Métodos Numéricos

Jaime E. Villate
Faculdade de Engenharia
Universidade do Porto

http://def.fe.up.pt

Eletricidade, Magnetismo e Circuitos

E-mail: villate@fe.up.pt
WWW: http://villate.org

Versão: 17 de setembro de 2015

ISBN: 978-972-99396-3-1

Conteúdo

Prefácio

Este livro descreve alguns algoritmos de cálculo numérico usando o Sistema de Álgebra Computacional *Maxima* (http://maxima.sourceforge.net). Parte do material já foi usado em outros dois livros[6, 7] mas é apresentado aqui no contexto geral dos métodos numéricos.

Jaime E. Villate
E-mail: villate@fe.up.pt
WWW: http://villate.org
Porto, dezembro de 2013

1 Raízes de equações

1.1 Equações transcendentes

Algumas equações que dependem de uma variável x podem ser resolvidas para obter um ou mais valores de x que verificam a equação. Por exemplo, a equação $x^2+4x+3=0$ tem duas soluções, $x=1$ e $x=3$, e a equação $e^x=3$ tem uma única solução $x=\ln(3)=1.0986\ldots$ Existem equações, chamadas **transcendentes**, em que não é possível escrever uma expressão matemática para a solução x e até pode ser difícil determinar se existem soluções e quantas. Por exemplo, a equação $x+1=\tan(x)$ é uma equação transcendente; as suas soluções só podem ser calculadas de forma aproximada, usando métodos numéricos.

Para facilitar a procura das soluções de uma equação transcendente, é conveniente rescrevê-la na forma $f(x)=0$; por exemplo, a equação $x+1=\tan(x)$ pode ser escrita como $x+1-\tan(x)=0$. O problema de encontrar as soluções consiste então em encontrar as **raízes** da função $f(x)$, ou seja, os pontos onde o gráfico de $f(x)$ corta o eixo das abcissas. No exemplo anterior, $f(x)=x+1-\tan(x)$, o gráfico da função no intervalo $0\leq x\leq 2\pi$ obtém-se no Maxima com o seguinte comando:

```
(%i1) plot2d(x+1-tan(x),[x,0,2*%pi],[y,-5,5],[ylabel,"f(x)"]);
```

e o resultado (ver figura 1.1) mostra que nesse intervalo existem duas soluções, próximas dos valores $x=1.1$ e $x=4.5$.

No caso das funções contínuas, se em dois pontos diferentes a e b (admitindo $a<b$), os sinais de $f(a)$ e $f(b)$ são diferentes, deve existir pelo menos uma raiz de $f(x)$ no intervalo $a<x<b$. No caso da figura 1.1, vê-se que $f(0)$ é positiva mas $f(1.3)$ é negativa; assim sendo, existe pelo menos uma raiz entre 0 e 1.3. Em $x=3$, $f(3)$ é positiva, mas comparando com o valor negativo de $f(1.3)$ não se pode concluir que existam raízes no intervalo $1.3<x<3$, já que nesse intervalo a função não é contínua.

1.2 Método de bissecções

Dados dois valores diferentes x_1 e x_2, onde $x_1<x_2$ e $f(x_1)\,f(x_2)<0$ (ou seja, os sinais de $f(x_1)$ e $f(x_2)$ são diferentes), calculam-se o ponto meio $x_m=(x_1+x_2)/2$ e o valor da função nesse ponto, $f(x_m)$. Se $f(x_m)$ for nula, x_m é a raiz procurada; caso contrário, se o sinal de $f(x_m)$ for o mesmo de $f(x_1)$ substitui-se x_1 por x_m, se o sinal de $f(x_m)$ for o mesmo de $f(x_2)$ substitui-se x_2 por x_m e o processo repete-se indefinidamente até se obter

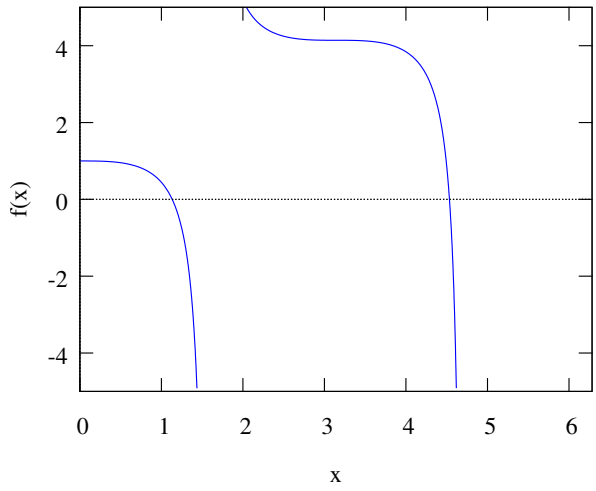

Figura 1.1: Gráfico da função $x+1-\tan(x)$.

um intervalo de comprimento muito reduzido:

$$|x_2 - x_1| < \varepsilon \tag{1.1}$$

onde $\varepsilon/2$ é a precisão com que se quer calcular a raiz. O valor final de x_m será o que melhor aproxima o valor da raiz, com a precisão desejada.

Exemplo 1.1
Calcule a raiz de $f(x) = x+1-\tan(x)$ no intervalo $0 < x < 1.3$ com 2 casas decimais, usando o método de bisseções.

Resolução. Definem-se primeiro a função e os pontos iniciais:

```
(%i2) f(x) := float (x+1-tan(x))$
(%i3) [x1, x2]: [1, 1.2]$
```

É importante usar `float`, para garantir que a função dê sempre um número e não uma expressão.

Os valores da função nos pontos iniciais são

```
(%i4) [f(x1), f(x2)];
(%o4)          [.4425922753450977, - .3721516221263186]
```

e como os sinais são diferentes, é possível usar o método das bissecções. O ponto meio e o valor da função nesse ponto são:

```
(%i5) [xm: (x1+x2)/2, f(xm)];
(%o5)                  [1.1, .1352403427513478]
```

Como o sinal da função nesse ponto é positivo, substitui-se x_1 pelo ponto meio e repete-se o passo anterior:

```
(%i6) x1: xm$
(%i7) [xm: (x1+x2)/2, f(xm)];
(%o7)                [1.15, - .08449694875532554]
```

Como agora a função é negativa, substitui-se x_2 por x_m e repete-se o mesmo procedimento até se observar que o ponto meio não mude, até à segunda casa decimal:

```
(%i8) x2: xm$
(%i9) [xm: (x1+x2)/2, f(xm)];
(%o9)                [1.125, .03242872362782112]
(%i10) x1: xm$
(%i11) [xm: (x1+x2)/2, f(xm)];
(%o11)             [1.1375, - .02411658214848611]
(%i12) x2: xm$
(%i13) [xm: (x1+x2)/2, f(xm)];
(%o13)             [1.13125, 0.00461493349083808]
```

O valor da raiz é $x = 1.13$. Há que ter muita atenção a qual dos valores, x_1 ou x_2, deve ser substituído por x_m, para garantir que a raiz esteja sempre dentro do intervalo atual.

As iterações também podiam ter sido feitas mais rapidamente usando um ciclo `while`. Por exemplo, para encontrar a raiz com 3 casas decimais (precisão igual a 0.0005), começando com os mesmos valores iniciais, pode escrever-se:

```
(%i14) [x1, x2]: [1.1, 1.2]$
(%i15) while abs(x2-x1) > 0.001 do
  (xm:(x1+x2)/2, if f(x1)*f(xm) > 0 then x1:xm else x2:xm, print(xm))$
1.15
1.125
1.1375
1.13125
1.134375
1.1328125
1.13203125
```

A raiz, com precisão de 3 casas decimais é 1.132. Observe-se que a função já tinha sido definida previamente e que já estava garantido que os sinais da função são diferentes nos dois pontos iniciais.

1.3 Método de falsa posição

No método das bissecções, se um dos valores $f(x_1)$ ou $f(x_2)$ estiver muito mais próximo de zero, espera-se que a raiz também esteja mas próxima do ponto x_1 ou x_2 em que a função está mais próxima de zero. Como tal, será mais eficiente usar o ponto x_r em que a reta que passa pelos pontos $(x_1, f(x_1))$ e $(x_2, f(x_2))$ intersecta o eixo dos x. Uma relação geométrica de semelhança entre triângulos permite demonstrar que x_r é dado pela expressão

$$x_r = x_2 - f(x_2)\frac{x_2 - x_1}{f(x_2) - f(x_1)} \tag{1.2}$$

O método de falsa posição é semelhante ao método de bissecções, excepto que em vez de se usar o ponto meio de cada intervalo usa-se o ponto x_r definido pela equação anterior.

Exemplo 1.2
Calcule a raiz de $f(x) = x + 1 - \tan(x)$ no intervalo $0 < x < 1.3$ com 3 casas decimais, usando o método de falsa posição.

