀ : _] :=
  Xr[n, x₀] = Normal[Series[g[y], {y, y₀, n}]] //. Drop[diffgr, -1] /. x → x₀ //. {g[y₀] → x₀, g[y] → x, y₀ → f[x₀], y → 0}

Xn[n_, x₀ : _, m_] := Nest[Xr[n, #] &, x₀, m]
```

```
(x₁ ≈ Xr[#, x₀]) & /@ Range[4] // TableForm
```

$$x_1 \approx x_o - \frac{f[x_o]}{f'[x]}$$

$$x_1 \approx x_o - \frac{f[x_o]}{f'[x]} - \frac{f[x_o]^2 f''[x]}{2 f'[x]^3}$$

$$x_1 \approx x_o - \frac{f[x_o]}{f'[x]} - \frac{f[x_o]^2 f''[x]}{2 f'[x]^3} - \frac{f[x_o]^3 \left(3 f''[x]^2 - f'[x] f^{(3)}[x]\right)}{6 f'[x]^5}$$

$$x_1 \approx x_o - \frac{f[x_o]}{f'[x]} - \frac{f[x_o]^2 f''[x]}{2 f'[x]^3} - \frac{f[x_o]^3 \left(3 f''[x]^2 - f'[x] f^{(3)}[x]\right)}{6 f'[x]^5} + \frac{f[x_o]^4 \left(-15 f''[x]^3 + 10 f'[x] f''[x] f^{(3)}[x] - f'[x]^2 f^{(4)}[x]\right)}{24 f'[x]^7}$$

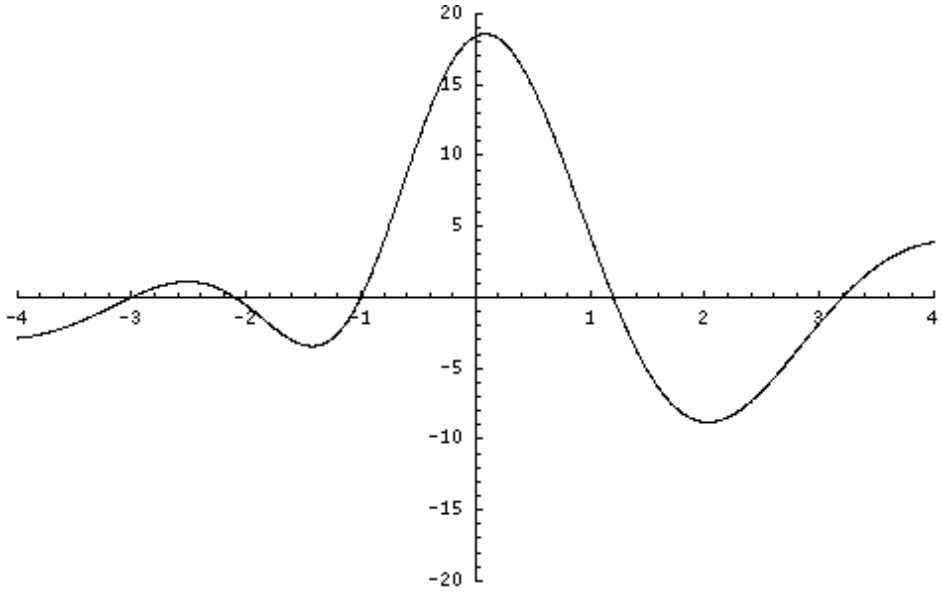
Desenhe uma curva paramétrica da aproximação de ordem n à função f(x) começando em $P_o = \{x_o, f(x_o)\}$.

```
nrord[n_, x₀ : _] := PP[{X[n, x₀], y}, {y, Sequence @@ Sort[{f[x₀], 0}]},
    PlotRange → {{-2, -1}, {-1, 1}}, PlotStyle → {Thickness[0.005], Hue[n/4]},
    DisplayFunction → Identity, PlotPoints → 500] /. f → ϕ /. PP → ParametricPlot;
```

□ Exemplo I

```
ϕ[x_] := Sin[x - x^2]
```

```
ϕ[x_] := (x - 1.2) (x + 2.1) (x - 3.2) (x + 3) Tanh[x + 1] e^(-.2 x^2)
grf = Plot[{ϕ[x]}, {x, -4, 4}, PlotRange → {4 {-1, 1}, 20 {-1, 1}}, PlotPoints → 1500];
```



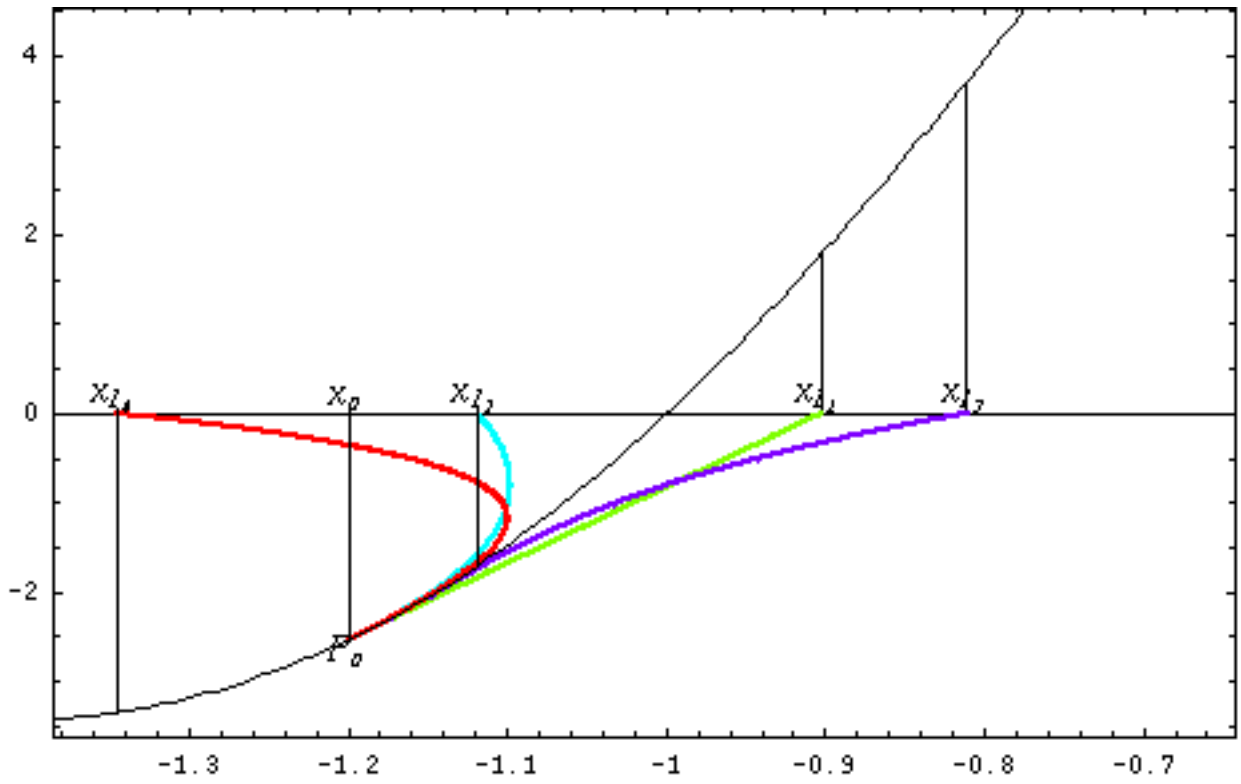
?? Subscript

Subscript[x, y] is an object that formats as x with a subscript y. More…

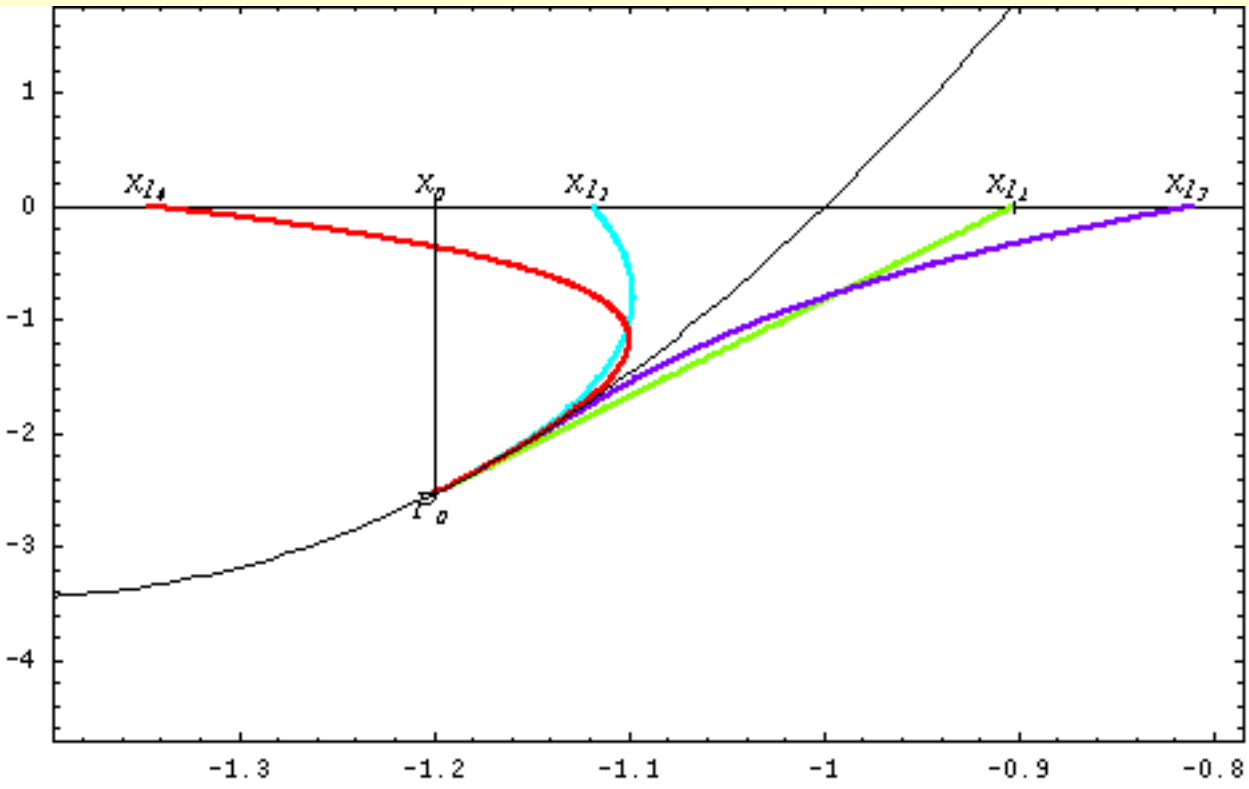
Subscript[ti, z_] := Sequence @@ {FontFamily → Times, FontSlant → Italic, FontSize → z}

Subscript[tb, z_] := Sequence @@ {FontFamily → Times, FontWeight → Bold, FontSize → z}

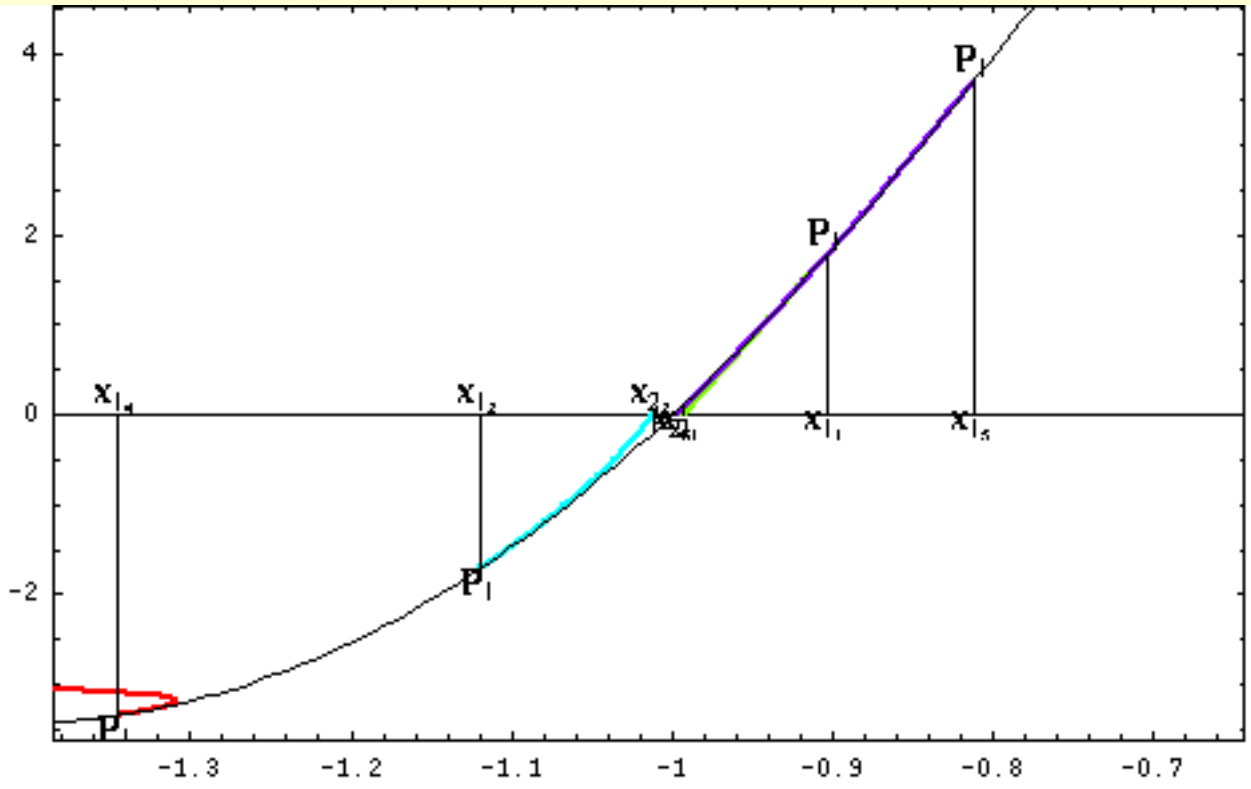
```
x0 = −1.2;
itr1 = (
Show[#1, DisplayFunction → (Display[$Display, #1] &), Epilog → (
{
Line[{{x₀, .05}, {x₀, ϕ[x₀]}}], Line[{{ξ, .05}, {ξ, ϕ[ξ]}}],
Text[StyleForm[P₀, ti₁₄], {x₀, −.15 + ϕ[x₀]}],
Text[StyleForm["x₀", ti₁₄], {x₀, .15}],
Text[StyleForm[x₁#1 , ti₁₄], {ξ, .15}]
} /. ξ → Xr[#1, x₀] /. x₀ → x0 /. f → ϕ &) /@ Range[4]] &)[
(Show[{nrord[#1, x0], grf}, PlotRange → {{−1.38496, −0.6428}, {−3.62336, 4.55795}},
Frame → True, Axes → {True, False}] &) /@ Range[4]];
```



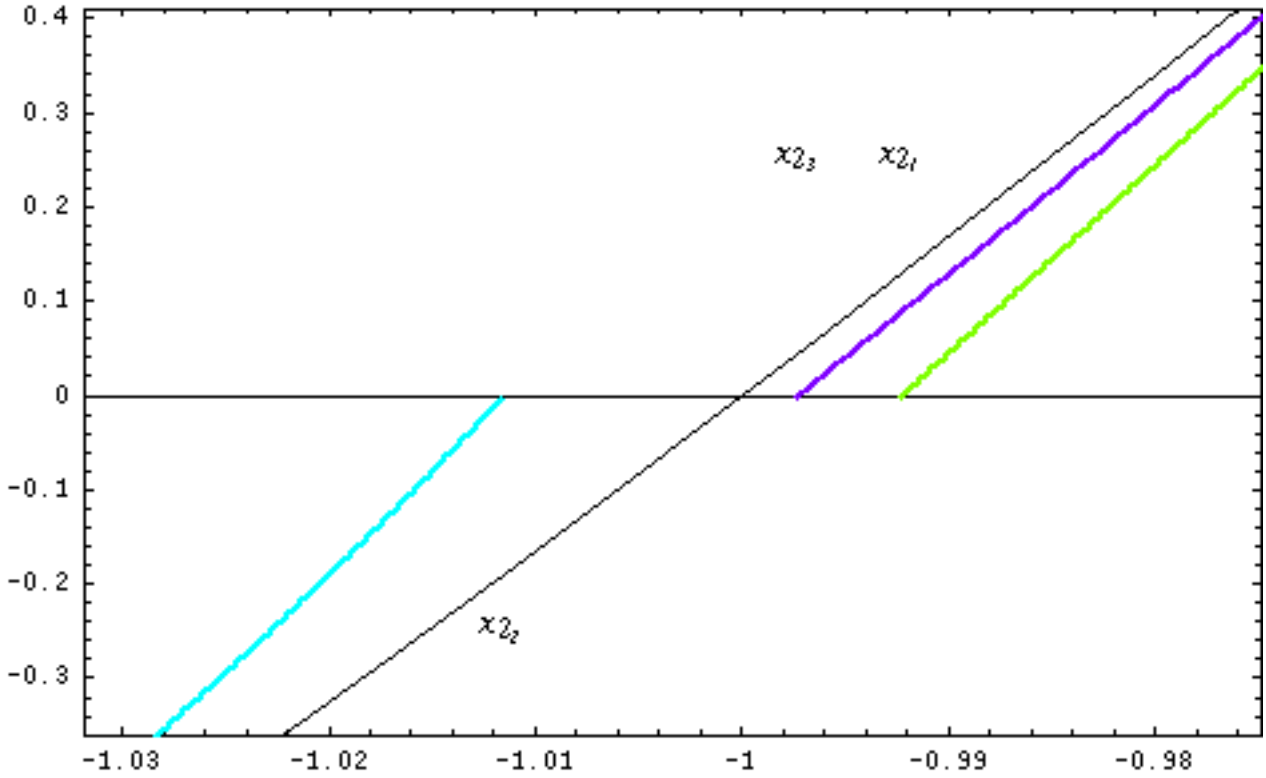