Resolução. Aproveitando que a função é a mesma que já foi definida no exemplo da secção anterior, basta definir novamente os valores iniciais e modificar o ciclo `while` usado na secção anterior. É conveniente também guardar os valores da função já calculados, para evitar a redundância de serem calculados novamente a cada iteração:

```
(%i16) [x1, x2]: [1.1, 1.2]$
(%i17) [f1, f2]: [f(x1), f(x2)];
(%o17)       [.1352403427513478, - .3721516221263186]
```

Os valores da função nos dois extremos do intervalo são armazenados nas variáveis f_1 e f_2; o resultado (`%o17`) permite também conferir que o sinal da função muda entre x_1 e x_2. Como será claro nos resultados no fim deste exemplo, neste método a distância entre x_1 e x_2 não tem que ser pequena para que o valor x_r esteja muito próximo da raiz. Assim sendo, a condição de paragem $|x_2 - x_1| < \varepsilon$ já não serve e no seu lugar deve comparar-se cada valor x_r com o que tenha sido obtido na iteração anterior, que representaremos por x_0. Inicialmente pode admitir-se que x_0 e x_r são os próprios x_1 e x_2.

```
(%i18) [x0, xr]: [x1, x2]$
```

Para obter a precisão de 3 casas decimais, usa-se o comando:

```
(%i19) while abs(xr-x0) > 0.0005 do
  (x0:xr, xr:x2-f2*(x2-x1)/(f2-f1), fr:f(xr),
  print (x1, x2, xr),
  if f1*fr>0 then (x1:xr,f1:fr) else (x2:xr,f2:fr))$
1.1 1.2 1.126654017428903
1.126654017428903 1.2 1.131297851469829
1.131297851469829 1.2 1.132100363591035
1.132100363591035 1.2 1.132238851322909
```

Note-se que o valor da raiz com três casas decimais, 1.132, foi obtido na quarta iteração, enquanto que no método de bisseções foram precisas 7 iterações. Neste caso foram também apresentados os valores de x_1 e x_2 em cada iteração, para mostrar que o valor de x_2 permanece sempre em 1.2 e, como tal, $|x_2 - x_1|$ não diminui muito.

1.4 Método de aproximações sucessivas

Quando a equação $f(x) = 0$ pode ser escrita na forma,

$$x = g(x) \tag{1.3}$$

o método de aproximações sucessivas consiste em começar com um valor inicial x_0 e calcular a sequência $x_1 = g(x_0), x_2 = g(x_2), \ldots$ Dependendo da função g e do valor inicial,

em alguns casos a sequência aproxima-se de um limite, isto é, dentro de uma tolerância numérica dada, o valor de $g(x_n)$ será igual a x_n a partir de algum inteiro n e, nesse caso x_n será raiz de $f(x)$.

Exemplo 1.3
Encontre a raiz de $f(x) = x + 1 - \tan(x)$ mais próxima de $x = 1.1$, com 3 casas decimais, usando o método de aproximações sucessivas.

Resolução. A equação $x + 1 - \tan(x) = 0$ pode também ser escrita:

$$x = \tan(x) - 1$$

Definindo a função $g(x) = \tan(x) - 1$ e com valor inicial $x_0 = 1.1$, os seguintes 6 valores na sequência x_i são:

```
(%i20) g(x) := float (tan(x)-1)$
(%i21) xi: 1.1$
(%i22) for i:1 thru 6 do (xi: g(xi), print (xi))$
.9647596572486523
.4429269858864537
- .5256388221063686
- 1.580073542507877
106.7878688889125
- 1.026287385725812
```

Esta sequência não parece ser convergente. O problema neste caso é que uma das condições para que a sequência seja convergente é que, na vizinhança da raiz que se pretende encontrar, o declive de $g(x)$ seja menor que 1, que não é o que acontece neste caso:

```
(%i23) subst(x=1.1, diff(tan(x)-1, x));
(%o23)                    4.860280510751842
```

O problema pode ser resolvido quando se consegue inverter a função usada. Ou seja, em vez de se usar $x = \tan(x) - 1$ pode usar-se a equação inversa, $\arctan(x+1) = x$, que implica usar $g(x) = \arctan(x+1)$. No Maxima, a função inversa da tangente é a função `atan`. Com valor inicial $x_0 = 1.1$, os primeiros termos da sequência x_i são:

```
(%i24) g(x) := float (atan(x+1))$
(%i25) xi: 1.1$
(%i26) for i:1 thru 6 do (xi: g(xi), print (xi))$
1.126377116893798
1.131203287160338
1.132075737308623
1.132233108872913
1.132261484138675
1.132266600045208
```

que converge rapidamente para o valor da raiz.

1.5 Método de Newton

Tal como no método de aproximações sucessivas, basta ter um valor inicial para criar uma sequência que se aproxima da raiz. As vantagens em relação ao método de aproximações sucessivas é que não é preciso escrever a equação $f(x) = 0$ na forma $x = g(x)$ e a sequência converge em muitos mais casos. A desvantagem é que é preciso conhecer a derivada, $f'(x)$, da função $f(x)$.

Se $f(x)$ é uma função contínua e derivável, o valor de $f(x_{i+1})$ pode ser calculado, a partir do valor de $f(x_i)$, usando a série de Taylor

$$f(x_{i+1}) = f(x_i) + (x_{i+1} - x_i)\, f'(x_i) + \cdots \tag{1.4}$$

Se $|x_{i+1} - x_i|$ for suficientemente pequeno, os dois primeiros termos da série, apresentados na equação acima, são uma boa aproximação para o valor da série completa. Se o ponto x_{i+1} for uma raiz, então $f(x_{i+1}) = 0$ e a equação anterior conduz à relação de recorrência,

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \tag{1.5}$$

que permite definir uma sequência de valores x_i que deverá aproximar-se do valor da raiz.

Exemplo 1.4
Encontre a raiz de $f(x) = x + 1 - \tan(x)$ mais próxima de $x = 1.1$, com 3 casas decimais, usando o método de Newton.

Resolução. A função $f(x)$ já foi definida no comando (`%i2`) mas como é necessário também definir a sua derivada, vai repetir-se a definição de $f(x)$ para comparar com a forma como $f'(x)$ deve ser definida:

```
(%i27) f(x) := float (x+1-tan(x))$
(%i28) df(x) := float (''(diff (x+1-tan(x), x)));
                                                  2
(%o28)                df(x) := float(1 - sec (x))
```

Nas definições de $f(x)$ e $df(x)$ (derivada de $f(x)$), a função `float` não é aplicada imediatamente, mas fica indicada na definição das funções e só será aplicada quando seja dado um valor a x para calcular $f(x)$ ou $df(x)$. Se a função `float` fosse aplicada imediatamente quando $f(x)$ é definida, o resultado seria $x + 1.0 + \tan(x)$ e se mais tarde fosse calculado $f(1)$ o resultado seria $2.0 + \tan(1)$, sem que $\tan(1)$ fosse aproximada por um número de vírgula flutuante.

No entanto, no caso de `diff(x+1+tan(x),x)`, o que se pretende é que a derivada seja calculada imediatamente e o resultado usado para definir a função `df(x)`, em vez de que `diff(x+1+tan(x),x)` seja calculado após ter sido dado um valor para x (uma expressão como `diff(1+1+tan(1),1)` produz um erro). A sintaxe `''`(...) foi

usada para forçar a que a expressão dentro dos parêntesis seja calculada e simplificada imediatamente.

Usando o valor inicial $x_0 = 1.1$, os seguintes 6 valores na sequência x_i são:

```
(%i29) xi: 1.1$
(%i30) for i:1 thru 6 do (xi: xi-f(xi)/df(xi), print (xi))$
1.135033812277287
1.13228759380087
1.132267726299738
1.132267725272885
1.132267725272885
1.132267725272885
```

Note-se a rapidez com que o método converge; após apenas 3 interações já se obtêm 4 casas decimais para a raiz e após a quinta iteração já é obtida a solução com o número máximo de casas decimais (15) que é possível obter com o formato de dupla precisão numérica usado pela função `float`.

1.6 Sistemas de equações transcendentes

O método de Newton pode ser generalizado facilmente para resolver um sistema de n variáveis com n equações contínuas e deriváveis. Por exemplo, no caso de duas equações com duas variáveis, $f(x,y) = 0$ e $g(x,y) = 0$, os primeiros termos nas séries de Taylor para as duas funções são:

$$f(x_{i+1},y_{i+1}) = f(x_i,y_i) + (x_{i+1}-x_i)\left.\frac{\partial f}{\partial x}\right|_{(x_i,y_i)} + (y_{i+1}-y_i)\left.\frac{\partial f}{\partial y}\right|_{(x_i,y_i)} \tag{1.6}$$

$$g(x_{i+1},y_{i+1}) = g(x_i,y_i) + (x_{i+1}-x_i)\left.\frac{\partial g}{\partial x}\right|_{(x_i,y_i)} + (y_{i+1}-y_i)\left.\frac{\partial g}{\partial y}\right|_{(x_i,y_i)} \tag{1.7}$$

Os índices (x_i, y_i) indicam que x e y devem ser substituídas por (x_i, y_i). Substituindo $f(x_{i+1},y_{i+1}) = 0$ e $g(x_{i+1},y_{i+1}) = 0$ e escrevendo o sistema em forma matricial, obtém-se,

$$\mathbf{J}(x_i,y_i)\,(\mathbf{r}_{i+1}-\mathbf{r}_i) = -\mathbf{F}(x_i,y_i) \tag{1.8}$$

onde $\mathbf{J}(x_i,y_i)$ é a **matriz jacobiana**:

$$\mathbf{J}(x,y) = \begin{bmatrix} \dfrac{\partial f}{\partial x} & \dfrac{\partial f}{\partial y} \\ \dfrac{\partial g}{\partial x} & \dfrac{\partial g}{\partial y} \end{bmatrix} \tag{1.9}$$

calculada em $x = x_i$ e $y = y_i$. As matrizes $\mathbf{r}_{i+1}$, $\mathbf{r}_i$ e $\mathbf{F}(x_i,y_i)$ são:

$$\mathbf{r}_{i+1} = \begin{bmatrix} x_{i+1} \\ y_{i+1} \end{bmatrix} \qquad \mathbf{r}_i = \begin{bmatrix} x_i \\ y_i \end{bmatrix} \qquad \mathbf{F}(x_i,y_i) = \begin{bmatrix} f(x_i,y_i) \\ g(x_i,y_i) \end{bmatrix} \tag{1.10}$$

A equação (1.8) permite definir a relação de recorrência para x_{i+1} e y_{i+1},

$$\mathbf{r}_{i+1} = \mathbf{r}_i - \mathbf{J}^{-1}(x_i, y_i)\,\mathbf{F}(x_i, y_i) \tag{1.11}$$

onde $\mathbf{J}^{-1}(x,y)$ é a matriz inversa da matriz jacobiana.