```
Show[%, PlotRange → {{−1.39532, −0.784518}, {−4.7181, 1.77418}}];
```



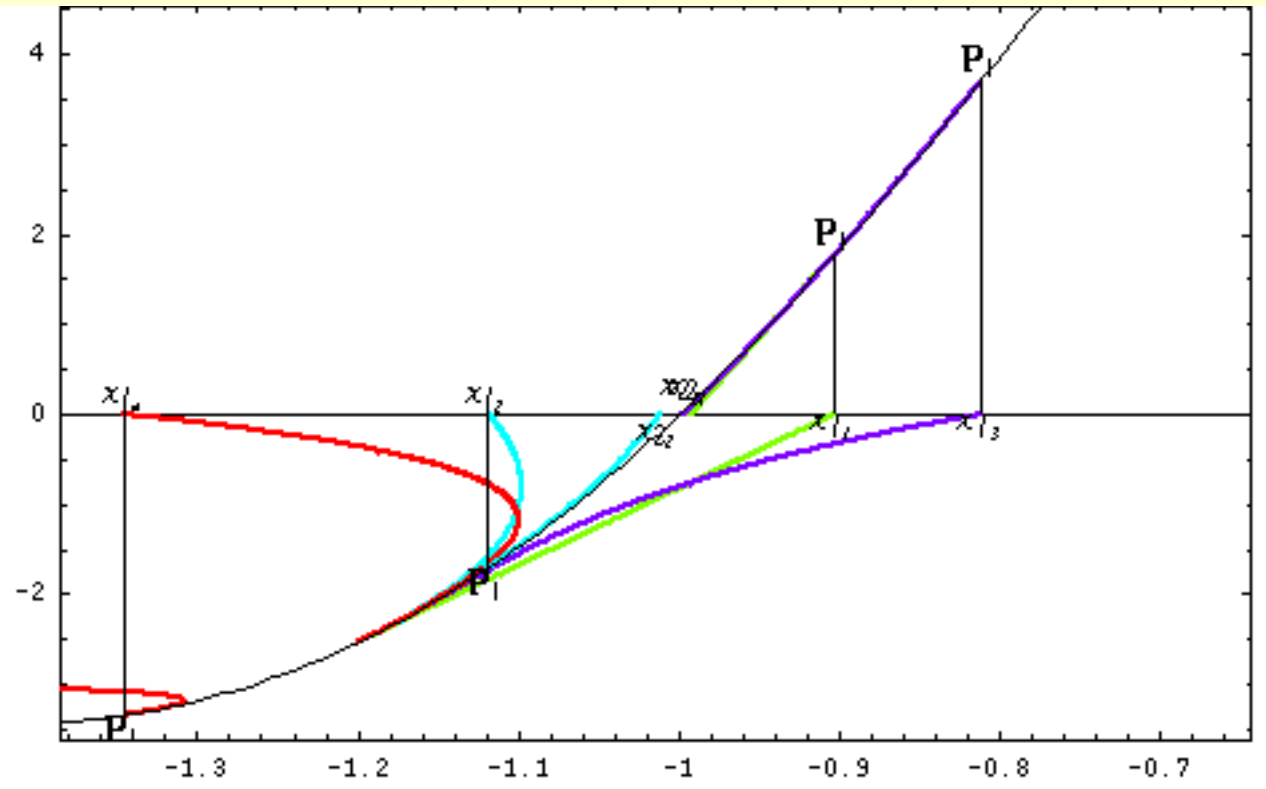
```
itr2 = (Show[#1, DisplayFunction → (Display[$Display, #1] &), Epilog → (
(σ = Sign[ϕ[Xr[#1, x_o]]];
{
Line[{{ξ, −σ .005}, {ξ, ϕ[ξ]}}],
Line[{{Xr[#1, ξ], −σ .005}, {Xr[#1, ξ], ϕ[Xr[#1, ξ]]}}],
Text[SF[P_1, tb_14], {ξ, σ .2 + ϕ[ξ]}],
Text[SF[x_1#, tb_14], {ξ, −σ .15}],
Text[SF[x_2#, tb_14], {Xr[#1, ξ], −σ .15}]
} /. ξ → Xr[#1, x_o] /. x_o → x0 /. f → ϕ) &) /@ Range[4]] &)[
Show[{nrord[#1, Xr[#1, x0] /. f → ϕ], grf},
        PlotRange → {{−1.38496, −0.6428}, {−3.62336, 4.55795}},
Axes → {True, False}, Frame → True] & /@ Range[4]];
```




```
(Show[%, PlotRange → #1] &)@{{−1.03181, −0.974932}, {−0.360815, 0.410412}}
```

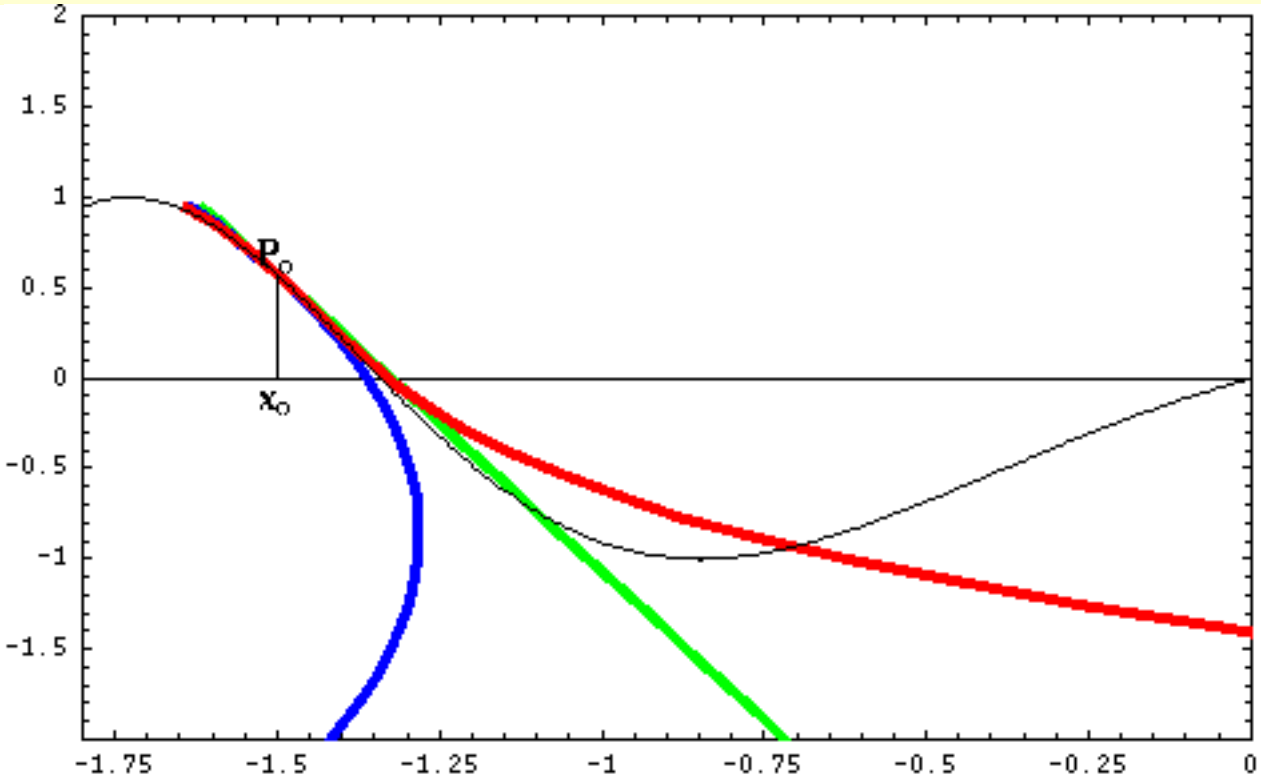



```
Show[{itr1, itr2},
  PlotRange → {{−1.38496, −0.6428}, {−3.62336, 4.55795}}, Axes → {True, False},
  Epilog → (
      (σ = Sign[ϕ[Xr[#1, x0]]]; {
          Line[{{Xr[#1, x0], .05}, {Xr[#1, x0], ϕ[Xr[#1, x0]]}}],
          Line[{{Xr[#1, x0], .05}, {Xr[#1, x0], σ .05}}],
          Text[StyleForm[P1, tb14], {Xr[#1, x0], σ .2 + ϕ[Xr[#1, x0]]}],
          Text[StyleForm[x1#, ti14], {Xr[#1, x0], −σ .15}],
          Text[StyleForm[x2#, ti14], {Xn[#1, x0, 2], σ .25}]
        } /. x0 → x0 /. f → ϕ) &) /@ Range[4]]
```

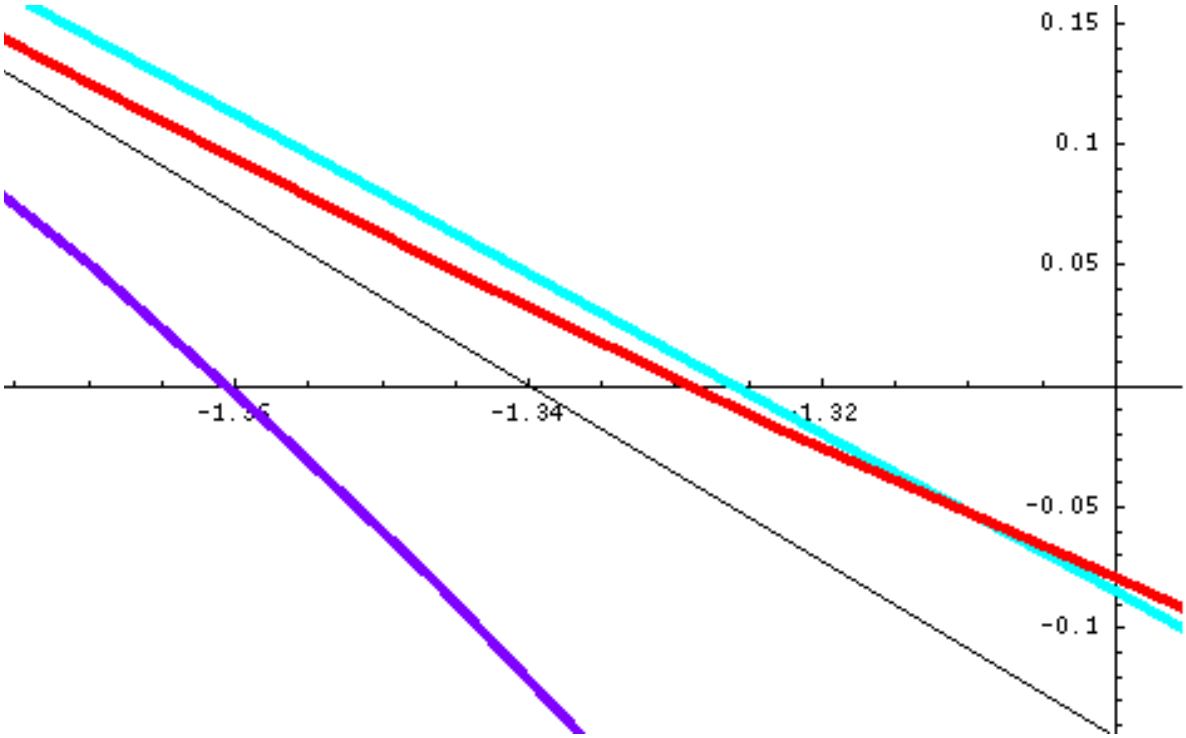


```
nrgrf = ParametricPlot[{X[#, x₀], y} //. f → ϕ /. x₀ → −1.5 // Release, {y, ϕ[−1.8], −2},
        PlotRange → {{−2, −1}, {−1, 1}}, PlotStyle → {Thickness[0.01], Hue[#/3]},
        DisplayFunction → Identity] & /@ Range[3];
```

```
Show[{nrgrf, grf},
  DisplayFunction → (Display[$Display, #1] &), PlotRange → {{−1.8, 0}, {−2, 2}},
  Epilog → {
      Line[{{x₀, 0}, {x₀, ϕ[x₀]}}],
      Text[StyleForm[P₀, tb₁₄], {x₀, .09 + ϕ[x₀]}],
      Text[StyleForm["x₀", tb₁₄], {x₀, −.15}]
    } /. x₀ → −1.5, Frame → True];
```

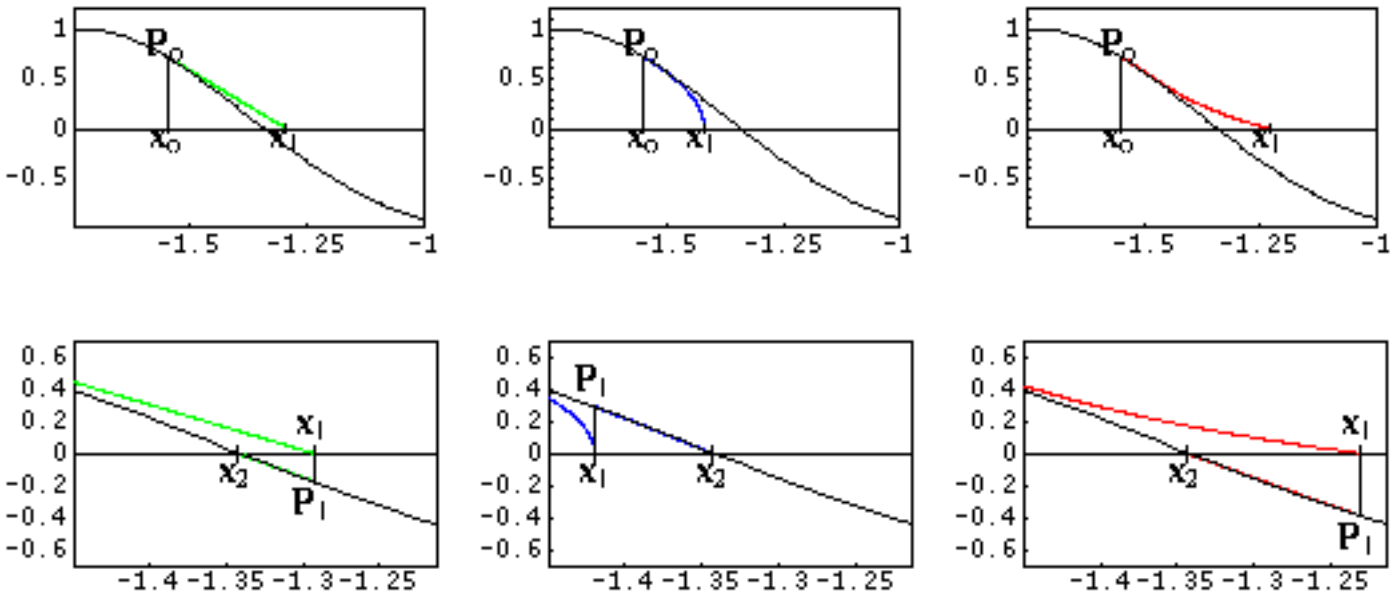



```
Show[%, PlotRange → {{−1.37577‘, −1.29547‘}, {0.157619‘, −0.142693‘}}]
```

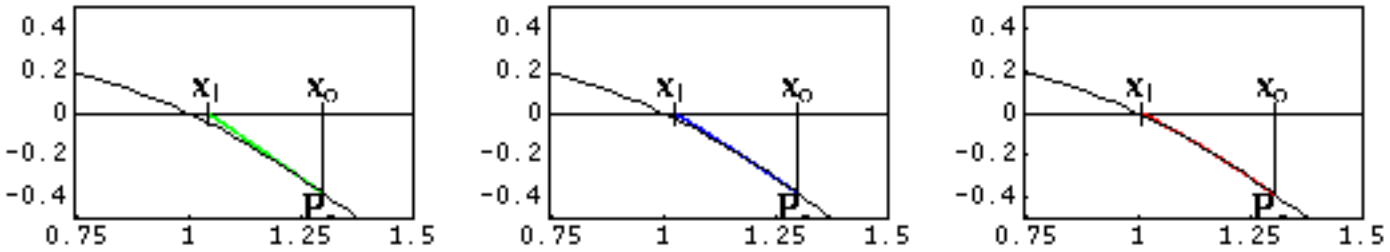



```
x0 = −1.55;
```

```
Show[{nrord[#, −1.55], grf},
        PlotRange → {{−1.75, −1}, {−1, 1.2}},
        FrameTicks → {Range[−2, −1, .25], Automatic, False, False},
        Epilog → {
                Line[{{x₀, −.05}, {x₀, ϕ[x₀]}}],
                Line[{{Xr[#, x₀], 0.05}, {Xr[#, x₀], −.05}}],
                Text[StyleForm[P₀, tb₁₄], {x₀, .09 + ϕ[x₀]}],
                Text[StyleForm["x₀", tb₁₄], {x₀, −.15}],
                Text[StyleForm[x₁, tb₁₄], {Xr[#, x₀], −.15}]
            } /. x₀ → −1.55 /. f → ϕ, Frame → True] & /@ Range[3] // GraphicsArray //
    Show[#, DisplayFunction → $DisplayFunction] &;
```




```
x0 = 1.3;
```




```
x0 = −12 / 10;
```

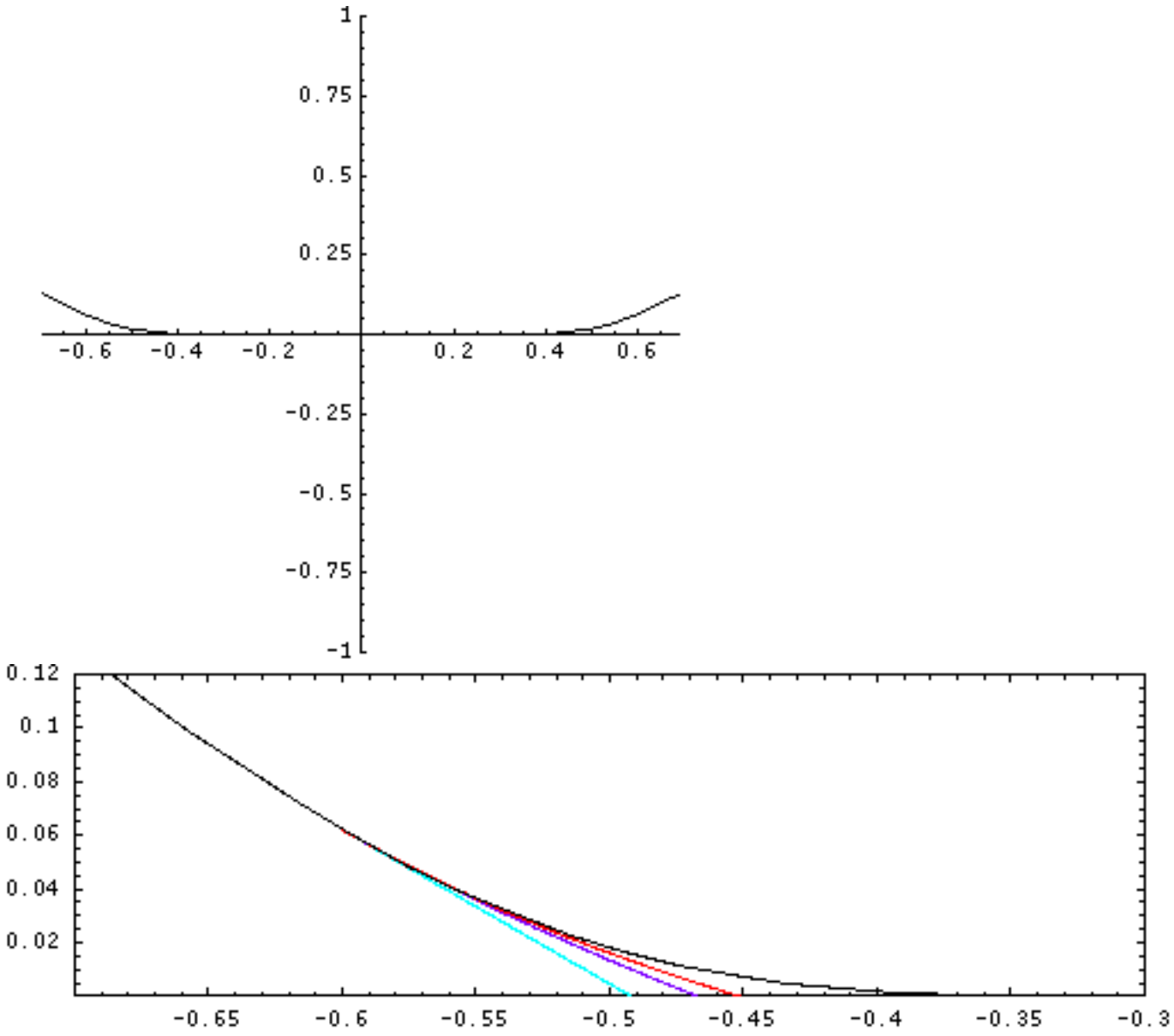
```
Print[
  SetAccuracy[x1 = N[x0, 30], 30], "\t",
  SetAccuracy[x2 = N[x0, 30], 30], "\t",
  SetAccuracy[x3 = N[x0, 30], 30], "\t",
  SetAccuracy[x4 = N[x0, 30], 30]]
For[n = 0, n < 6, n++,
Thread[{x1, x2, x3, x4} = ({Xr[1, x1], Xr[2, x2], Xr[3, x3], Xr[4, x4]} /. f → ϕ)];
Print[
    SetAccuracy[x1, 30], "\t",
    SetAccuracy[x2, 30], "\t",
    SetAccuracy[x3, 30], "\t",
    SetAccuracy[x4, 30]]]
```

```
-1.20000000000000000000000000000000   -1.20000000000000000000000000000000
-1.20000000000000000000000000000000   -1.20000000000000000000000000000000
-0.90294561832407316259718754736 2    -1.11875642150474652325442548317
-0.81146287574384268204141790192 8    -1.34564470414736403469646575104
-0.99228515215096702029029529513      -1.01154357881178014189060831995 4
-0.99728972591296805028093785950      -2947.61224581615169881843030452728 3
-0.99993520126966928263101408447 3    -1.00000505724283828001830443100
-0.99999999950843443397019427720 8    -2947.61401278475022991187870502471 9
-0.99999995313158307475021047139      -1.00000000000000004440892098500 6
-1.00000000000000000000000000000000   -2947.61577975228965442511253058910 4
-1.00000000000000000000000000000000   -1.00000000000000000000000000000000
-1.00000000000000000000000000000000   -2947.61754671876997235813178122043 6
-1.00000000000000000000000000000000   -1.00000000000000000000000000000000
-1.00000000000000000000000000000000   -2947.61931368419072896358557045459 7
```

□ EXEMPLO II

$\phi[x\_] := e^{\frac{-1}{x^2}}$

```
grf = Plot[ϕ[x], {x, −1, 1}, PlotRange → {−1, 1},
      AspectRatio → 1, PlotStyle → Thickness[0.003‘], PlotPoints → 100];
```



■ Polinómios Quadráticos

As raízes de polinómios do grau 2 e 3 podem ser determinadas por fórmulas conhecidas: por exemplo, no caso de equações quadráticas, existem três expressões possíveis, mas apenas a última é numéricamente estável quando $a_0 << a_1$.

```
Solve[x^2 + 2 a1 x + a0 == 0, x]
```

$x_- = -a_1 - \sqrt{a_1^2 - a_0}$	$x_- = \frac{a_0}{-a_1 - \sqrt{a_1^2 - a_0}}$	$x_1 = -a_1 - \text{Sign}[a_1]\sqrt{-a_0 + a_1^2}$
$x_+ = -a_1 + \sqrt{a_1^2 - a_0}$	$x_+ = \frac{a_0}{-a_1 + \sqrt{a_1^2 - a_0}}$	$x_2 = \frac{a_0}{-a_1 - \text{Sign}[a_1]\sqrt{-a_0 + a_1^2}}$

□ EXEMPLO: $x^2 + 4x + \frac{1}{9} = 0$

$$\begin{pmatrix} x_- \to -a_1 - \sqrt{-a_0 + a_1^2} \\ x_+ \to -a_1 + \sqrt{-a_0 + a_1^2} \end{pmatrix} \begin{pmatrix} x_- \to \frac{a_0}{-a_1 + \sqrt{-a_0 + a_1^2}} \\ x_+ \to \frac{a_0}{-a_1 - \sqrt{-a_0 + a_1^2}} \end{pmatrix} \begin{pmatrix} \text{"x}_1\text{"} \to -a_1 - \text{Sign}[a_1]\sqrt{-a_0 + a_1^2} \\ \text{"x}_2\text{"} \to \frac{a_0}{-a_1 - \text{Sign}[a_1]\sqrt{-a_0 + a_1^2}} \end{pmatrix} /.$$

$$\left\{a_1 \to 2, a_0 \to \frac{1}{9.}\right\} // \text{SetAccuracy}[\#1, 20] \& // \text{TableForm}$$

$x_- \to -3.9720265943665387098$	$x_- \to -3.9720265943665427066$	$x_1 \to -3.9720265943665387098$
$x_+ \to -0.0279734056334612902$	$x_+ \to -0.0279734056334613179$	$x_2 \to -0.0279734056334613179$

$$\text{Thread}\left[\left(\begin{pmatrix} x_-^2 + a_0 + 2x_- a_1 \\ x_+^2 + a_0 + 2x_+ a_1 \end{pmatrix} /. \begin{pmatrix} a_1 \to 2 \\ a_0 \to \frac{1}{9} \end{pmatrix}\right) ==$$

$$\left(\begin{pmatrix} x_-^2 + a_0 + 2x_- a_1 \\ x_+^2 + a_0 + 2x_+ a_1 \end{pmatrix} //. \begin{pmatrix} x_- \to -a_1 - \sqrt{-a_0 + a_1^2} \\ x_+ \to -a_1 + \sqrt{-a_0 + a_1^2} \\ a_1 \to 2 \\ a_0 \to \frac{1}{9} \end{pmatrix} // \text{N}\right] // \text{TableForm}$$

$$\frac{1}{9} + 4x_- + x_-^2 == 0.$$
$$\frac{1}{9} + 4x_+ + x_+^2 == 1.08095 \times 10^{-16}$$

$$\left(\begin{pmatrix} x_-^2 + a_0 + 2x_- a_1 \\ x_+^2 + a_0 + 2x_+ a_1 \end{pmatrix} /. \begin{pmatrix} a_1 \to 2 \\ a_0 \to \frac{1}{9} \end{pmatrix}\right) == \text{N}\left[\begin{pmatrix} x_-^2 + a_0 + 2x_- a_1 \\ x_+^2 + a_0 + 2x_+ a_1 \end{pmatrix} //. \begin{pmatrix} x_- \to \frac{a_0}{-a_1 + \sqrt{-a_0 + a_1^2}} \\ x_+ \to \frac{a_0}{-a_1 - \sqrt{-a_0 + a_1^2}} \\ a_1 \to 2 \\ a_0 \to \frac{1}{9} \end{pmatrix}\right] // \text{Thread} //$$

TableForm

$$\frac{1}{9} + 4x_- + x_-^2 == 1.59872 \times 10^{-14}$$
$$\frac{1}{9} + 4x_+ + x_+^2 == 0.$$

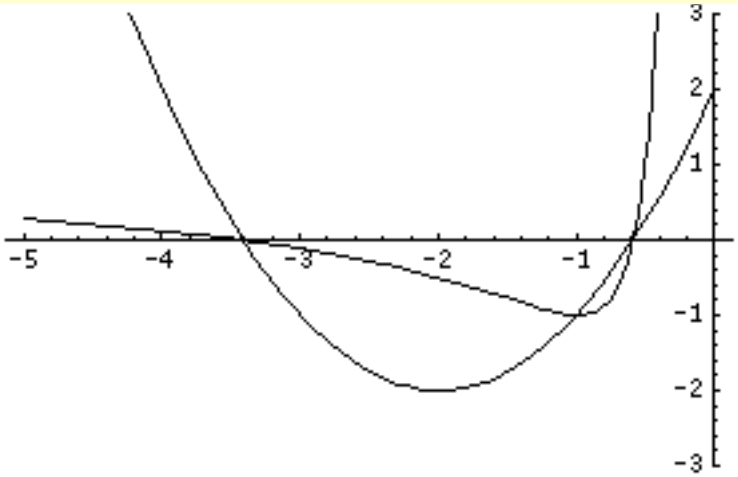
$$\left(\begin{pmatrix} a_0 + 2\,a_1\,x_1 + x_1^2 \\ a_0 + 2\,a_1\,x_2 + x_2^2 \end{pmatrix} /. \begin{pmatrix} a_1 \to 2 \\ a_0 \to \frac{1}{9} \end{pmatrix}\right) ==$$

$$N\left[\begin{pmatrix} a_0 + 2\,a_1\,x_1 + x_1^2 \\ a_0 + 2\,a_1\,x_2 + x_2^2 \end{pmatrix} //. \begin{pmatrix} x_1 \to -a_1 - \text{Sign}[a_1]\sqrt{-a_0 + a_1^2} \\ x_2 \to \frac{a_0}{-a_1 - \text{Sign}[a_1]\sqrt{-a_0 + a_1^2}} \\ a_1 \to 2 \\ a_0 \to \frac{1}{9} \end{pmatrix}\right] // \text{Thread} // \text{TableForm}$$

$$\frac{1}{9} + 4\,x_1 + x_1^2 == 0.$$
$$\frac{1}{9} + 4\,x_2 + x_2^2 == 0.$$

$x_- \to -3.9720265943665387098$	$x_- \to -3.9720265943665427066$	$x_1 \to -3.9720265943665387098$
$x_+ \to -0.027973405633461 2902$	$x_+ \to -0.027973405633461 3179$	$x_2 \to -0.027973405633461 3179$
$x_-^2 + 4\,x_- + \frac{1}{3} = 0.$	$x_-^2 + 4\,x_- + \frac{1}{9} = 1.6 \times 10^{-14}$	$x_1^2 + 4\,x_1 + \frac{1}{9} = 0.$
$x_+^2 + 4\,x_+ + \frac{1}{3} = 1.1 \times 10^{-16}$	$x_+^2 + 4\,x_+ + \frac{1}{9} = 0.$	$x_2^2 + 4\,x_2 + \frac{1}{9} = 0.$

$$\text{Plot}\left[\left\{x^2 + 4\,x + 2,\ \frac{1}{x^2}\left(x^2 + 4\,x + 2\right)\right\},\ \{x,\ -5,\ 0\},\ \text{PlotRange} \to 3\,\{-1,\ 1\}\right]$$



- Graphics -

■ Polinómio de grau 3

No caso cúbico, as raízes de $x^3 + 3\,a_2\,x^2 + 3\,a_1\,x + a_0 = 0$ podem ser encontradas através do seguinte processo: designando os três valores de $\sqrt[3]{-1}$ por z_0, z_-, z_+, onde

$$\text{Solve}\left[x^3 == -1,\ x\right] /.\ z\_\text{Rule} :\to (\text{Fc}\ /@\ \text{ComplexExpand}\ /@\ z)\ //\ \text{Flatten}$$

$$\left\{x \to -1,\ x \to \frac{1}{2}\left(1 + i\sqrt{3}\right),\ x \to \frac{1}{2}\left(1 - i\sqrt{3}\right)\right\}$$

$z_0 = e^{i\,\pi} = -1$	$z_- = e^{-i\,\pi/3} = \frac{1}{2}\left(1 - i\sqrt{3}\right)$	$z_+ = e^{i\,\pi/3} = \frac{1}{2}\left(1 + i\sqrt{3}\right)$