Exemplo 1.5
Encontre os pontos de intersecção da circunferência $x^2 + y^2 = 3$ com a hipérbole $y = 1/x$.

Resolução. Os gráficos da circunferência e da hipérbole podem ser traçados com o comando:

```
(%i31) plot2d ([sqrt(3-x^2), -sqrt(3-x^2), 1/x], [x,-3,3],
        [y,-2.25,2.25], [legend,false], [color,blue,blue,red]);
```

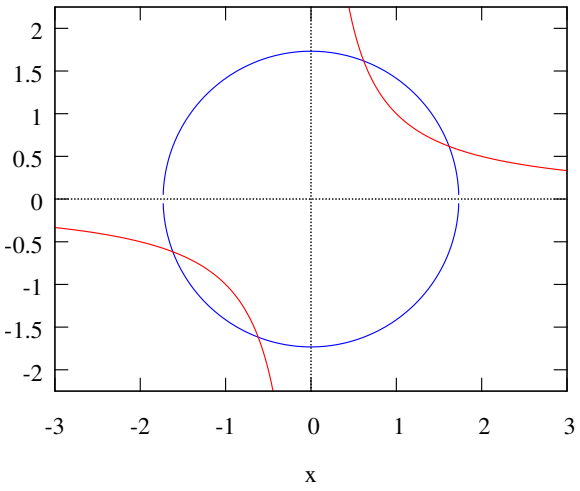

Figura 1.2: Gráficos da circunferência $x^2 + y^2 = 3$ e a hipérbole $y = 1/x$.

O gráfico (figura 1.2) mostra que existem quatro pontos de intersecção, simétricos em relação às retas $y = \pm x$. Basta encontrar um desses quatro pontos, por exemplo, o que está mais próximo do ponto inicial $x_0 = 0.5$, $y_0 = 1$. Escrevendo as duas equações na forma $f(x,y) = 0$ e $g(x,y) = 0$, as duas funções são definidas pelas expressões,

```
(%i32) f: x^2+y^2-3$
(%i33) g: 1-x*y$
```

e a inversa da matriz jacobiana é:

```
(%i34) Jinv: invert (jacobian ([f,g], [x,y]))$
```

Define-se a função vectorial $\mathbf{F}(\mathbf{r}_i)$,

```
(%i35) F(ri) := matrix ([subst ([x=ri[1][1], y=ri[2][1]], f)],
        [subst ([x=ri[1][1], y=ri[2][1]], g)])$
```

A seguir, dá-se o valor inicial para a matriz $\mathbf{r}_i$ e realizam-se algumas iterações:

```
(%i36) ri: matrix ([0.5], [1])$
(%i37) for i:1 thru 6 do
         (ri: ri - subst ([x=ri[1][1], y=ri[2][1]], Jinv).F(ri),
        print (transpose (ri)))$
[ .5833333333333334  1.833333333333333 ]
[ .6089080459770114  1.633908045977012 ]
[ .6178866256040899  1.61819150365287 ]
[ .6180339655308433  1.618034011991991 ]
[ .6180339887498942  1.618033988749896 ]
[ .6180339887498948  1.618033988749895 ]
```

onde o ponto entre `Jinv` e `F(ri)` indica produto matricial.

Tendo em contra a simetria das soluções, as quatro soluções (x, y) são então: (0.6180, 1.6180), (1.6180, 0.6180), (-0.6180, -1.6180) e (-1.6180, -0.6180).

2 Sistemas de equações lineares

2.1 Representação matricial

Um sistema linear de ordem n é um sistema com n variáveis $x_1, x_2, \ldots, x_n$, que satisfazem n equações lineares:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{21}x_2 + \cdots + a_{2n}x_n &= b_2 \\ &\vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n \end{aligned} \tag{2.1}$$

Se as equações são linearmente independentes, existe sempre uma única solução. A **matriz aumentada** do sistema é:

$$\left[\begin{array}{cccc|c} a_{11} & a_{12} & \cdots & a_{1n} & b_1 \\ a_{21} & a_{22} & \cdots & a_{2n} & b_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} & b_n \end{array}\right] \tag{2.2}$$

2.2 Método de eliminação de Gauss

A qualquer equação no sistema pode somar-se outra das equações, multiplicada por uma constante k, e a solução do sistema não se altera. Na matriz aumentada, essa operação corresponde a substituir uma linha L_i por $L_i + kL_j$. Essa propriedade pode ser usada para obter uma matriz **triangular superior**, em que a_{ij} é zero, se $i < j$:

$$\left[\begin{array}{cccc|c} a_{11} & a_{12} & \cdots & a_{1n} & b_1 \\ 0 & a'_{22} & \cdots & a'_{2n} & b'_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & a'_{nn} & b'_n \end{array}\right] \tag{2.3}$$

que corresponde a outro sistema de equações com a mesma solução do sistema original (as constantes a'_{ij} e b'_j não são as mesmas a_{ij} e b_j da matriz inicial).

A última equação nesse sistema permite obter facilmente o valor da variável x_n

$$a'_{nn}x_n = b'_n \quad \Rightarrow \quad x_n = \frac{b'_n}{a'_{nn}} \tag{2.4}$$

Esse resultado pode ser substituído na penúltima equação para obter o valor de x_{n-1},

$$a'_{n-1,n-1}x_{n-1} + a'_{n-1,n}x_n = b'_{n-1} \quad \Rightarrow \quad x_n = \frac{b'_n - 1}{a'_{n-1,n-1}} - \frac{a'_{n-1,n}b'_n}{a'_{n-1,n-1}a'_{nn}} \tag{2.5}$$

e assim sucessivamente, até se obter o valor de todas as variáveis.

Exemplo 2.1
Resolva o sistema de equações:

$$\begin{aligned} 8x + 4.5z - 12.4 &= 3.2y \\ 2x - 0.8y &= 7.6 + 6.1z \\ x + 3y + 10.2z &= 8.4 \end{aligned}$$

usando o método de eliminação de Gauss.

Resolução. Usando o Maxima, convém guardar-se as 3 equações em 3 variáveis

```
(%i1) eq1: 8*x + 4.5*z - 12.4 = 3.2*y$
(%i2) eq2: 2*x - 0.8*y = 7.6 + 6.1*z$
(%i3) eq3: x + 3*y + 10.2*z = 8.4$
```

Este sistema pode ser escrito na mesma forma do sistema geral (2.1) para obter a matriz aumentada. A função `augcoefmatrix` realiza esse procedimento dando como resultado a matriz aumentada; é necessário dar dois argumentos a `augcoefmatrix`, a lista das equações e a lista das variáveis, na ordem que se quiser. Neste caso, a matriz aumentada pode ser guardada na variável *m* com o seguinte comando:

```
(%i4) m: float (augcoefmatrix ([eq1, eq2, eq3], [x, y, z]));
                    [ 8.0   - 3.2    4.5    - 12.4 ]
                    [                              ]
(%o4)               [ 2.0   - 0.8  - 6.1    - 7.6  ]
                    [                              ]
                    [ 1.0    3.0    10.2    - 8.4  ]
```

optou-se por usar a função `float` para evitar que a matriz seja escrita com números racionais.

Há que ter em conta que os números na última coluna não são b_i mas sim $-b_i$ (ou seja, no sistema (2.1) todas as equações foram igualadas a zero).

Antes de começar com o processo de eliminação podem fixar-se um número reduzido de algarismos significativos, por exemplo 4, para os resultados que serão apresentados a seguir, de modo a evitar que as matrizes tenham muitos algarismos:

```
(%i5) fpprintprec: 4$
```

Como está a ser usado o formato de vírgula flutuante de dupla precisão (8 bytes), internamente os cálculos continuarão a ser feitos com 16 algarismos significativos, mas cada vez que se apresentem os resultados, serão arredondados para 4 algarismos significativos.

Para reduzir a matriz à forma triangular, começa-se por substituir a segunda linha, L_2, por $L_2 - (1/4)L_1$, onde L_1 representa toda a primeira linha da matriz. O objetivo é fazer com que a_{21} fique igual a zero (note-se que 1/4 foi calculado dividindo a_{21} por a_{11}).