```
{(e^i)^{Arg[-1], Arg[1/2 (1 - i Sqrt[3])], Arg[1/2 (1 + i Sqrt[3])]}} // TableForm
```

podemos criar dois coeficientes $Q(a_1, a_2)$ e $P(a_0, a_1, a_2)$ definidos por

$$Q = 2^4 \times 3^2 \left(a_1 - a_2^2\right) \quad ; \quad P = -2^5 \times 3^3 \left(a_0 - 3\,a_1\,a_2 + 2\,a_2^3\right)$$

```
Solve[x^3 + 3 a2 x^2 + 3 a1 x + a0 == 0, x]
```

$$x \to -a_2 - \frac{2^{1/3}\,(9\,a_1 - 9\,a_2^2)}{3\left(-27\,a_0 + 81\,a_1\,a_2 - 54\,a_2^3 + \sqrt{4\,(9\,a_1 - 9\,a_2^2)^3 + (-27\,a_0 + 81\,a_1\,a_2 - 54\,a_2^3)^2}\,\right)^{1/3}} +$$

$$+ \frac{\left(-27\,a_0 + 81\,a_1\,a_2 - 54\,a_2^3 + \sqrt{4\,(9\,a_1 - 9\,a_2^2)^3 + (-27\,a_0 + 81\,a_1\,a_2 - 54\,a_2^3)^2}\,\right)^{1/3}}{3\,2^{1/3}}$$

$$x \to -a_2 + \frac{\left(1 + i\sqrt{3}\right)(9\,a_1 - 9\,a_2^2)}{3\,2^{2/3}\left(-27\,a_0 + 81\,a_1\,a_2 - 54\,a_2^3 + \sqrt{4\,(9\,a_1 - 9\,a_2^2)^3 + (-27\,a_0 + 81\,a_1\,a_2 - 54\,a_2^3)^2}\,\right)^{1/3}} -$$

$$- \frac{\left(1 - i\sqrt{3}\right)\left(-27\,a_0 + 81\,a_1\,a_2 - 54\,a_2^3 + \sqrt{4\,(9\,a_1 - 9\,a_2^2)^3 + (-27\,a_0 + 81\,a_1\,a_2 - 54\,a_2^3)^2}\,\right)^{1/3}}{6\,2^{1/3}}$$

$$x \to -a_2 + \frac{\left(1 - i\sqrt{3}\right)(9\,a_1 - 9\,a_2^2)}{3\,2^{2/3}\left(-27\,a_0 + 81\,a_1\,a_2 - 54\,a_2^3 + \sqrt{4\,(9\,a_1 - 9\,a_2^2)^3 + (-27\,a_0 + 81\,a_1\,a_2 - 54\,a_2^3)^2}\,\right)^{1/3}} -$$

$$- \frac{\left(1 + i\sqrt{3}\right)\left(-27\,a_0 + 81\,a_1\,a_2 - 54\,a_2^3 + \sqrt{4\,(9\,a_1 - 9\,a_2^2)^3 + (-27\,a_0 + 81\,a_1\,a_2 - 54\,a_2^3)^2}\,\right)^{1/3}}{6\,2^{1/3}}$$

```
(Solve[x^3 + 3 a2 x^2 + 3 a1 x + a0 == 0, x] /. (9 a1 - 9 a2^2) -> Q/16 //. (-27 a0 + 81 a1 a2 - 54 a2^3) -> P/32) /.
  {1 + i Sqrt[3] -> 2 z+, -1 - i Sqrt[3] -> -2 z+, 1 - i Sqrt[3] -> 2 z-, i (i + Sqrt[3]) -> -2 z-} // TableForm
```

$$x \to \frac{-Q + \left(P + \sqrt{P^2 + Q^3}\,\right)^{2/3}}{12\left(P + \sqrt{P^2 + Q^3}\,\right)^{1/3}} - a_2$$

$$x \to \frac{-\left(P + \sqrt{P^2 + Q^3}\,\right)^{2/3} z_- + Q\,z_+}{12\left(P + \sqrt{P^2 + Q^3}\,\right)^{1/3}} - a_2$$

$$x \to \frac{Q\,z_- - \left(P + \sqrt{P^2 + Q^3}\,\right)^{2/3} z_+}{12\left(P + \sqrt{P^2 + Q^3}\,\right)^{1/3}} - a_2$$

As raízes do polinómio podem agora escrever–se, fazendo as substituições referidas e simplificando,

$$x_o = -a_2 + \frac{1}{12}\left(\frac{Q}{\left(P+\sqrt{Q^3+P^2}\right)^{1/3}} z_o - \left(P+\sqrt{Q^3+P^2}\right)^{1/3} (z_o)^*\right)$$

$$x_+ = -a_2 + \frac{1}{12}\left(\frac{Q}{\left(P+\sqrt{Q^3+P^2}\right)^{1/3}} z_+ - \left(P+\sqrt{Q^3+P^2}\right)^{1/3} (z_+)^*\right)$$

$$x_- = -a_2 + \frac{1}{12}\left(\frac{Q}{\left(P+\sqrt{Q^3+P^2}\right)^{1/3}} z_- - \left(P+\sqrt{Q^3+P^2}\right)^{1/3} (z_-)^*\right)$$

Quando os coeficientes forem tais que P = 0 e Q < 0 , as três raízes são reais

$$x_o == \frac{1}{12}\sqrt{Q}\,(z_o - (z_o)^*) - a_2 \;== -a_2$$

$$x_+ == \frac{1}{12}\sqrt{Q}\,(z_+ - (z_+)^*) - a_2 \;== -a_2 + \frac{1}{6}\sqrt{3\,Q}\,i$$

$$x_- == \frac{1}{12}\sqrt{Q}\,(z_- - (z_-)^*) - a_2 \;== -a_2 - \frac{1}{6}\sqrt{3\,Q}\,i$$

Note que quando os coeficientes forem tais que $Q^3 = -P^2$ as três raízes são reais, mas duas são degeneradas:

$$x_o = -a_2 - \frac{1}{12}P^{1/3}\,(z_o + (z_o)^*) = -a_2 + \frac{1}{6}P^{1/3}$$

$$x_+ = -a_2 - \frac{1}{12}P^{1/3}\,(z_+ + (z_+)^*) = -a_2 - \frac{1}{12}P^{1/3}$$

$$x_- = -a_2 - \frac{1}{12}P^{1/3}\,(z_- + (z_-)^*) = -a_2 - \frac{1}{12}P^{1/3}$$

Solução aproximada de ODEs

■ Método de Euler

Como integrar aproximadamente uma equação diferencial ordinária:

$$\frac{dx(t)}{dt} == f(x,\, t)$$

com condição inicial

$$x(t_o) == x_o$$

Se soubéssemos a solução x(t) na vizinhança de x_o poderíamos desenvolvê–la em série de Taylor e guardar apenas o termo linear em t

```
x[δt] ≈ Series[x[t], {t, 0, 3}] /. x[0] → x0 /. t → δt /.
    Derivative[n_][x][0] :→ HoldForm[Dt[x, {t, n}]t=0] // TraditionalForm
```

$$x(\delta t) \approx x_0 + \left(\frac{dx}{dt}\right)_{t=0} \delta t + \frac{1}{2}\left(\frac{d^2x}{dt^2}\right)_{t=0} \delta t^2 + \frac{1}{6}\left(\frac{d^3x}{dt^3}\right)_{t=0} \delta t^3 + O(\delta t^4)$$

$$x(t_o + \delta t) \approx x_o + \left(\frac{dx}{dt}\right)_{t=0} \delta t + O(\delta t^2)$$

$$x_1 = x(t_o + \delta t)$$
$$t_1 = t_o + \delta t$$

Como $\left(\frac{dx}{dt}\right)_{t=t_o} = f(x_o,\ t_o)$, podemos escrever

$$x_1 = x_o + f(x_o,\ t_o)\,\delta t$$
$$t_1 = \delta t$$

A seguir assumimos que $x(t_1) \approx x_1$ e usamos $\left(\frac{dx}{dt}\right)_{t=t_1} = f(x_1,\ t_1)$ para deduzir que

$$x(t_1 + \delta t) \approx x(t_1) + \left(\frac{dx}{dt}\right)_{t=t_1} \delta t + O(\delta t^2)$$

$$\Downarrow$$

$$x_2 = x(t_1 + \delta t)$$
$$t_2 = t_1 + \delta t$$

pelo que

$$x_2 = x_1 + f(x_1,\ t_1)\,\delta t$$

Em geral chegamos à expressão

$$x_{k+1} = x_k + f(x_k,\ t_k)\,\delta t$$
$$t_{k+1} = t_k + \delta t$$

□ EXEMPLO V

Como exemplo vejamos o caso de :

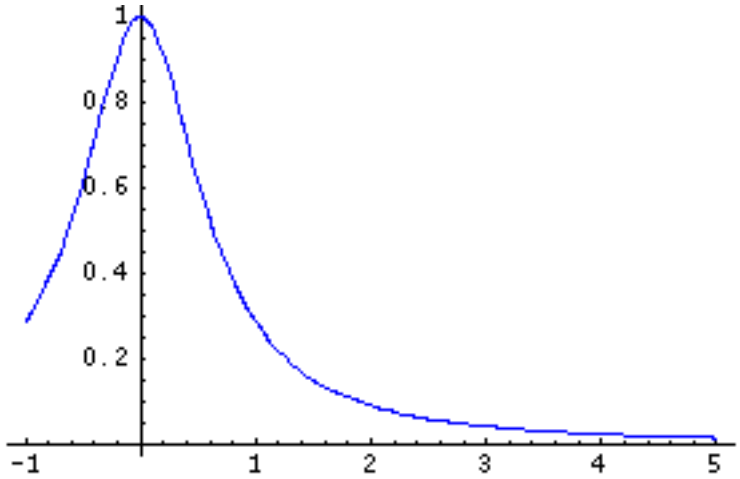
```
f[x_, t_] := -5 t x^2
```

Uma solução exacta existe para esta equação separável com condição inicial x[0] = 1:

```
X[t_] = (x[t] /. DSolve[{x'[t] == f[x[t], t], x[0] == 1} // ReleaseHold, x[t], t])[[1]]
```

$$\frac{2}{2 + 5\,t^2}$$

```
pl = Plot[X[t], {t, -1, 5}, PlotStyle → {Thickness[0.005'], RGBColor[0, 0, 1]}, PlotPoints → 100];
```



A função clr é definida aqui porque é necessário especificar que um subconjunto de definições atribuídas ao símbolo Subscript devem ser Unset. Todas as definições de funções como $x_{n\_}$ são atribuídas a Subscript. Fazer Clear[Subscript] apagaria mais do que seria desejável...

```
clr := {Off[Unset::"norep"];
ClearAll[t, x];
x_n_ =.;
t_n_ =.;
HoldForm[Unset[Subscript[x, #]]] & /@ Range[0, m] // ReleaseHold;
HoldForm[Unset[Subscript[t, #]]] & /@ Range[0, m] // ReleaseHold;};
```

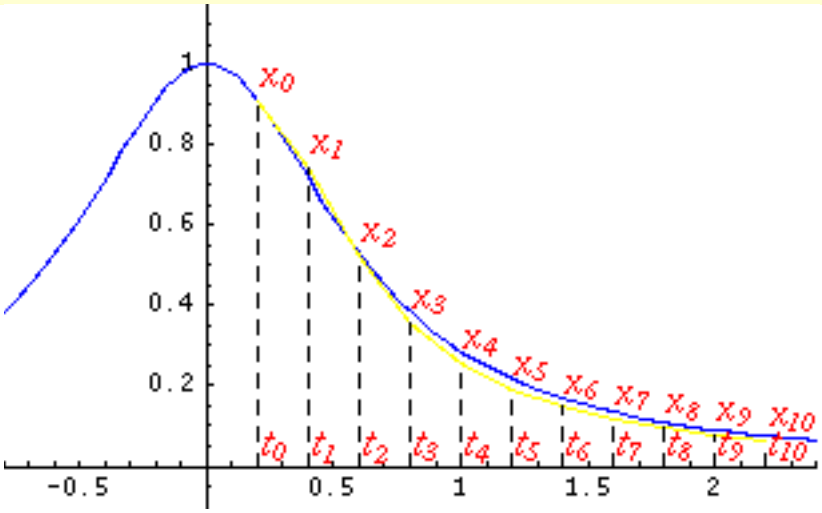
```
Definition[Subscript]
```

```
Subscript[ti, z_] := Sequence @@ {FontFamily → Times, FontSlant → Italic, FontSize → z}

Subscript[tb, z_] := Sequence @@ {FontFamily → Times, FontWeight → Bold, FontSize → z}

Subscript[fp, n_] := ListPlot[({Subscript[t, #1], Subscript[
                x, #1]} &) /@ Range[0, n], NoDspl, Axes → True]
```

Nas definições que implementam o método de Euler simples, as atribuições da forma $x_{n\_} := x_n = \text{def}[n]$ criam uma *memória* dos valores já calculados para x_k à medida que x_n é invocado para valores específicos de n = k . Da próxima vez que x_n for invocado com n = k, def[k] não é recalculado, e o x_k memorizado é devolvido. A seguir $lp_{n\_}$ implementa um ListPlot dos valores x_n calculados para n instantes t_n.

```
m = 10;
clr;

δt := .2;
t0 := 0.2;
t_n_ := t_n = t_{n-1} + δt;
x0 := x0 = X[t0];
x_n_ := x_n = x_{n-1} + f[x_{n-1}, t_{n-1}] δt

lp_n_ := ListPlot[{t#, x#} & /@ Range[0, n], DisplayFunction → Identity,
      Axes → True, PlotStyle → {Thickness[0.005'], RGBColor[1, 1, 0]}];
```

```
eul1 = Show[{ pl, lp_m },
    DisplayFunction → $DisplayFunction,
    PlotRange → {{-.8, (m + 2) δt}, {-.1, 1.15}}, ImageSize → 420, Axes → {True, True},
    Epilog → {{AbsoluteDashing[{5, 5, 5}], Line[{{t_#, 0}, {t_#, x_#}}] & /@ Range[0, m]},
        RGBColor[1, 0, 0],
        Txt[ti_14] /@ {Sequence @@ ({ToString[t]_#, {t_#, 0}, {-1, -1}} & /@ Range[0, m]),
            Sequence @@ ({ToString[x]_#, {t_#, x_# + .01}, {-1, -1}} & /@ Range[0, m])}}];
```



■ O MÉTODO DE EULER MELHORADO

Notando que o integral da equação diferencial entre instantes t_k e t_{k+1} é a área sob $f(x, t)$

$$\int_{t_k}^{t_{k+1}} \frac{dx(t)}{dt}\, dt = x(t_{k+1}) - x(t_k) = \int_{t_k}^{t_{k+1}} f(x, t)\, dt$$

poderíamos dar uma melhor aproximação desta área fazendo

$$\int_{t_k}^{t_{k+1}} f(x, t)\, dt \approx \frac{1}{2}\left(f(x_{k+1}, t_{k+1}) + f(x_k, t_k)\right)(t_{k+1} - t_k)$$

$$\Downarrow$$

$$x_{k+1} = x_k + \frac{1}{2}\left(f(x_{k+1}, t_{k+1}) + f(x_k, t_k)\right)\delta t$$
$$t_{k+1} = t_k + \delta t$$

Resta agora substituir x_{k+1} no lado direito por uma primeira aproximação, que pode ser $\tilde{x}_{k+1} = x_k + f(x_k, t_k)\,\delta t$, donde temos

$$\tilde{x}_{k+1} = x_k + f(x_k, t_k)\,\delta t$$

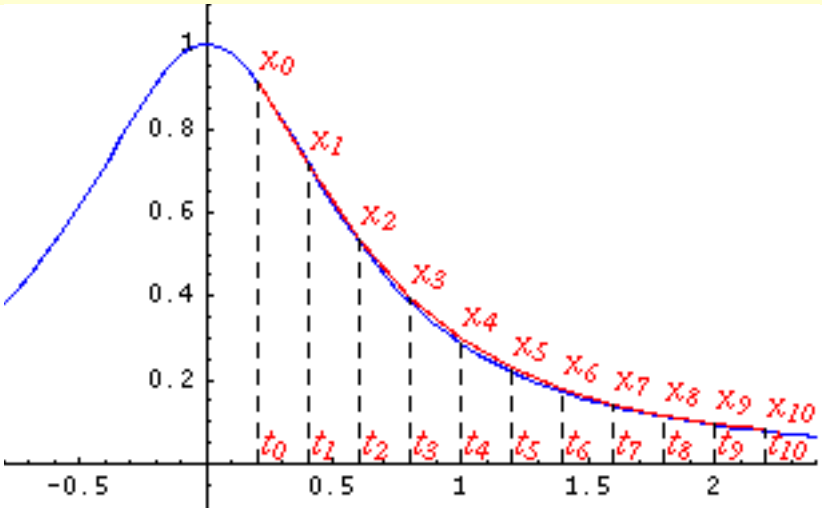
$$x_{k+1} = x_k + \frac{1}{2}\left(f(\tilde{x}_{k+1}, t_{k+1}) + f(x_k, t_k)\right)\delta t$$

$$t_{k+1} = t_k + \delta t$$

A implementação do método melhorado é semelhante à anterior:

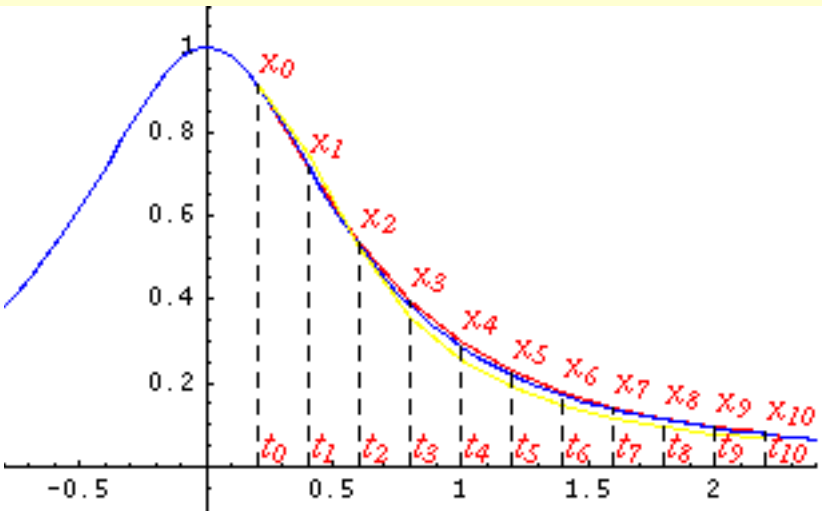
```
m = 10;
clr;
δt := .2;
t0 := 0.2;
tn_ := tn = tn−1 + δt;
x0 := x0 = X[t0];
xn_ := xn = xn−1 + 1/2 (f[xn−1 + f[xn−1, tn−1] δt, tn] + f[xn−1, tn−1]) δt

lpn_ := ListPlot[{t#, x#} & /@ Range[0, n], DisplayFunction → Identity,
        Axes → True, PlotStyle → {Thickness[0.005'], RGBColor[1, 0, 0]}];
```

```
eul2 = Show[{pl, lpm},
    DisplayFunction → $DisplayFunction,
    PlotRange → {{−.8, (m + 2) δt}, {−.1, 1.1}}, ImageSize → 420, Axes → {True, True},
    Epilog →
      {{AbsoluteDashing[{5, 5, 5}], Line[{{t#, 0}, {t#, x#}}] & /@ Range[0, m]}, RGBColor[1, 0, 0],
        Txt[ti14] /@ {Sequence @@ ({ToString[t]#, {t#, 0}, {−1, −1}} & /@ Range[0, m]),
            Sequence @@ ({ToString[x]#, {t#, x# + .01}, {−1, −1}} & /@ Range[0, m])}}];
```



A comparação dos dois métodos é feita sobrepondo os gráficos eul1 e eul2.

```
Show[{eul2, eul1}];
```



■ Método de Runge-Kutta

💡 Usando o método de Euler, faz–se uma estimativa do ponto $\tilde{x}_1 = x_o + f[t_o, x_o]\,\frac{1}{2}\,\delta t$ a meio do passo δt, i.e. no instante $\tilde{t}_1 = t_o + \frac{1}{2}\,\delta t$.

💡💡 O valor da derivada aí é aproximado por $f[\tilde{t}_1,\ \tilde{x}_1]$. Usando esta derivada para substituir no método de Euler obtém–se um novo ponto $x_1 = x_o + f[\tilde{t}_1,\ \tilde{x}_1]\,\delta t$

Este método tem erros de ordem $O[\delta t^3]$. De facto, da expansão em série de f[t, x] na vizinhança de $t_o,\ x_o$ obtém–se

$$f[\tilde{t}_1,\ \tilde{x}_1] = f\left[t_o + \frac{1}{2}\,\delta t,\ x_o + f[x_o, t_o]\,\frac{1}{2}\,\delta t\right] \approx f[t_o,\ x_o] + \left(\frac{df}{dt}\right)_{t=t_o}\,\frac{1}{2}\,\delta t + O[\delta t^2]$$

i.e. $\left(\frac{d^2x}{dt^2}\right)_{t=t_o} == \frac{2}{\delta t}\,(f[\tilde{t}_1,\ \tilde{x}_1] - f[t_o,\ x_o]) + O[\delta t]$. Substituindo na expressão

```
x[t0 + δt] ≃ (          Series[x[t], {t, t0, 2}]          //.
      {{t0 → 0}, {t → δt}, {x[0] → x0}, {x^(n_)[0] :→ HoldForm[(Dt[x, {t, n}])_(t=t0)]},
        {HoldForm[(Dt[x, {t, 1}])_(t=t0)] → f[t0, x0]}}) // TraditionalForm
```

$$x(\delta t + t_o) \simeq x_o + f(\delta t_o, x_o)\,\delta t + \frac{1}{2}\left(\frac{d^2x}{d\,\delta t^2}\right)_{\delta t=\delta t_o}\,\delta t^2 + O(\delta t^3)$$

conclui–se que

```
(% /. ((Dt[x, {t, 2}])_(t=t0)) → 2/δt (f[t̃1, x̃1] - f[t0, x0]) // Simplify) /. (4) → 3 // TraditionalForm
```

$$x(\delta t + t_o) \simeq x_o + f(\tilde{t}_1, \tilde{x}_1)\,\delta t + O(\delta t^3)$$

O método de Runge–Kutta de 4ª ordem deriva da mesma idéia fazendo mais passos intermédios

$\tilde{x}_1 = x_o + f[x_o, t_o]\,\frac{1}{2}\,\delta t$	$\tilde{x}_2 = x_o + f[\tilde{t}_1,\ \tilde{x}_1]\,\frac{1}{2}\,\delta t$	$\tilde{x}_3 = x_o + f[\tilde{t}_2,\ \tilde{x}_2]\,\delta t$
$\tilde{t}_1 = t_o + \frac{\delta t}{2}$	$\tilde{t}_2 = t_o + \frac{\delta t}{2}$	$\tilde{t}_3 = t_o + \delta t$

⇓

$x_1 = x_o + \frac{1}{6}\,f[x_o, t_o]\,\delta t + \frac{1}{3}\,f[\tilde{t}_1, \tilde{x}_1]\,\delta t + \frac{1}{3}\,f[\tilde{t}_2,\ \tilde{x}_2]\,\delta t + \frac{1}{6}\,f[\tilde{t}_3,\ \tilde{x}_3]\,\delta t$
$x\,(t_o + \delta t) \simeq x_1 + O[\delta t^5]$

■ Runge-Kutta Adaptativo

Para adaptar o passo δt às necessidades de precisão pode–se usar a seguinte técnica: cada etapa $t_k \to t_{k+1}$ do cálculo é efectuada uma segunda vez com dois passos de $\frac{1}{2}\delta t$, i.e $t_k \to t_{k+1/2} \to t_{k+1}$. Sendo um método de 4ª ordem, os erros nas aproximações são

$$x(t_k + \delta t) \simeq x_{k+1} + O[\delta t^5]$$
$$x\left(t_k + 2\,\frac{\delta t}{2}\right) \simeq \hat{x}_{k+1} + 2\,O\left[\left(\frac{\delta t}{2}\right)^5\right]$$

A diferença entre x_{k+1} e $\hat{x}_{k+1}$ dá uma estimativa do erro de truncação cometido $\Delta_k = \hat{x}_{k+1} - x_{k+1} \approx O[\delta t^5]$

Assim é de esperar que, para passos diferentes δt e $\tilde{\delta} t$ se obtenha uma relação entre passo e erro que se comporta como $\frac{\tilde{\delta} t}{\delta t} = \left(\frac{\tilde{\Delta}}{\Delta}\right)^{\frac{1}{5}}$

Assim, quando $\tilde{\Delta}$ é a precisão desejada, se o erro cometido no passo k é $\Delta_k > \tilde{\Delta}$ então o cálculo de x_{k+1} deve ser repetido com passo $\tilde{\delta} t = \delta t \left(\frac{\tilde{\Delta}}{\Delta_k}\right)^{0.2}$.

Se pelo contrário $\Delta_k < \tilde{\Delta}$, então o próximo ponto x_{k+2} pode ser calculado com um passo maior $\tilde{\delta} t = \delta t \left(\frac{\tilde{\Delta}}{\Delta_k}\right)^{0.2}$.

Transformações de Fourier

■ Integrais de Fourier

Inicializações

```
Get["Graphics`FilledPlot`"]
SetOptions[FilledPlot, Frame → True, FrameTicks → {Automatic, {0, 1, 2}, None, None},
    GridLines → Automatic, RotateLabel → False, DefaultFont → {"Helvetica", 10.}];
```

— *FilterOptions::shdw : Symbol FilterOptions appears in multiple contexts {Utilities`FilterOptions`, System`Convert`CommonDump`}; definitions in context Utilities`FilterOptions` may shadow or be shadowed by other definitions.*

```
Get["Utilities`Notation`"]
```

```
Symbolize[ℱ_^-1]
```

```
Symbolize[ℱ_x^-1 , WorkingForm → TraditionalForm]
```

```
SetOptions[FourierTransform, FourierParameters → {0, 2 π}];
SetOptions[InverseFourierTransform, FourierParameters → {0, 2 π}];
```

■ Definição

A transformada de Fourier duma função f $\in \mathcal{L}^2_{\mathbb{C}}(\mathbb{R})$ é definida por

$$\boxed{\mathcal{F}_k(f) = \int_{-\infty}^{+\infty} f[x]\, e^{2\pi i k x}\, dx \equiv F[k]}$$

sendo a transformação inversa

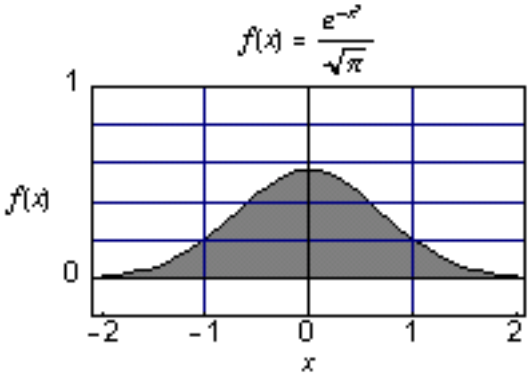
$$\boxed{\mathcal{F}_x^{-1}(F) = \int_{-\infty}^{+\infty} F[k]\, e^{-2\pi i k x}\, dk \equiv f[x]}$$

Aqui k $\in \mathbb{R}$ é uma frequência espacial ou número de ondas $\left(\mathrm{m}^{-1}\right)$. Substituindo x por tempo t, deve–se substituir k por uma frequência ν em unidades $\left(\mathrm{s}^{-1}\right)$.

■ Exemplo

A Gaussiana