No Maxima, a função `rowop`(m, i, j, k) produz uma nova matriz, em que a linha L_i da matriz m é substituída por $L_i - kL_j$ e as restantes linhas ficam iguais. Assim sendo, a substituição $L_2 - (1/4)L_1 \longrightarrow L_2$ pode ser feita com o seguinte comando:

```
(%i6) m: rowop (m, 2, 1, 1/4);
                    [ 8.0  - 3.2    4.5    - 12.4 ]
                    [                             ]
(%o6)               [ 0.0   0.0   - 7.225  - 4.5  ]
                    [                             ]
                    [ 1.0   3.0    10.2    - 8.4  ]
```

A seguir, para eliminar $a_{31} = 1.0$ usando novamente $a_{11} = 8.0$, deverá ser feita a substituição $L_3 - kL_1 \longrightarrow L_3$, onde $k = a_{11}/a_{31} = 1/8$. Isto é,

```
(%i7) m: rowop (m, 3, 1, 1/8);
                    [ 8.0  - 3.2    4.5    - 12.4 ]
                    [                             ]
(%o7)               [ 0.0   0.0   - 7.225  - 4.5  ]
                    [                             ]
                    [ 0.0   3.4    9.637   - 6.85 ]
```

Neste momento deveria usar-se a'_{22} para eliminar a'_{23}. No entanto, como $a'_{22} = 0.0$, isso não é possível ($k = a'_{23}/a'_{22}$ não existe). Em casos como este pode-se trocar a ordem das segunda e terceira linhas, que equivale a alterar a ordem das equações, sem que por isso se altere a solução. A função `rowswap` do Maxima permite trocar a ordem de duas linhas:

```
(%i8) m: rowswap (m, 2, 3);
                    [ 8.0  - 3.2    4.5    - 12.4 ]
                    [                             ]
(%o8)               [ 0.0   3.4    9.637   - 6.85 ]
                    [                             ]
                    [ 0.0   0.0   - 7.225  - 4.5  ]
```

ficando assim uma matriz diagonal superior, que permite passar à segunda fase do método.

Para resolver as equações associadas com cada linha da matriz é conveniente definir uma lista com as variáveis, na mesma ordem em que foram usadas para encontrar a matriz aumentada, seguidas por 1:

```
(%i9) vars: [x, y, z, 1]$
```

Multiplicando a terceira linha da matriz m pela lista de variáveis, obtém-se a última equação, na qual só aparece a variável z; resolve-se essa equação para encontrar o valor de z, que será guardado para ser usado nos passos seguintes

```
(%i10) sol: float (solve (m[3].vars));
(%o10)                        [z = - .6228]
```

Multiplicando a segunda linha da matriz pela lista de variáveis e substituindo o valor que já foi encontrado para z, obtém-se uma equação que depende unicamente de y. A resolução dessa equação conduz ao valor de y e é conveniente acrescentar esse resultado na mesma lista em que foi armazenado o valor de z

```
(%i11) sol: append (float (solve ( subst (sol, m[2].vars))), sol);
(%o11)                  [y = 3.78, z = - .6228]
```

Finalmente, basta repetir o passo anterior, mas agora multiplicando pela primeira linha da matriz, para determinar o valor de x

```
(%i12) sol: append (float (solve ( subst (sol, m[1].vars))), sol);
(%o12)           [x = 3.412, y = 3.78, z = - .6228]
```

2.3 Método de Gauss-Jordan

O método de eliminação de Gauss tem a vantagem de não alterar o valor do determinante da matriz com coeficientes a_{ij} e, como tal, o determinante pode ser calculado facilmente multiplicando os valores a'_{ii} na diagonal da matriz triangular obtida.

O método de Gauss-Jordan consiste em transformar a matriz aumentada até os coeficientes a'_{ij} corresponder à matriz identidade, ou seja, iguais a 1 se $i = j$ ou zero caso contrário. A grande vantagem é que os valores finais de b'_i são os valores das variáveis, sem ser preciso realizar mais contas. Em contrapartida, a matriz final não permite calcular o determinante de a_{ij}.

Para conseguir transformar a matriz na matriz identidade, começa-se por dividir a primeira equação por a_{11}, para que a'_{11} fique igual a 1, e a seguir usa-se essa primeira linha para eliminar o primeiro elemento em todas as outras linhas. A seguir divide-se a segunda linha por a'_{22}, para que este fique igual a 1 e usa-se essa segunda linha para eliminar o segundo elemento em todas as outras linhas, incluindo a primeira.

Não existe um comando específico do Maxima para dividir uma linha de uma matriz por uma constante; no entanto, a divisão por k pode ser feita com a função `rowop`(m, i, i, $1-1/k$), que substitui a linha L_i da matriz m, por $L_i - (1-1/k)\,L_i = L_i/k$.

Exemplo 2.2
Determine a solução do seguinte sistema, pelo método de Gauss-Jordan

$$\begin{aligned} 2x_1 - 1.2x_2 - 4x_3 &= 5 \\ -3x_1 + 3x_2 + 5x_3 &= 12 \\ 2.5x_1 + 4x_2 - 3x_3 &= -6 \end{aligned}$$

Resolução. Para mostrar como é possível obter soluções exatas, na resolução deste exemplo só será usada a função `float` no fim e os dois números de vírgula flutuante 1.2 e 2.5 na primeira e terceira equações serão escritos como as fracções 12/10 e 25/10.

Em vez de se usar a função `augcoefmatrix`, a matriz aumentada pode também ser inserida diretamente usando a função `matrix`

```
(%i13) m: matrix([2,-12/10,-4,5],[-3,3,5,12],[25/10,4,-3,-6]);
                        [         6          ]
                        [  2    - -  - 4   5  ]
                        [         5          ]
                        [                    ]
(%o13)                  [ - 3   3     5   12 ]
                        [                    ]
                        [  5                 ]
                        [  -    4    - 3  - 6 ]
                        [  2                 ]
```

Começa-se por dividir a primeira linha pelo seu primeiro elemento (ou seja 2):

```
(%i14) m: rowop (m, 1, 1, 1 - 1/m[1][1]);
                        [         3          5  ]
                        [  1    - -  - 2    -  ]
                        [         5          2  ]
                        [                    ]
(%o14)                  [ - 3   3     5   12 ]
                        [                    ]
                        [  5                 ]
                        [  -    4    - 3  - 6 ]
                        [  2                 ]
```

A seguir, subtrai-se a primeira linha, multiplicada pelo primeiro elemento da segunda linha, para eliminar o primeiro elemento de segunda linha:

```
(%i15) m: rowop (m, 2, 1, m[2][1]);
                        [       3          5  ]
                        [ 1   - -  - 2     -  ]
                        [       5          2  ]
                        [                     ]
                        [      6          39  ]
(%o15)                  [ 0    -    - 1   --  ]
                        [      5          2   ]
                        [                     ]
                        [ 5                   ]
                        [ -    4    - 3  - 6  ]
                        [ 2                   ]
```

e faz-se o mesmo com a primeira e a terceira linhas, para eliminar o primeiro elemento da terceira linha:

```
(%i16) m: rowop (m, 3, 1, m[3][1]);
                        [       3          5   ]
                        [ 1   - -  - 2     -   ]
                        [       5          2   ]
```

```
                          [                    ]
                          [     6          39  ]
(%o16)                    [ 0   -    - 1   --  ]
                          [     5          2   ]
                          [                    ]
                          [    11           49 ]
                          [ 0  --    2    - -- ]
                          [    2            4  ]
```

O mesmo procedimento repete-se para a segunda coluna; isto é, divide-se a segunda linha pelo seu segundo elemento,

```
(%i17) m: rowop (m, 2, 2, 1 - 1/m[2][2]);
                          [      3         5   ]
                          [ 1  - -  - 2    -   ]
                          [      5         2   ]
                          [                    ]
                          [           5    65  ]
(%o17)                    [ 0   1   - -    --  ]
                          [           6    4   ]
                          [                    ]
                          [    11           49 ]
                          [ 0  --    2    - -- ]
                          [    2            4  ]
```

e usa-se essa nova linha para eliminar o segundo elemento na primeira e na terceira linhas:

```
(%i18) m: rowop (rowop (m, 1, 2, m[1][2]), 3, 2, m[3][2]);
                          [          5    49   ]
                          [ 1  0   - -    --   ]
                          [          2    4    ]
                          [                    ]
                          [          5    65   ]
(%o18)                    [ 0  1   - -    --   ]
                          [          6    4    ]
                          [                    ]
                          [       79        813 ]
                          [ 0  0  --     - --- ]
                          [       12        8  ]
```

Finalmente, repete-se o mesmo procedimento para a terceira coluna:

```
(%i19) m: rowop (m, 3, 3, 1 - 1/m[3][3]);
                          [          5     49   ]
                          [ 1  0   - -     --   ]
                          [          2     4    ]
                          [                    ]
                          [          5     65   ]
(%o19)                    [ 0  1   - -     --   ]
                          [          6     4    ]
```

```
                          [                   ]
                          [              2439 ]
                          [ 0  0   1   - ---- ]
                          [              158  ]
(%i20) m: rowop (rowop (m, 1, 3, m[1][3]), 2, 3, m[2][3]);
                           [             2081 ]
                           [ 1  0   0  - ---- ]
                           [              79  ]
                           [                  ]
                           [            535   ]
(%o20)                     [ 0  1   0   ---   ]
                           [            158   ]
                           [                  ]
                           [             2439 ]
                           [ 0  0   1  - ---- ]
                           [             158  ]
```

Os três números na última coluna são os valores exatos das 3 variáveis. Para aproximar esses números por números de vírgula flutuante e colocá-los numa lista, extrai-se a quarta linha da matriz transposta e aplica-se a função `float`

```
(%i21) float (transpose(m)[4]);
(%o21)                [- 26.34, 3.386, - 15.44]
```

Note-se que se tivesse sido usada a função `augcoefmatrix` para construir a matriz, seria necessário mudar os sinais desses 3 valores obtidos.