```
f[x_] := 1/Sqrt[π] e^(-x^2)
```




```
Fgr1 = FilledPlot[f[x], {x, −2, 2}, PlotRange → {−.2, 1}, AspectRatio → 2.1/4,
       FrameLabel → ({x // TraditionalForm, f[x] // HoldForm // TraditionalForm}),
       PlotLabel → (HoldForm[f[x] = z] /. z → f[x] // TraditionalForm)];
```

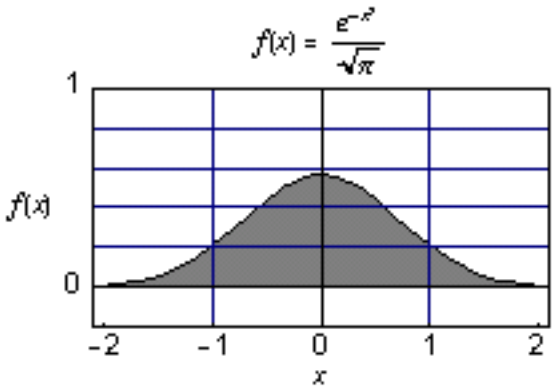
```
{{Area == ∫_-∞^∞ f[x] dx, ||f||^2 == ∫_-∞^∞ f[x]^2 dx}} // TableForm
```

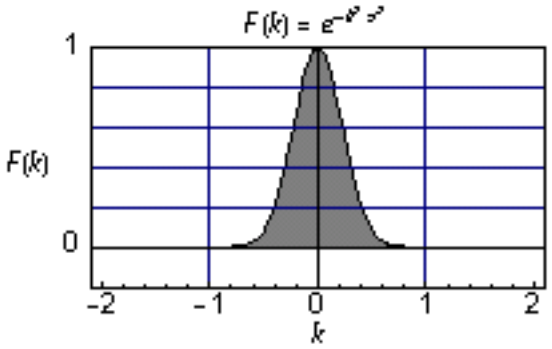
$\mathcal{A}rea == 1$ $\qquad$ $\|f\|^2 == \frac{1}{\sqrt{2\pi}}$

A Transformada de Fourier da Gaussiana é outra gaussiana

```
FourierTransform[f[x], x, k] == (F[k_] = FourierTransform[f[x], x, k])
```

$\mathcal{F}_k[f] == e^{-k^2 \pi^2}$






```
Fgr2 = FilledPlot[F[k], {k, −2, 2}, PlotRange → {−.2, 1}, AspectRatio → 2.1 / 4,
    FrameLabel → ({k // TraditionalForm, F[k] // HoldForm // TraditionalForm}), PlotLabel →
      (HoldForm[F[k] = z] /. z → F[k] // TraditionalForm), DisplayFunction → Identity];
Show[GraphicsArray[{Fgr1, Graphics[{Rectangle[{−2.2, −.8}, {2, 1.5}, Fgr2]},
      PlotRange → {{−2, 2}, {−.2, 1}}]}], DisplayFunction → (Display[{"stdout"}, #1] &)];
```

$$\left\{\left\{\mathcal{A}rea == \int_{-\infty}^{\infty} \mathrm{F[k]}\, d\mathrm{k},\ \|\mathrm{F}\|^2 == \int_{-\infty}^{\infty} \mathrm{F[k]}^2\, d\mathrm{k}\right\}\right\} \text{// TableForm}$$

$$\mathcal{A}rea == \frac{1}{\sqrt{\pi}} \qquad \|\mathrm{F}\|^2 == \frac{1}{\sqrt{2\,\pi}}$$

A transformada inversa dá a gaussiana original

$$\mathcal{F}_{\mathrm{k}}^{-1}[\mathrm{F}] == \text{InverseFourierTransform}\left[e^{-\mathrm{k}^2\,\pi^2},\ \mathrm{k},\ \mathrm{x}\right]$$

$$\mathcal{F}_{\mathrm{k}}^{-1}[\mathrm{F}] == \frac{e^{-\mathrm{x}^2}}{\sqrt{\pi}}$$

Propriedades da Transformada de Fourier

- **Linearidade:**

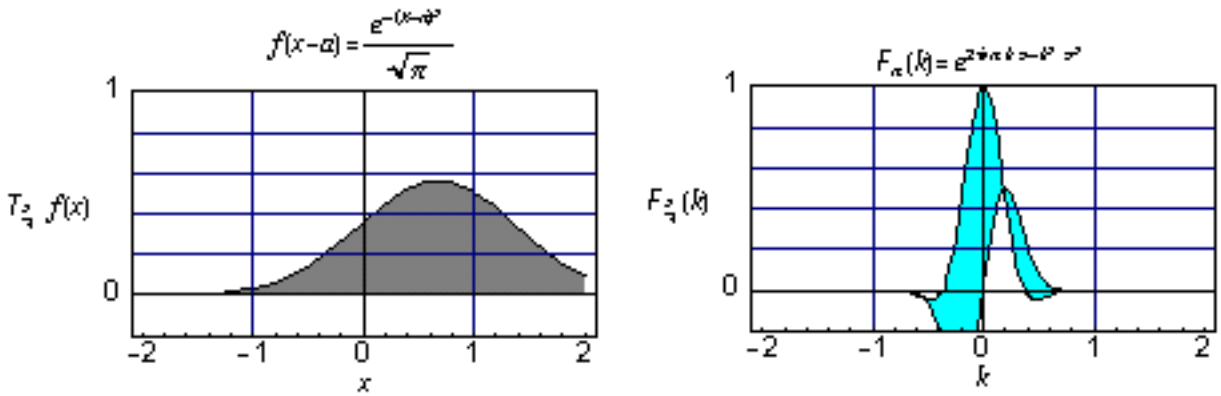
$$\mathcal{F}_k[A\,f + B\,g] = A\,\mathcal{F}_k[f] + B\,\mathcal{F}_k[g]$$

- **Translação:**

$$T_a\,f[x] = f[x-a] \implies \mathcal{F}_k[T_a\,f] = e^{2\,\pi\,i\,a\,k}\ \mathcal{F}_k[f]$$

$$\mathcal{F}_{\mathrm{k}}[\text{HoldForm}[\mathrm{T_a\ f}]] == (\mathrm{F_a[k]} = \text{FourierTransform}[\mathrm{f[x-a]},\ \mathrm{x},\ \mathrm{k}] \text{ // ExpandAll}) \text{ // TraditionalForm}$$

$$\mathcal{F}_{\mathrm{k}}\ (\mathrm{T_a\ f}) == e^{2\,i\,\mathrm{a\,k}\,\pi - \mathrm{k}^2\,\pi^2}$$



```
Fgr3 = FilledPlot[({F_a[k] /. a → 2/3 // Re, F_a[k] /. a → 2/3 // Im}),
        {k, −2, 2}, PlotRange → {−.2, 1}, AspectRatio → 2.1/4,
        FrameLabel → ({k // TraditionalForm, F_{2/3}[k] // HoldForm // TraditionalForm}), PlotLabel →
          (HoldForm[F_a[k] = z] /. z → (F_a[k]) // TraditionalForm), DisplayFunction → Identity];
```

```
Fgr4 = FilledPlot[f[x − a] /. a → 2/3, {x, −2, 2}, PlotRange → {−.2, 1}, AspectRatio → 2.1/4,
        FrameLabel → ({x // TraditionalForm, T_{2/3} f[x] // HoldForm // TraditionalForm}), PlotLabel →
          (HoldForm[f[x − a] = z] /. z → f[x − a] // TraditionalForm), DisplayFunction → Identity];
```

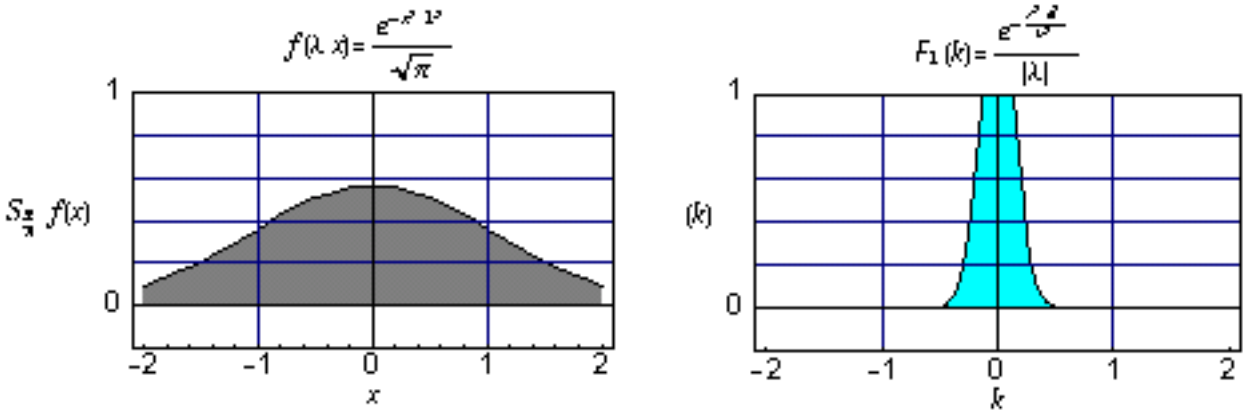
```
Show[GraphicsArray[
        {Fgr4, Graphics[{Rectangle[{−2.2, −.8}, {2, 1.5}, Fgr3]}, PlotRange → {{−2, 2}, {−.2, 1}}]} //
          Flatten], DisplayFunction → (Display[$Display, #1] &)];
```

■ DILATAÇÃO:

$$S_\lambda f[x] = f[\lambda x] \implies \mathcal{F}_k[S_\lambda f] == \frac{1}{|\lambda|}\mathcal{F}_{\frac{k}{\lambda}}[f]$$

```
F_k[HoldForm[S_λ f]] == (F_λ[k] = FourierTransform[f[λ x], x, k] // Simplify[#, λ ∈ Reals] &) //
  TraditionalForm
```

$$\mathcal{F}_k\,(S_\lambda\, f) == \frac{e^{-\frac{k^2\pi^2}{\lambda^2}}}{|\lambda|}$$



```
Fgr5 = FilledPlot[({Fλ[k] /. λ → 2/3 // Re, Fλ[k] /. λ → 2/3 // Im}),
    {k, −2, 2}, PlotRange → {−.2, 1}, AspectRatio → 2.1/4,
    FrameLabel → ({k // TraditionalForm, F2/3[k] // HoldForm // TraditionalForm}), PlotLabel →
      (HoldForm[Fλ[k] = z] /. z → (Fλ[k]) // TraditionalForm), DisplayFunction → Identity];
```

```
Fgr6 = FilledPlot[f[2/3 x], {x, −2, 2}, PlotRange → {−.2, 1}, AR → 2.1/4,
    FrameLabel → ({x // TraditionalForm, S2/3 f[x] // HoldForm // TraditionalForm}), PlotLabel →
      (HoldForm[f[λ x] = z] /. z → f[λ x] // TraditionalForm), DisplayFunction → Identity];
```

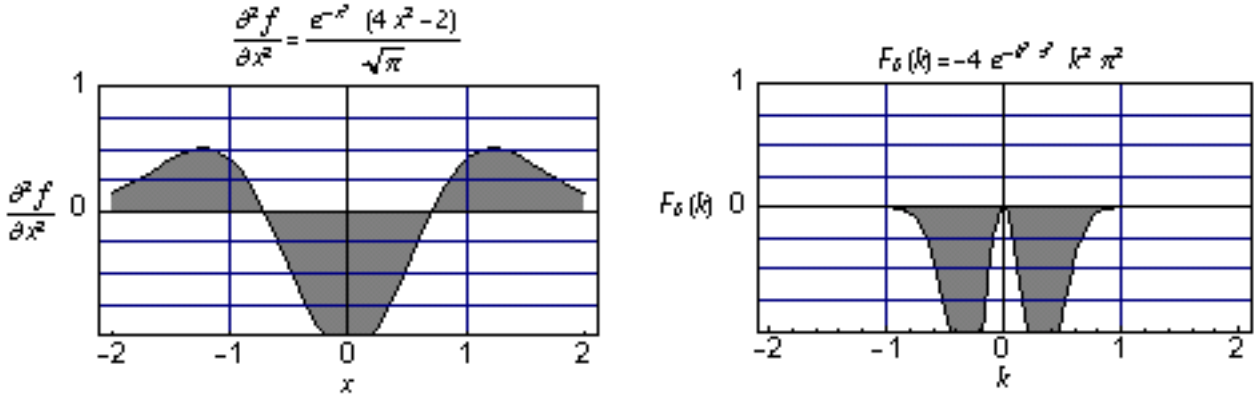
```
Show[GraphicsArray[
    {Fgr6, Graphics[{Rectangle[{−2.8, −1.0}, {2, 1.8}, Fgr5]}, PlotRange → {{−2, 2}, {−.2, 1}}]} //
      Flatten], DisplayFunction → (Display[$Display, #1] &)];
```

■ Derivação:

$$\mathcal{F}_k[\partial_x{}^n f] = (2\pi i k)^n \mathcal{F}_k[f]$$

```
ℱk[HoldForm[D[f, {x, 2}]]] ==
  (Fδ[k] = FourierTransform[D[f[x], {x, 2}], x, k] // Simplify[#, λ ∈ Reals] &) // TraditionalForm
```

$$\mathcal{F}_k\left(\frac{\partial^2 f}{\partial x^2}\right) == -4\, e^{-k^2\pi^2}\, k^2\pi^2$$




```
Fgr7 = FilledPlot[Fδ[k] // Re, {k, −2, 2}, PlotRange → {−1, 1}, AspectRatio → 2/4,
    FrameLabel → ({k // TraditionalForm, Fδ[k] // HoldForm // TraditionalForm}), PlotLabel →
      (HoldForm[Fδ[k] = z] /. z → (Fδ[k]) // TraditionalForm), DisplayFunction → Identity];
```

```
Fgr8 = FilledPlot[D[f[x], {x, 2}], {x, −2, 2}, PlotRange → {−1, 1}, AspectRatio → 2/4,
    FrameLabel → ({x // TraditionalForm, D[f, {x, 2}] // HoldForm // TraditionalForm}),
    PlotLabel → (HoldForm[D[f, {x, 2}] = z] /. z → (D[f[x], {x, 2}] // Simplify) // TraditionalForm),
    DisplayFunction → Identity];
```

```
Show[GraphicsArray[
    {Fgr8, Graphics[{Rectangle[{−2.4, −.8}, {2, 1.5}, Fgr7]}, PlotRange → {{−2, 2}, {−.2, 1}}]} //
      Flatten], DisplayFunction → (Display[$Display, #1] &)];
```

■ Convolução:

$$f * g[x] = \int_{-\infty}^{+\infty} f[\xi]\, g[x-\xi]\, d\xi \implies \mathcal{F}_k[f * g] = \mathcal{F}_k[f]\,\mathcal{F}_k[g]$$

□ Exemplo

Defina–se uma nova função g[x] = $e^{-(x-2)^2}$ sin(2 x):

```
g[x_] := Sin[2 x] e^-(x-2)^2
```

Pela definição, a convolução de g com f é:

$$f*g[x] == \frac{e^{-\frac{1}{2}(x-2)^2-\frac{1}{2}}\sin(x+2)}{\sqrt{2}}$$

```
"f*g"[x] == (((∫_-∞^+∞ f[ξ] g[x - ξ] dξ) Sqrt[e]/HoldForm[Sqrt[e]] // ExpToTrig // TrigReduce // Simplify // At[-1,
    TrigToExp] // At[-3, 2, Simplify[Together[#]] &] // ReleaseHold) // TraditionalForm
```

Se calcularmos a Transformada de Fourier de g[x] obtemos G[k] = $\mathcal{F}_k[g]$