Este algoritmo pode ser automatizado facilmente; o mesmo resultado obtido com os comandos (`%i13`) até (`%i20`) podia ter-se obtido com os seguintes comandos:

```
(%i22) m: matrix([2,-12/10,-4,5],[-3,3,5,12],[25/10,4,-3,-6])$
(%i23) for i:1 thru 3 do (
         m: rowop (m, i, i, 1 - 1/m[i][i]),
         for j:1 thru 3 do (
           if (i # j) then m: rowop (m, j, i, m[j][i])));
(%o23)                          done
```

onde # é o operador "diferente de".

Se em alguma das iterações `m[i][i]` for nulo, `1/m[i][i]` produz um erro. Para evitar esse tipo de erro seria necessário acrescentar instruções para conferir que `m[i][i]` não seja nulo e, caso contrário, trocar a linha L_i com alguma das linhas L_j com $j > i$.

3 Equações diferenciais

3.1 Forma geral das equações diferenciais

Uma **equação diferencial ordinária** —ou de forma abreviada, EDO— de ordem n é uma relação entre uma função $y(x)$ e as suas derivadas y', y'', ..., $y^{(n)}$. Por exemplo, uma possível equação diferencial ordinária de terceira ordem é:

$$3x^2 y''' + \frac{y}{2y'} = xy'' \tag{3.1}$$

Neste caso, é possível também escrever a equação mostrando explicitamente a expressão que define a derivada de terceira ordem:

$$y''' = \frac{y''}{3x} - \frac{y}{6x^2 y'} \tag{3.2}$$

É conveniente usar uma notação que permite ver a equação duma forma mais geral: a função y será denotada por y_1, y_2 será a primeira derivada y' e y_3 a segunda derivada y''. A equação diferencial é então:

$$y_3' = \frac{y_3}{3x} - \frac{y_1}{6x^2 y_2} \tag{3.3}$$

que define a derivada de y_2 em relação a (x, y_1, y_2, y_3). Para resolver este problema é necessário resolver simultaneamente as equações que definem as derivadas de y_1 e y_2. Ou seja, a equação original (3.1), de terceira ordem, foi transformada no sistema de 3 equações:

$$y_1' = y_2 \tag{3.4}$$
$$y_2' = y_3 \tag{3.5}$$
$$y_3' = \frac{y_3}{3x} - \frac{y_1}{6x^2 y_2} \tag{3.6}$$

Esta forma de escrever a equação diferencial tem também a vantagem de poder ser escrita de forma mais compacta; define-se o vetor,

$$\vec{y} = (y_1, y_2, y_3) \tag{3.7}$$

e o sistema de 3 equações diferenciais é equivalente a uma única equação diferencial vetorial:

$$\frac{\mathrm{d}\vec{y}}{\mathrm{d}x} = \vec{f}(x, \vec{y}) \tag{3.8}$$

onde a derivada do vetor $\vec{y}$ é outro vetor obtido derivando cada uma das suas componentes,

$$\frac{\mathrm{d}\vec{y}}{\mathrm{d}x} = (y_1', y_2', y_3') \tag{3.9}$$

e as três componentes da função vetorial $\vec{f}(x,\vec{y})$ são, neste caso,

$$\vec{f}(x,\vec{y}) = \left(y_2, y_3, \frac{y_3}{3x} - \frac{y_1}{6x^2 y_2}\right) \tag{3.10}$$

Assim sendo, para resolver equações diferencias ordinárias como, por exemplo, a equação (3.1), basta saber resolver equações de primeira ordem com a forma geral

$$\frac{\mathrm{d}y}{\mathrm{d}x} = f(x,y) \tag{3.11}$$

e generalizar o método ao caso em que y e f são vetores com várias componentes.

3.2 Equações diferenciais de primeira ordem

A forma geral das EDO de primeira ordem é

$$y' = f(x,y) \tag{3.12}$$

A função $y(x)$ é chamada **variável dependente** e x é a **variável independente**. Uma solução da EDO, num intervalo $[x_0, x_n]$, é qualquer função y de x que verifica a equação.

Existem em geral muitas soluções, por exemplo, a figura 3.1 mostra 7 possíveis soluções da equação $y' = (x-1)(x-3) - y$. Em cada ponto de cada uma das curvas, o declive tem o mesmo valor obtido substituindo as coordenadas do ponto na expressão $(x-1)(x-3) - y$. As diferentes soluções não se cruzam nunca, porque em cada ponto existe apenas um valor $(x-1)(x-3) - y$ e cada ponto (x,y) pertence unicamente a uma das soluções da equação diferencial.

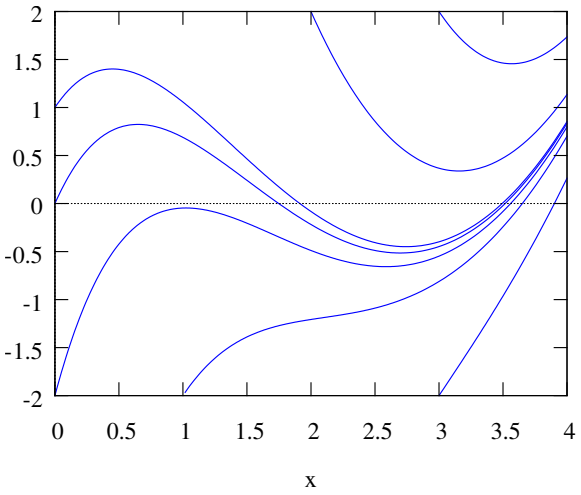

Figura 3.1: Soluções da equação diferencial $y' = (x-1)(x-3) - y$.

Se for dado um valor inicial y_0 para a função y no ponto inicial x_0, existe uma única solução $y(x)$, que é a curva tal que $y(x_0) = y_0$ e com declive igual a $(x-1)(x-3) - y(x)$ em qualquer valor de x.

Os métodos de resolução numérica consistem em calcular o valor da variável dependente y numa sequência discreta de pontos $\{x_0, x_1, x_2, \ldots, x_n\}$, usando alguma aproximação. O resultado obtido aplicando um determinado método será o conjunto de pontos $\{(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)\}$, que aproximam o gráfico da função $y(x)$. Nos métodos que usam incrementos constantes, o intervalo $[x_0, x_n]$ divide-se em n subintervalos de comprimento $h = (x_n - x_0)/n$, de forma que cada valor na sequência $\{x_i\}$ é igual ao valor anterior mais h:

$$\{x_0, x_1 = x_0 + h, x_2 = x_0 + 2h, \ldots, x_n = x_0 + nh\} \tag{3.13}$$

Por exemplo, a figura 3.2 mostra a solução da equação diferencial $y' = f(x,y)$ com $f = (x-1)(x-3) - y$ e condição inicial $y(0) = 1$, no intervalo $0 \leq x \leq 4$. Os cinco pontos sobre a curva são a aproximação da função usando apenas 5 pontos com abcissas 0, 1, 2, 3 e 4.

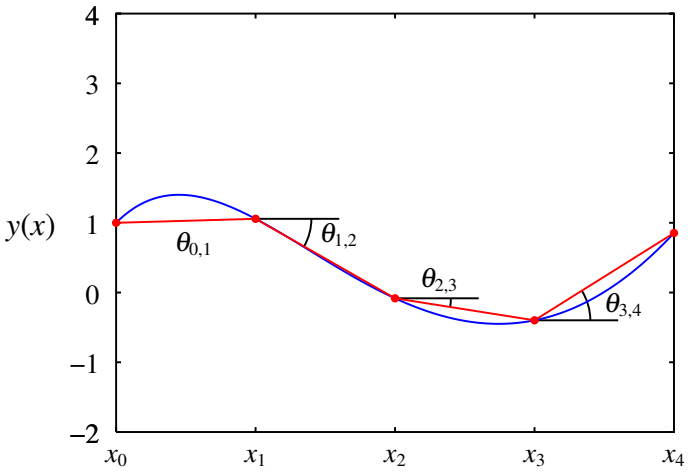

Figura 3.2: Uma solução da equação $y' = (x-1)(x-3) - y$, e aproximação com 5 pontos.

Cada ângulo $\theta_{i,j}$ na figura é a inclinação segmento entre os pontos (x_i, y_i) e $(x_{i,j}, y_{i,j})$. A tangente do ângulo $\theta_{i,j}$ (declive médio) é igual ao valor médio de $f(x,y)$ no intervalo $[x_i, x_j]$. A partir das coordenadas (x_i, y_i) de um dos pontos, o declive médio $\bar{f}_{i,i+1}$ permite calcular as coordenadas do ponto seguinte:

$$\boxed{x_{i+1} = x_i + h \qquad y_{i+1} = y_i + h\,\bar{f}_{i,i+1}} \tag{3.14}$$

Usando estas **relações de recorrência** de forma iterativa, consegue-se obter as coordenadas de todos os pontos (x_i, y_i), começando com os valores iniciais dados (x_0, y_0).

Se os declives médios $\bar{f}_{i,j}$ pudessem ser calculados de forma exata, os resultados obtidos seriam exatos. No entanto, para calcular o valor médio de $f(x,y)$ num intervalo seria

necessário conhecer a função $y(x)$, mas quando se usam métodos numéricos é porque não existem métodos para encontrar a expressão analítica dessa função. Os diferentes métodos numéricos que existem para resolver equações diferenciais correspondem a diferentes esquemas para estimar o valor médio aproximado da função $f(x,y)$ em cada intervalo.

3.2.1 Método de Euler

No método de Euler admite-se que $\bar{f}_{i,i+1} \approx f(x_i, y_i)$. Ou seja, o valor médio de f em cada intervalo é aproximado pelo valor de $f(x_i, y_i)$ no ponto inicial do intervalo. Realizando os cálculos no Maxima para o caso considerado acima, em que $f(x,y) = (x-1)(x-3) - y$ e com condição inicial $y(0) = 1$, pode usar-se uma lista p para armazenar as coordenadas dos pontos; inicia-se a lista com o ponto inicial e define-se a função $f(x,y)$ dada:

```
(%i1) p: [[0,1]]$
(%i2) f(x,y) := (x-1)*(x-3)-y$
```

Os quatro pontos seguintes na sequência são acrescentados usando um ciclo e usando as relações de recorrência (3.14) com $h = 1$:

```
(%i3) for i thru 4 do
        ([xi,yi]: last(p),
         p: endcons ([xi + 1, yi + f(xi, yi)], p))$
(%i4) p;
(%o4)        [[0, 1], [1, 3], [2, 0], [3, - 1], [4, 0]]
```

O gráfico na figura 3.3 mostra o resultado obtido (segmentos de reta e pontos), comparando-o com a solução exata (curva contínua). Observa-se que a solução obtida com o método de Euler não é uma aproximação muito boa. No entanto, reduzindo o valor dos incrementos em x de $h = 1$ para um valor menor, deverá ser possível obter uma aproximação melhor. É conveniente reduzir gradualmente o valor de h e comparar as soluções com as obtidas com valores maiores de h; se a solução obtida não variar significativamente quando h é reduzido, essa solução deverá ser uma boa aproximação da solução verdadeira.

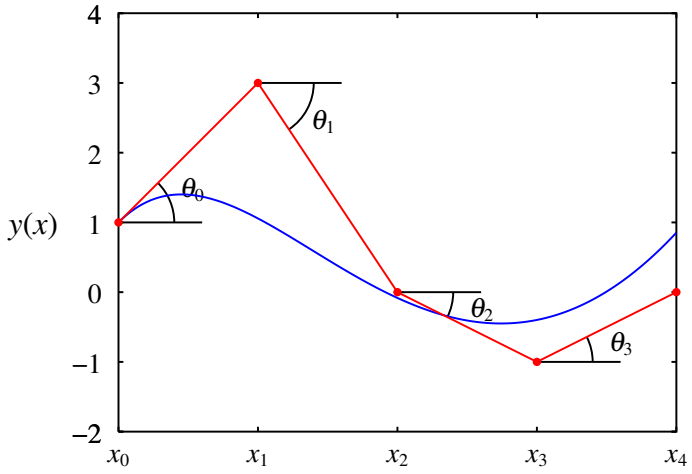

Figura 3.3: Método de Euler.

Exemplo 3.1
A carga armazenada num condensador ligado a uma pilha e a uma resistência é uma função Q que depende do tempo t e verifica a seguinte equação diferencial

$$\frac{dQ}{dt} = 1.25 - Q$$

No instante inicial, $t = 0$, a carga no condensador é nula. Usando o método de Euler, encontre uma sequência que aproxime $Q(t)$ no intervalo $0 \leq t \leq 6$.

Resolução. Neste caso, a função f que define a derivada não depende da variável independente t. Os valores iniciais e a função que define a derivada são:

```
(%i5) p: [[0,0]]$
(%i6) f(Q) := 1.25 - Q$
```

Começando com incrementos de tempo $h = 0.1$, são necessárias 60 iterações para terminar em $t = 6$.

```
(%i7) for i thru 60 do
        ([t0,Q0]: last(p),
         p: endcons ([t0 + 0.1, Q0 + 0.1*f(Q0)], p))$
```

O último ponto na lista p é:

```
(%i8) last(p);
(%o8)          [5.999999999999995, 1.247753737125107]
```

Convém repetir o processo, com um valor menor de h, por exemplo 0.01 e comparar o novo resultado com o resultado anterior:

```
(%i9) q: [[0,0]]$
(%i10) for i thru 600 do
         ([t0,Q0]: last(q),
          q: endcons ([t0 + 0.01, Q0 + 0.01*f(Q0)], q))$
(%i11) last(q);
(%o11)         [5.999999999999917, 1.246993738385861]
```

A pesar de que o resultado final coincide em 4 algarismos significativos nos dois casos, convém comparar as duas sequências completas. As duas soluções obtidas com $h = 0.1$ e $h = 0.01$ estão armazenadas nas listas p e q e podem ser representadas num gráfico. A figura 3.4 mostra que a discrepância entre as duas soluções obtidas não é muito grande, concluindo-se que a solução com $h = 0.01$ deverá estar já bastante próxima da solução verdadeira. O gráfico foi obtido usando o seguinte comando

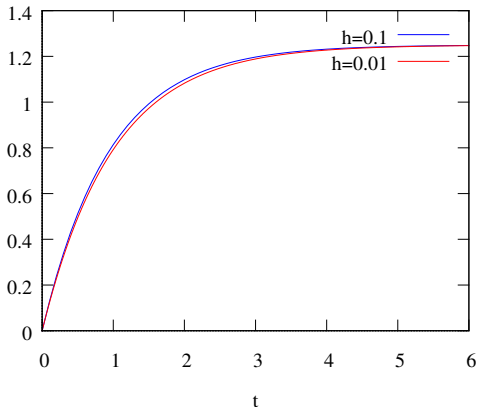

Figura 3.4: Soluções pelo método de Euler.

```
(%i12) plot2d([[discrete, p],[discrete, q]],
               [xlabel, "t"],[legend, "h=0.1", "h=0.01"])$
```

3.2.2 Método de Euler melhorado

A maior discrepância entre a sequência de pontos que aproximam a solução exata $y(x)$ na figura 3.2 e a sequência obtida usando o método de Euler, apresentada na figura 3.3, é no segundo ponto, (x_1, y_1). O valor médio do declive no primeiro intervalo, que era aproximadamente igual a zero, no método de Euler foi substituído por 2, que é o declive no ponto inicial (x_0, y_0). A figura 3.2 mostra que mantendo um valor fixo $y = y_0 = 1$, o declive $f(x,y)$ muda de um valor positivo em $x_0 = 0$ para um valor negativo em $x_1 = 1$. Se o valor do declive médio fosse aproximado pela média entre esses dois declives, o resultado seria muito melhor.

O método de Euler melhorado consiste em admitir que o declive médio $\bar{f}_{i,i+1}$ é igual à média entre os declives no ponto inicial (x_i, y_i) e no ponto final (x_{i+1}, y_{i+1}):

$$\boxed{\bar{f}_{i,i+1} \approx \frac{f(x_i,y_i)+f(x_{i+1},y_{i+1})}{2}} \tag{3.15}$$

O problema com esta equação é que para poder calcular $f(x_{i+1},y_{i+1})$ seria necessário saber o valor de y_{i+1}, mas esse valor é desconhecido enquanto a solução da equação não for conhecida. É possível fazer uma estimativa inicial do valor que y_{i+1} poderá ter e a seguir melhorar essa estimativa. Uma primeira estimativa muito rudimentar consiste em dizer que y_{i+1} é igual a y_i. Com essa abordagem, a resolução da mesma equação $y' = (x-1)(x-3) - y$ considerada na secção anterior, com valor inicial $y(0) = 1$ e com 5 pontos no intervalo $0 \le x \le 4$, pode ser feita assim: começa-se por iniciar a lista de pontos e definir a função para o declive

```
(%i13) p: [[0,1]]$
(%i14) f(x,y) := (x-1)*(x-3)-y$
```

Foi necessário definir novamente a função $f(x,y)$, pois a definição que foi inserida na secção anterior foi logo substituída pela função $f(Q)$ do exemplo 3.1. Basta repetir as 4 iterações mas usando agora a expressão (3.15) nas relações de recorrência (3.14)

```
(%i15) for i thru 4 do
         ([xi,yi]: last(p),
          p: endcons ([xi+1, yi + (f(xi,yi)+f(xi+1,yi))/2], p))$
(%i16) p;
                        3          1          1        3
(%o16)     [[0, 1], [1, -], [2, - -], [3, - -], [4, -]]
                        2          2          2        2
```

A figura 3.5 mostra a sequência de pontos obtida, juntamente com a solução exata (curva contínua na figura). O resultado é muito melhor que o resultado obtido com o método de Euler. Este resultado pode ser melhorado se a estimativa inicial $y_{i+1} = y_i$ for melhorada com o resultado obtido a partir dessa estimativa rudimentar, como se explica a seguir.