```
FourierTransform_k[g] == (G[k] = FourierTransform[g[x], x, k] // FullSimplify) // TraditionalForm
```

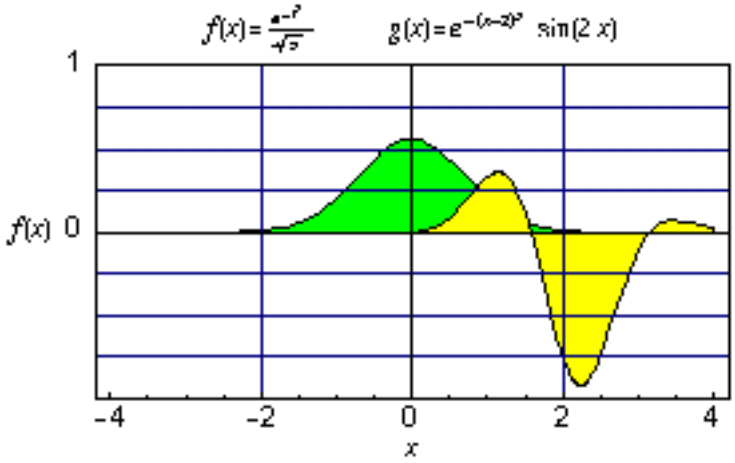
$$\mathcal{F}_k(g) == \frac{1}{2}\, i\, e^{-k\pi(\pi k+(2-4i))-(1+4i)}\left(-e^{8i}+e^{4k\pi}\right)\sqrt{\pi}$$

A transformada inversa do produto F[k] G[k] deve ser igual à convolução f ∗ g [x]

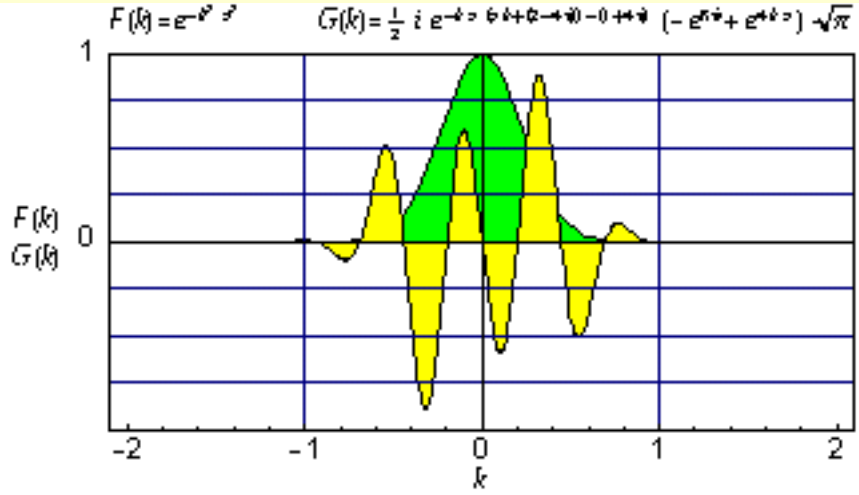
$$\mathcal{F}_x^{-1}(F\,G) == \frac{e^{-\frac{1}{2}(x-2)^2-\frac{1}{2}}\sin(x+2)}{\sqrt{2}}$$

```
InverseFourierTransform_x[F G] ==
    (fg[x] = InverseFourierTransform[F[k] G[k], k, x] Sqrt[e]/HoldForm[Sqrt[e]] // ExpToTrig // TrigReduce //
        Simplify // At[-1, TrigToExp] //
      At[-3, 2, Simplify[Together[#]] &] // ReleaseHold) // TraditionalForm
```

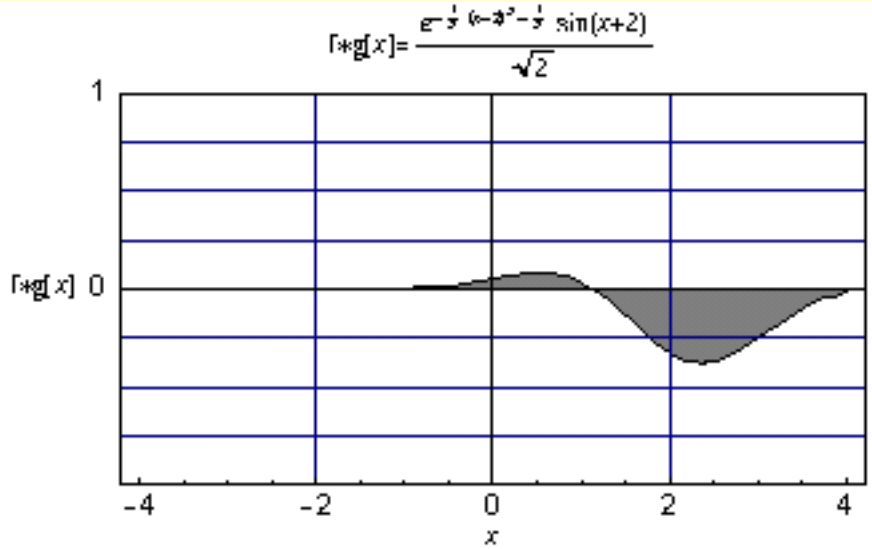
(a) – Gráficos




```
Fgr1 = FilledPlot[{f[x], g[x]}, {x, −4, 4}, PlotRange → {−1, 1}, AspectRatio → 2.1 / 4,
    FrameLabel → ({x // TraditionalForm, f[x] // HoldForm // TraditionalForm}), PlotLabel →
      ({HoldForm[f[x] = z] /. z → f[x], HoldForm[g[x] = z] /. z → g[x]} // List // TableForm //
        TraditionalForm), Fills → {{{1, Axis}, gr}, {{2, Axis}, RGBColor[1, 1, 0]}}];
```




```
Fgr9 = FilledPlot[{F[k], G[k] // Im}, {k, −2, 2},
    PlotRange → {−1, 1}, AspectRatio → 2 / 4, FrameLabel → ({k // TraditionalForm,
        {F[k] // HoldForm, G[k] // HoldForm} // TableForm // TraditionalForm}),
    PlotLabel → ({{HoldForm[F[k] = z] /. z → (F[k]), HoldForm[G[k] = z] /. z → (G[k])}} //
        TableForm // TraditionalForm), DisplayFunction →
      (Display[$Display, #1] &), Fills → {{{1, Axis}, gr}, {{2, Axis}, RGBColor[1, 1, 0]}}];
```



```
Fgr2 = FilledPlot[{fg[x]} // Chop, {x, −4, 4}, PlotRange → {−1, 1},
       AR → 2.1 / 4, FrameLabel → ({x // TraditionalForm, "f∗g"[x] // TraditionalForm}),
       PlotLabel → (HoldForm["f∗g"[x] = z] /. z → fg[x] // TraditionalForm)];
```

Correlação: $Corr[f, g][x] = \int_{-\infty}^{+\infty} f[\xi + x]\, g[\xi]\, d\xi \implies \mathcal{F}_k[Corr[f, g]] = \mathcal{F}_k[f]\, \mathcal{F}_k[g]^*$

Pela definição,

$$\text{Corr(f, g)} == \frac{e^{-\frac{1}{2}(x+2)^2-\frac{1}{2}}\sin(2-x)}{\sqrt{2}}$$

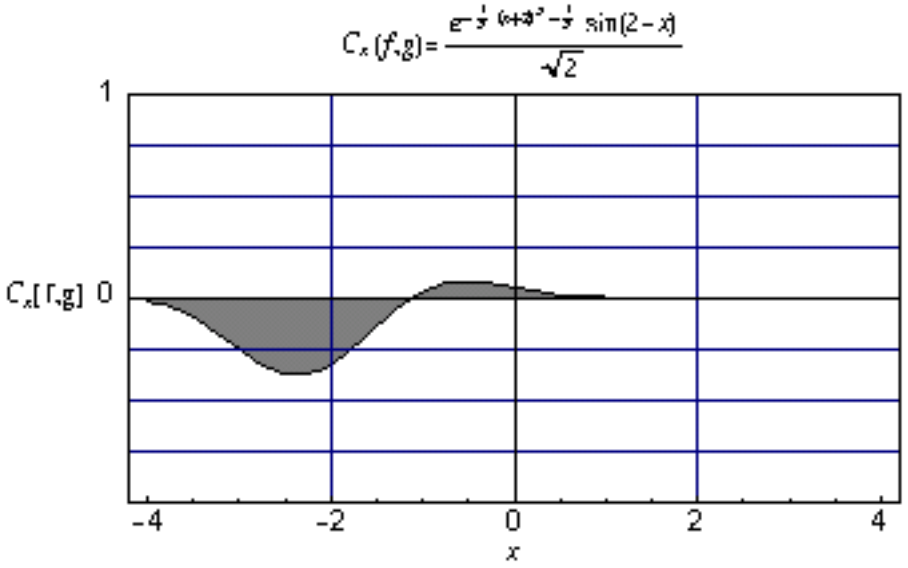
```
Corr[f, g] == (((∫_−∞^∞ f[x + ξ] g[ξ] dξ) √e / HoldForm[√e] // ExpToTrig // TrigReduce // Simplify // At[−1,
                TrigToExp] // At[−3, 2, Simplify[Together[#]] &] // ReleaseHold) // TraditionalForm
```

Usando InverseFourierTransform do produto F[k] G[k]*:

$$\mathcal{F}_x^{-1}(F\,G^*) == \frac{e^{-\frac{1}{2}(x+2)^2-\frac{1}{2}}\sin(2-x)}{\sqrt{2}}$$

```
ℱx^−1[F G*] == (Cx[f, g] = (InverseFourierTransform[F[k] Conjugate[G[k]]
                // Simplify[#, k ∈ Reals] & // Release, k, x]
                √e / HoldForm[√e] // Simplify[# // ExpToTrig, x ∈ Reals] &) /.
               z_Cosh :→ TrigReduce[z] /. z_Sinh :→ TrigReduce[z] // Simplify //
               At[−1, TrigToExp] // At[−3, 2, Simplify] // ReleaseHold) // TraditionalForm
```

(a) – Gráficos



```
Fgr2 = FilledPlot[{Cx[f, g]} // Release, {x, −4, 4}, PlotRange → {−1, 1}, AspectRatio → 2.1 / 4,
       FrameLabel → ({x // TraditionalForm, "Cx[f,g]" // TraditionalForm}),
       PlotLabel → (HoldForm[Cx[f, g] = z] /. z → Cx[f, g] // TraditionalForm)];
```

Auto−Correlação (Wiener−Khinchin): $\mathcal{F}_k[Corr[f, f]] = |\mathcal{F}_k[f]|^2$

Potência Espectral (Parseval): $\mathcal{P} = \int_{-\infty}^{+\infty} |f[x]|^2 \, dx = \int_{-\infty}^{+\infty} |\mathcal{F}_k[f]|^2 \, dk$

Séries de Fourier

□ DEFINIÇÃO

$$\boxed{f[x] = \frac{a_0}{2} + \sum_{n=1}^{\infty} \left(a_n \, Cos\left[\frac{n\pi}{L} x\right] + b_n \, Sin\left[\frac{n\pi}{L} x\right] \right)}$$

$$a_n = \frac{1}{L} \int_c^{c+2L} f[x] \, Cos\left[\frac{n\pi}{L} x\right] dx \qquad ; \qquad b_n = \frac{1}{L} \int_c^{c+2L} f[x] \, Sin\left[\frac{n\pi}{L} x\right] dx$$

$$\boxed{f[x] = \sum_{n=-\infty}^{\infty} c_n \, e^{i \frac{n\pi}{L} x}}$$

$$c_n = \frac{1}{L} \int_c^{c+2L} f[x] \, e^{-i \frac{n\pi}{L} x} \, dx = \begin{cases} \frac{1}{2}(a_n - i\, b_n) & (n > 0) \\ \frac{1}{2}(a_{-n} + i\, b_{-n}) & (n < 0) \\ \frac{1}{2} a_0 & (n = 0) \end{cases}$$

□ EXEMPLOS

```
<< Calculus`FourierTransform`
```

```
sw[b_] := FilledPlot[{2.5 / Abs[b] UnitStep[1 / (2 Abs[b]) − Abs[x]], −2.5 / Abs[b] UnitStep[1 / (2 Abs[b]) − Abs[x]]},
          {x, −1 / (2 Abs[b]), 1 / (2 Abs[b])}, GridLines → None, Frame → False,
          DisplayFunction → Identity] // (#1 /. #1[[1, 1]] :→ (Drop[#1, {1, 2}] &)[#1[[1, 1]]] &);
```

As definições aqui implementadas são:

(a) – $c_n = \int_{-1/2}^{1/2} f(t) e^{2\pi i n t} \, dt$ com FourierParameters → {0, 1} ou

(b) – $c_n = |b|^{(1-a)/2} \int_{-1/(2|b|)}^{1/(2|b|)} f(t) e^{2\pi i b n t} \, dt$ com FourierParameters → {a, b}.

As funções aqui presentes FourierSeries constroem aproximações da função f[x] a partir dos seus coeficientes de Fourier, i.e. implementam a *Transformada Inversa de Fourier* de $\mathcal{F}_k$[f]. As respectivas definições são

$$f[x] = \sum_{n=-k}^{k} c_n\, e^{-2\pi i n t} \qquad \text{com FourierParameters} \to \{0,\ 1\}$$

ou

$$f[x] = |b|^{(1+a)/2} \sum_{n=-k}^{k} c_n\, e^{-2\pi i b n t} \qquad \text{com FourierParameters} \to \{a,\ b\}$$

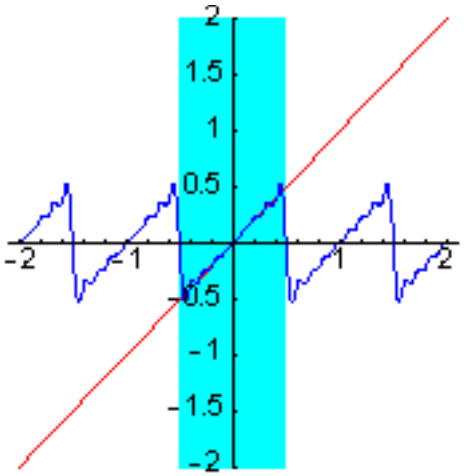
```
Simplify /@ FourierSeries[x + x^2, x, 8]
```

$$\frac{1}{12} + \frac{e^{-4 i \pi x}(1 - 2 i \pi)}{8 \pi^2} + \frac{e^{4 i \pi x}(1 + 2 i \pi)}{8 \pi^2} + \frac{e^{-6 i \pi x}(-1 + 3 i \pi)}{18 \pi^2} + \frac{e^{-8 i \pi x}(1 - 4 i \pi)}{32 \pi^2} + \frac{e^{8 i \pi x}(1 + 4 i \pi)}{32 \pi^2} + \frac{e^{-10 i \pi x}(-1 + 5 i \pi)}{50 \pi^2} + \frac{e^{-12 i \pi x}(1 - 6 i \pi)}{72 \pi^2} + \frac{e^{12 i \pi x}(1 + 6 i \pi)}{72 \pi^2} + \frac{e^{-14 i \pi x}(-1 + 7 i \pi)}{98 \pi^2} + \frac{e^{-16 i \pi x}(1 - 8 i \pi)}{128 \pi^2} + \frac{e^{16 i \pi x}(1 + 8 i \pi)}{128 \pi^2} - \frac{i e^{2 i \pi x}(-i + \pi)}{2 \pi^2} + \frac{i e^{-2 i \pi x}(i + \pi)}{2 \pi^2} - \frac{i e^{6 i \pi x}(-i + 3 \pi)}{18 \pi^2} - \frac{i e^{10 i \pi x}(-i + 5 \pi)}{50 \pi^2} - \frac{i e^{14 i \pi x}(-i + 7 \pi)}{98 \pi^2}$$