A partir do valor inicial y_0, começa-se por fazer a primeira estimativa do valor de y_1, usando $y_{1,0} = y_0$, como nos cálculos acima, e calcula-se $y_{1,1} = y_0 + h f(x_0, y_{1,0})$. Esse

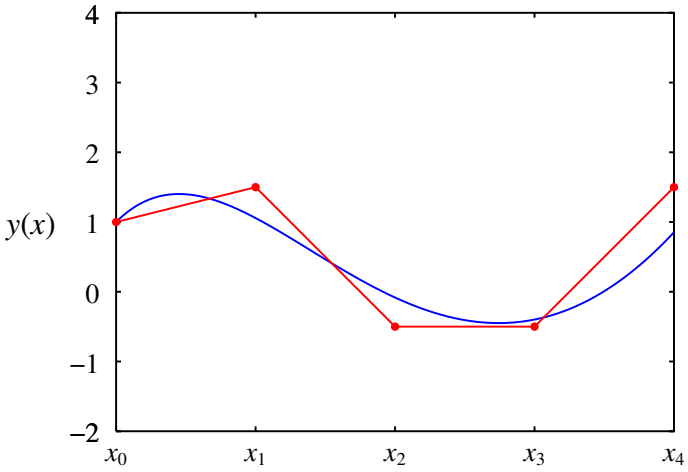

Figura 3.5: Solução exata e solução obtida com o método usado em (%i15).

valor estará mais próximo do valor real de y_1 do que a estimativa inicial $y_{1,0}$; assim sendo, espera-se que a sequência $y_{1,n} = y_0 + h\,f(x_0, y_{1,n-1})$ converja para um valor mais próximo do valor real y_1. Calculam-se alguns termos dessa sequência, até $|y_{1,n} - y_{1,n-1}|$ ser menor que uma precisão desejada. Nesse momento usa-se $y_1 = y_{1,n}$ como valor inicial para o seguinte intervalo e o processo repete-se iterativamente em todo o intervalo de x. No caso do problema resolvido acima, com precisão de 0.0001, o método pode ser implementado assim:

```
(%i17) f(x,y) := float ((x-1)*(x-3)-y)$
(%i18) p: [[0,1]]$
(%i19) for i thru 4 do
         ([xi,yi]: last(p),
          y1: yi,
          y2: yi + (f(xi,yi)+f(xi+1,y1))/2,
          while abs(y1-y2) > 0.0001 do
                (y1: y2, y2: yi + (f(xi,yi)+f(xi+1,y1))/2),
          p: endcons ([xi+1, y2], p))$
(%i20) p;
(%o20) [[0, 1], [1, 1.33331298828125], [2, .1111229788511992],
                  [3, - .2962674737252655], [4, .9012259028472571]]
```

A função f foi definida novamente, usando-se o comando `float`, para evitar obterr fracções com numeradores e denominadores muito grandes; nos exemplos anteriores foram feitos menos cálculos e, por isso, não produziram frações com números grandes. O resultado (figura 3.6) mostra que a solução obtida é muito melhor do que nas figuras 3.3 e 3.5, em relação à solução real.

A pesar de se ter usado uma precisão de 0.0001, não se pode garantir que a solução obtida esteja dentro dessa precisão em relação á solução real, já que a equação 3.15 é apenas uma aproximação e não o valor médio real da derivada. A aproximação pode ser melhorada se em vez de se fazer a média do declive em dois pontos, o declive fosse calculado em

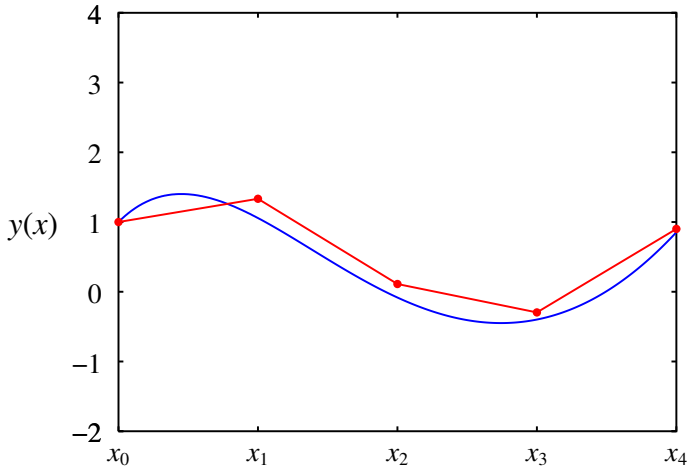

Figura 3.6: Solução exata e solução obtida com o método de Euler melhorado.

outros pontos nessa vizinhança e fosse feita uma média com diferentes pesos que podem ser definidos de forma a minimizar o erro. Existem muitos métodos diferentes que usam esse esquema; o mais popular é o que será descrito na seguinte secção.

3.2.3 Método de Runge-Kutta de quarta ordem

Neste método o valor médio do declive $\bar{f}_{i,i+1}$ obtém-se a partir da média dos declives em 4 pontos diferentes, com pesos diferentes (figura 3.7).

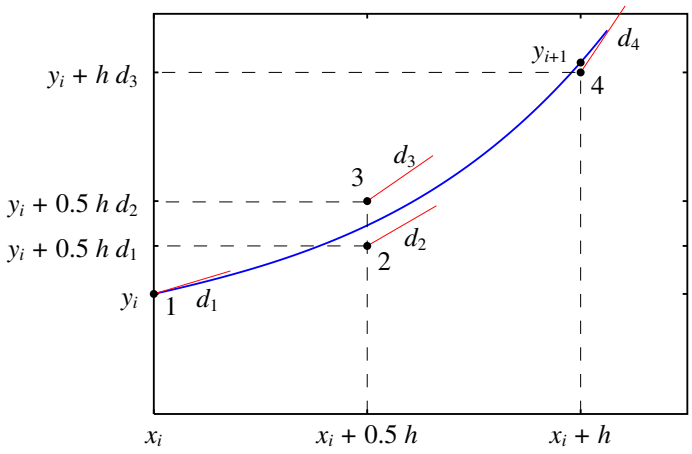

Figura 3.7: Os declives usados no método de Runge-Kutta.

Começa-se por calcular o declive no ponto inicial do intervalo, tal como no método de Euler:

$$d_1 = f(x_i, y_i) \tag{3.16}$$

a seguir, realiza-se um deslocamento na direção desse declive, avançando-se uma distância

$h/2$ no eixo das abcissas, até um ponto 2 (ver figura 3.7). Nesse ponto 2, calcula-se um segundo valor do declive

$$d_2 = f(x_i + h/2, y_i + (h/2)d_1) \tag{3.17}$$

Esse novo valor do declive é usado novamente, para realizar outro deslocamento a partir do ponto inicial, avançando $h/2$ na direção do eixo das abcissas, até um outro ponto 3, onde é calculado um terceiro valor do declive

$$d_3 = f(x_i + h/2, y_i + (h/2)d_2) \tag{3.18}$$

seguindo a direção desse declive d_3, realiza-se um terceiro deslocamento, a partir do ponto inicial, desta vez avançando uma distância h no sentido do eixo das abcissas, para chegar até um ponto 4, onde se calcula um quarto valor do declive

$$d_4 = f(x_i + h, y_i + h\,d_3) \tag{3.19}$$

Pode mostrar-se que para minimizar o erro cometido, o valor médio do declive deve ser aproximado pela seguinte combinação linear dos quatro declives calculados:

$$\bar{f}_{i,i+1} \approx \frac{1}{6}(d_1 + 2d_2 + 2d_3 + d_4) \tag{3.20}$$

no exemplo da figura 3.7, esse valor médio da derivada desloca o ponto inicial até o ponto 4, que está bastante próximo da solução exata da equação.

Para aplicar este método à mesma equação $y' = (x-1)(x-3) - y$ considerada nas secções anteriores, com valor inicial $y(0) = 1$ e com 5 pontos no intervalo $0 \le x \le 4$, inicia-se a lista de pontos e define-se a função para o declive:

```
(%i21) p: [[0,1]]$
(%i22) h: 1$
```

Neste caso foi dado explicitamente o valor do comprimento dos intervalos, para que as expressões seguintes sejam mais claras.