```
FourierTrigSeries[x + x^2, x, 8]
```

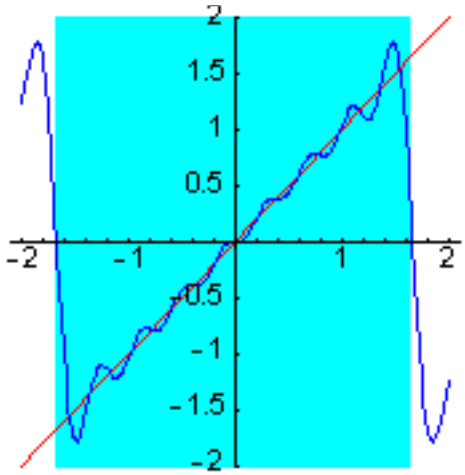
$$\frac{1}{12} - \frac{\text{Cos}[2 \pi x]}{\pi^2} + \frac{\text{Cos}[4 \pi x]}{4 \pi^2} - \frac{\text{Cos}[6 \pi x]}{9 \pi^2} + \frac{\text{Cos}[8 \pi x]}{16 \pi^2} - \frac{\text{Cos}[10 \pi x]}{25 \pi^2} + \frac{\text{Cos}[12 \pi x]}{36 \pi^2} - \frac{\text{Cos}[14 \pi x]}{49 \pi^2} + \frac{\text{Cos}[16 \pi x]}{64 \pi^2} + \frac{\text{Sin}[2 \pi x]}{\pi} - \frac{\text{Sin}[4 \pi x]}{2 \pi} + \frac{\text{Sin}[6 \pi x]}{3 \pi} - \frac{\text{Sin}[8 \pi x]}{4 \pi} + \frac{\text{Sin}[10 \pi x]}{5 \pi} - \frac{\text{Sin}[12 \pi x]}{6 \pi} + \frac{\text{Sin}[14 \pi x]}{7 \pi} - \frac{\text{Sin}[16 \pi x]}{8 \pi}$$

A representação da função antisimétrica f[x] = x no intervalo $[-1/2, +1/2]$ através duma série de Fourier é periódica com período 2 L = 1.



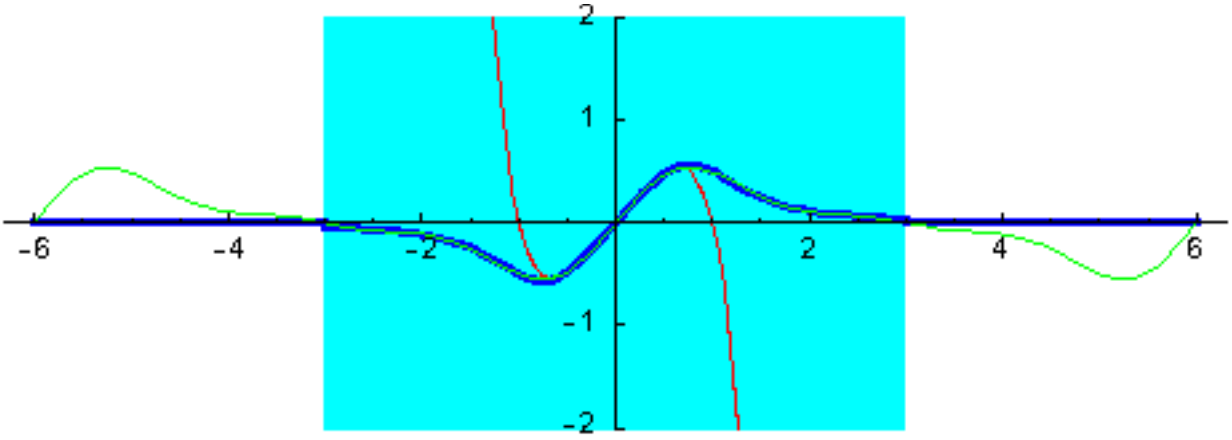
```
Show[{sw[1], Plot[{x, FourierTrigSeries[x, x, 8]} // Release, {x, −2, 2},
      PlotStyle → {RGBColor[1, 0, 0], RGBColor[0, 0, 1]}, DisplayFunction → Identity]},
  DisplayFunction → (Display[$Display, #1] &), ImageSize → 180,
  AspectRatio → 1, PlotRange → {−2, 2}];
```

Com a opção dada por FourierParameters → {0, b} o intervalo de amostragem de f[x] é $\left[-\frac{1}{2\,|b|}, +\frac{1}{2\,|b|}\right]$. Aqui, b = 1/6.




```
Show[{sw[.3], Plot[{x, FourierTrigSeries[x, x, 8, FourierParameters → {0, .3}]} // Release, {x, −2, 2},
      PlotStyle → {RGBColor[1, 0, 0], RGBColor[0, 0, 1]}, DisplayFunction → Identity]},
  DisplayFunction → (Display[$Display, #1] &), ImageSize → 180,
  AspectRatio → 1, PlotRange → {−2, 2}];
```

O gráfico seguinte mostra a qualidade da reconstrução do sinal original $f(x) = \frac{x}{1+x^4}$ no intervalo [−3, 3] (azul), através de uma série de potências de ordem 8 (vermelho) e uma série de Fourier com 4 frequências (verde).




```
Show[{sw[1/6.], Plot[{Series[x/(1 + x^4), {x, 0, 8}] // Normal, If[Abs[x] < 3, x/(1 + x^4), 0],
      FourierTrigSeries[x/(1. + x^4), x, 4, FourierParameters → {0, 1/6.}] // Chop // Re} //
    Release, {x, −6, 6}, PlotStyle → {{Thickness[0.003`], RGBColor[1, 0, 0]},
      {Thickness[0.008`], RGBColor[0, 0, 1]}, RGBColor[0, 1, 0]},
    PlotPoints → 60, DisplayFunction → Identity]},
  DisplayFunction → (Display[{"stdout"}, #1] &), PlotRange → 2 {−1, 1},
  ImageSize → 180, AspectRatio → 4/12,
  Ticks → {Automatic, Range[−2, 2]}];
```

□ ORTOGONALIDADE

As funções $s_n(x) = \text{Sin}\left[n \frac{\pi}{L} x\right]$ e $c_m = \text{Cos}\left[n \frac{\pi}{L} x\right]$ formam uma base ortonormada de $\mathcal{L}^2\left([-L, L]\right)$, i.e. $\langle s_n \mid s_m \rangle = \delta_{nm}$, $\langle c_n \mid s_m \rangle = 0$, $\langle c_n \mid c_m \rangle = \delta_{nm}$.

$$\langle s_n \mid s_m \rangle = \frac{1}{L} \int_c^{c+2L} \text{Sin}\left[n \frac{\pi}{L} x\right] \text{Sin}\left[m \frac{\pi}{L} x\right] dx \text{ // FullSimplify[\#, \{m, n\} } \in \text{ Integers] \&}$$

0

$$\langle s_n \mid s_n \rangle = \frac{1}{L} \int_c^{c+2L} \text{Sin}\left[n \frac{\pi}{L} x\right] \text{Sin}\left[n \frac{\pi}{L} x\right] dx \text{ // FullSimplify[\#, \{n\} } \in \text{ Integers] \&}$$

1

$$\langle c_n \mid s_m \rangle = \frac{1}{L} \int_c^{c+2L} \text{Cos}\left[n \frac{\pi}{L} x\right] \text{Sin}\left[m \frac{\pi}{L} x\right] dx \text{ // FullSimplify[\#, \{m, n\} } \in \text{ Integers] \&}$$

0

$$\langle c_n \mid s_n \rangle = \frac{1}{L} \int_c^{c+2L} \text{Cos}\left[n \frac{\pi}{L} x\right] \text{Sin}\left[n \frac{\pi}{L} x\right] dx \text{ // FullSimplify[\#, \{m, n\} } \in \text{ Integers] \&}$$

0

$$\langle c_n \mid c_n \rangle = \frac{1}{L} \int_c^{c+2L} \text{Cos}\left[n \frac{\pi}{L} x\right] \text{Cos}\left[n \frac{\pi}{L} x\right] dx \text{ // FullSimplify[\#, \{n\} } \in \text{ Integers] \&}$$

1

Transformada Rápida de Fourier

Os algoritmos que implementam a transformada rápida de Fourier assumem que a função a transformar $h(\zeta)$ está definida e amostrada num intervalo $0 \le \zeta \le (N-1)\Delta x$, com valores $q_r = h(\zeta_r)$ onde $\zeta_r = r\,\Delta x$ para $(r = 0, 1, \ldots, ns-1)$. Daí que, para amostrar e transformar uma função $f(x)$ numa janela $\mathcal{I} = \left[-\frac{\Lambda}{2}, \frac{\Lambda}{2}\right]$ de largura $\Lambda = N\,\Delta x$ seja necessário transladar toda a função por $\frac{\Lambda}{2}$, e efectivamente definir uma nova função $\mathbf{h(\zeta_r) = f\left(\zeta_r - \frac{\Lambda}{2}\right) = f\left(\left(r - \frac{N}{2}\right)\Delta x\right)}$

No *Mathematica*, a Transformada de Fourier discreta de uma lista $\{q_r\}$ de N pontos é a lista $\{\hat{q}_s\}$ onde

$$\hat{q}_{s'} = \frac{1}{\sqrt{N}} \sum_{r'=1}^{N} q_r\, e^{2\pi i (s'-1)\frac{(r'-1)}{N}}$$

Se pusemos $\Delta\kappa = \frac{2\pi}{\Lambda}$, $\mathbf{s = s' - 1}$, $\mathbf{r = r' - 1}$ então

$$\hat{q}_{s+1} = \frac{1}{\sqrt{N}} \sum_{r=0}^{N-1} q_{r+1}\, e^{i\left(\frac{2\pi}{\Lambda} s\right) r\, \Delta x} = \frac{1}{\sqrt{N}} \sum_{r=0}^{N-1} h(\zeta_r)\, e^{i\, \kappa_s\, \zeta_r}$$

$$\kappa_s = \begin{cases} 0 & \text{para } s = 0 \\ s\,\Delta\kappa & \text{para } s = 1,\ \ldots,\ \frac{N}{2} - 1 \\ \frac{N}{2}\,\Delta\kappa & \text{para } s = \frac{N}{2} \text{ (aliasing)} \\ (s - N)\,\Delta\kappa & \text{para } s = \frac{N}{2} + 1,\ \ldots,\ N - 1 \end{cases}$$

Designando por $\mathbf{x_r = \zeta_r - \frac{\Lambda}{2} \equiv \left(r - \frac{N}{2}\right) \Delta x}$ os pontos no intervalo $\mathcal{I}$, podemos reescrever

$$\hat{q}_{s+1} = \frac{1}{\sqrt{N}} \sum_{r=0}^{N-1} h(\zeta_r)\, e^{i\, \kappa_s\, \zeta_r} =$$

$$\frac{1}{\sqrt{N}} \sum_{r=0}^{N-1} f(x_r)\, e^{i\, \kappa_s \left(x_r + \frac{\Lambda}{2}\right)} =$$

$$\sqrt{\Delta x}\, e^{i\, \kappa_s\, \frac{\Lambda}{2}} \left(\frac{1}{\sqrt{\Lambda}} \sum_{r=0}^{N-1} f(x_r)\, e^{i\, \kappa_s\, x_r} \right) = \sqrt{\Delta x}\, e^{i\, \pi\, s}\, \hat{f}(\kappa_s)$$

Assim conclui–se que

$$\hat{f}(\kappa_s) = \frac{1}{\sqrt{\Delta x}}\, e^{-i\, \pi\, s}\, \hat{q}_{s+1}$$

Estes coeficientes $\hat{q}_{s'}$ diferem em sinal alterno dos provenientes de implementações C de FFT, onde o primeiro elemento duma lista tem índice 0.

Convém ainda indicar que, de acordo com as normas da FFT,

(a) – o primeiro coeficiente $\hat{q}_{s'=1}$ corresponde á frequência ('número de onda') $\kappa_{s'-1} = 0$.

(b) – os primeiros coeficientes $\hat{q}_{s'}$ $\left(s' = 2,\ \ldots,\ \frac{N}{2}\right)$ correspondem às frequências positivas $\kappa_{s'-1} = (s' - 1)\,\Delta\kappa$,

(c) – o coeficiente $\hat{q}_{\frac{N}{2}+1}$ corresponde ao ponto de 'aliasing' $\kappa_{s'-1} = \frac{N}{2}\,\Delta\kappa = (s' - 1)\,\Delta\kappa$

(d) – e os últimos $\hat{q}_{s'}$ $\left(s' = \frac{N}{2} + 2,\ \ldots,\ N\right)$ vão das frequências mais negativas para as menos negativas $\kappa_{s'-1} = (s' - 1 - N)\,\Delta\kappa$.

É assim desejável definir um operador **sortft[list]** que produza uma lista $\{p_k\}$ ordenada crescentemente com a frequência: o índice k aqui corresponde portanto a uma frequência $\mathbf{f = \frac{2\,ny}{ns}\left(k - \frac{ns}{2} - 1\right)}$. Note–se ainda que f é uma frequência, e não uma frequência angular $\omega = 2\pi f$.

Assim, o resultado de **Fourier[Cdata] → sftdata** não é directamente a transformada de Fourier da discretização, é necessário inverter o sinal dos coeficientes de dois em dois para obter **ftdata** e depois rearranjar com **sortft[ftdata]** para obter os verdadeiros coeficientes! Contudo, para inverter com **InverseFourier** deve–se usar o formato de **sftdata**.

■ Notas de Implementação

A discretização de uma função $f(x)$ num intervalo $\mathcal{I} = [x_{min}, x_{max}]$, utilizando $N = ns = 2^n$ pontos pode ser efectuada definindo um operador **Sample[*f*, n, $\mathcal{I}$]**, cujos produtos são:

Cdata	Lista de 2^n valores complexos $f[x_i]$
ny	Frequencia de Nyquist $= \frac{1}{2}$ sr
sr	Taxa de Amostragem $= \frac{N}{x_{max} - x_{min}}$
Δx	Resolução da Amostragem $= \frac{x_{max} - x_{min}}{N}$

Se em vez de uma janela de amostragem $\mathcal{I}$ se pretende fixar uma frequência de Nyquist $ny = \frac{1}{2}$ sr do espectro, é mais útil definir a função **SamplingWindow[ny, ns, x_0]** que produza uma janela adequada $\mathcal{I}_{ny}$ centrada no ponto x_0. Note–se que se $x_0 = 0$, a amostra r corresponde à coordenada $x_r = \left(r - \frac{ns}{2}\right)\Delta x$. Uma vez definida a janela de amostragem $\mathcal{I}$ bem como a resolução $n = \log_2(N)$ (ou equivalentemente a frequência de Nyquist *ny*) é possível efectuar uma transformada rápida de Fourier definindo **FFTAnalyse[*f*, n, ny]** cujo produto, entre outros, é uma lista sftdata de coeficientes de Fourier complexos.

Pode–se também definir operadores como **displayFT[freqdata, tks, opts]** e **displayIFT[spacedata, tks, opts]** que produzam os gráficos de **Fourier** e **InverseFourier** a partir de listas de coeficientes apropriados.

A transformada de Fourier em tempo da função de correlação $\mathcal{P}(t)$ num intervalo $\mathcal{T} = [0, T]$ feita com $\mathcal{N}$ incrementos $\Delta\tau = \frac{T}{\mathcal{N}}$ necessita da amostragem $\mathcal{P}(t_r)$ em instantes $0 \le t_r \le (\mathcal{N} - 1)\Delta\tau$ com $t_r = r\,\Delta\tau$, o que permite resoluções em frequência $\Delta\nu = \frac{2\pi}{T}$. A largura de banda é assim $|\nu| \le \frac{\mathcal{N}}{2}\Delta\nu \equiv \frac{\pi}{\Delta\tau}$, o que significa que $\Delta\tau$ deve ser escolhido de forma que as energias (de facto as frequências já que estamos a fazer $\hbar = 1$) dos estados ligados $\nu \le \Delta V_{max}$ estejam incluidos nesta banda, e por isso $\Delta\tau \le \frac{\pi}{\Delta V_{max}}$.