A seguir realizam-se as 4 iterações necessárias para chegar até o ponto $x = 4$. Em cada iteração calculam-se os quatro declives d_1, d_2, d_3 e d_4 e a média com os pesos usados neste método.

```
(%i23) for i thru 4 do
         ([xi,yi]: last(p),
          d1: f(xi, yi),
          d2: f(xi+h/2, yi+(h/2)*d1),
          d3: f(xi+h/2, yi+(h/2)*d2),
          d4: f(xi+h, yi+h*d3),
          p: endcons ([xi+h, yi + h*(d1+2*d2+2*d3+d4)/6], p))$
(%i24) p;
(%o24) [[0, 1], [1, 1.020833333333333], [2, -.09635416666666652],
                  [3, -.3902994791666666], [4, .8744710286458335]]
```

A figura 3.8 mostra a sequência de pontos obtida, juntamente com a solução exata (curva contínua na figura). O resultado é bastante bom, a pesar do valor tão elevado que foi usado para os incrementos da variável x. No caso da equação diferencial resolvida neste exemplo, existem métodos analíticos que permitem encontrar a expressão para a curva contínua que foi apresentada na figura; nos casos em que não é possível encontrar a solução analítica, o método de Runge-Kutta de quarta ordem é uma boa opção para encontrar uma boa aproximação numérica.

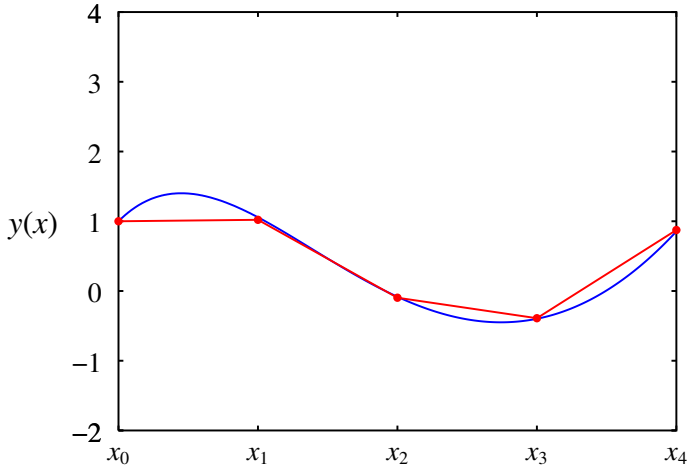

Figura 3.8: Solução exata e solução obtida com o método de Runge-Kutta.

Já existe uma função `rk` predefinida no Maxima, que executa o método de Runge-Kutta de quarta ordem. No exemplo acima, bastava indicar a expressão de $f(x,y)$, o nome que identifica a variável independente, o seu valor inicial e o intervalo em que se quer obter a solução, incluindo o valor dos incrementos h. Nomeadamente, a mesma lista obtida com os comandos (%i17) até (%i21) podia ser obtida com um único comando:

```
(%i25) rk ((x-1)*(x-3)-y, y, 1, [x,0,4,1]);
(%o25) [[0.0, 1.0], [1.0, 1.020833333333333],
[2.0, - .09635416666666652], [3.0, - .3902994791666666],
[4.0, .8744710286458335]]
```

3.3 Sistemas de equações diferenciais

Os métodos descritos nas secções anteriores podem ser generalizados facilmente a um sistema de várias equações, quando sejam conhecidas condições iniciais para todas as variáveis. Tal como se explica no inicio do capítulo, basta admitir que y é um vetor e que f é uma função vetorial.

Exemplo 3.2
Usando o método de Runge-Kutta de quarta ordem, encontre a solução do seguinte sistema de equações diferenciais:

$$\begin{cases} \dfrac{dx}{dt} = 4 - x^2 - 4y^2 \\ \dfrac{dy}{dt} = y^2 - x^2 + 1 \end{cases}$$

no intervalo, $0 \leq t \leq 4$, com condições iniciais $x_0 = -1.25$e $y_0 = 0.75$, em $t_0 = 0$.

Resolução. A variável independente é t e a variável dependente é o vetor $\vec{r} = (x, y)$, que no Maxima é representado como uma lista com duas partes, [x, y]. A expressão $\vec{f}$ que define as derivadas das variáveis dependentes também será uma lista com duas partes, $[4 - x^2 - 4y^2, y^2 - x^2 + 1]$, mas é mais conveniente (para poder generalizar ao caso de n variáveis) representá-la em função das componentes da lista r:

```
(%i26) f(t,r) := [4-r[1]^2-4*r[2]^2, r[2]^2-r[1]^2+1]$
```

foi dada também a variável independente t como argumento da função, a pesar de que neste caso $\vec{f}$ não depende de t, para que os comandos usados a seguir possam ser reutilizados em casos mais gerais.

Com as condições iniciais $t_0 = 0$ e $\vec{r}_0 = (-1.25, 0.75)$ cria-se a lista onde serão inseridos os resultados das iterações

```
(%i27) p: [[0,-1.25,0.75]]$
```

Observe-se que a lista inicial [0,-1.25,0.75] e todas as outras listas que serão acrescentadas, incluem a variável independente t e o vetor dependente $\vec{r}$. Para extrair apenas a variável independente t usa-se o comando `first`, que extrai o primeiro elemento de uma lista e para extrair o vector $\vec{r}$ usa-se o comando `rest`, que produz uma nova lista excluindo o primeiro elemento da lista original; isto é, rest([0, -1.25, 0.75]) produz [-1.25, 0.75].

Usando incrementos $h = 0.02$ para a variável independente, será necessário realizar 200 iterações para terminar em $t = 4$.

```
(%i28) h: 0.02$
(%i29) for i thru 200 do
         (ti: first (last(p)),
         ri: rest (last(p)),
         d1: f (ti, ri),
         d2: f (ti+h/2, ri+(h/2)*d1),
         d3: f (ti+h/2, ri+(h/2)*d2),
         d4: f (ti+h, ri+h*d3),
         p: endcons (cons (ti+h, ri+h*(d1+2*d2+2*d3+d4)/6), p))$
```

O comando `cons` é semelhante a `endcons`, mas insere um elemento no início de uma lista, em vez de no fim. Neste caso `cons` foi usado para inserir o valor de t na lista [x, y],

produzindo a lista $[t, x, y]$, que foi logo inserida no fima da lista p dos resultados.

O último ponto na solução obtida é:

```
(%i30) last(p);
(%o30) [4.000000000000003, 1.232365393486131, - .7493152366008236]
```

O comando `rk` também pode ser usado para obter o mesmo resultado:

```
(%i31) p:rk([4-x^2-4*y^2,y^2-x^2+1],[x,y],[-1.25,0.75],[t,0,4,h])$
```

Para imprimir os resultados é conveniente usar o comando `printf`. Esse comando aceita um formato que diz como devem ser imprimidos os dados. Neste exemplo, usando o formato `"~{~{~f~^, ~}~%~}"` conseguem-se imprimir os 3 elementos t, x e y de cada iteração numa linha separada, separados por uma vírgula e um espaço. Os dois símbolos `~{` e `~}` usam-se para delimitar um formato que deve ser aplicado recursivamente a todos os elementos de uma lista, neste caso a lista p. Dentro desses símbolos, o formato `"~{~f~^, ~}~%"` determina que o formato `"~f~^, "` seja aplicado novamente a cada elemento das sublistas de p, neste caso a t_i, x_i e y_i, e a seguir se insira um fim de linha (símbolo `~%`). O símbolo `~f` é usado para escrever números de vírgula flutuante e o símbolo `~^` indica que o que vem a seguir só será escrito se não se tiver chegado ao último elemento; ou seja, só será escrita uma vírgula após t_i e x_i, mas não após y_i.

Para imprimir unicamente os 9 resultados para $t = 0, 0.5, \ldots, 4$, extraem-se unicamente esses elementos da lista p, usando o comando `makelist`:

```
(%i32) printf(true,"~{~{~f~^, ~}~%~}", makelist(p[25*i+1],i,0,8))$
0.0, -1.25, 0.75
0.5, -1.1773384444565893, 0.8294212227394446
1.0, -1.5811924578377141, 0.818421880110717
1.5, -1.5173593339612323, 0.037780513984502606
2.0, -0.0005587219405601279, 0.06781851774672519
2.5, 1.4685416405394016, 0.14800909710023502
3.0, 1.6981292199163442, -0.7368848754678627
3.5, 1.188041502209304, -0.8582807863663763
4.0, 1.232365393486131, -0.7493152366008236
```

Finalmente, pode ser conveniente gravar esses resultados num ficheiro. Por exemplo, os programas de folha de cálculo conseguem ler dados como estão apresentados acima, dentro de ficheiros do tipo CSV (*Comma Separated Values*). Para gravar o que é apresentado no ecrã num ficheiro, usa-se o comando `with_stdout`

```
(%i33) with_stdout ("resultados.csv",
       printf(true,"~{~{~f~^, ~}~%~}", makelist(p[25*i+1],i,0,8)))$
```

Os resultados ficam gravados no ficheiro resultados.csv.

Bibliografia

[1] Davenport, J.H., Siret, Y. e Tournier, E. *Computer Algebra: Systems and Algorithms for Algebraic Computation*. Academic Press, San Diego, U.S.A., 1988.

[2] Chapra, S. C. *Numerical Methods for Engineers*. Sexta edição, McGraw-Hill, Boston, U.S.A., 2010.

[3] Kiusalaas, J. *Numerical Methods in Engineering with Python*. Segunda edição, Cambridge University Press, Cambridge, U.K., 2010.

[4] Kreyszig, E. *Advanced Engineering Mathematics*. Nona edição, John Wiley & Sons, Hoboken, U.S.A., 2006.

[5] Redfern, D., Chandler, E. e Fell, R. N. *Macsyma ODE Lab Book*. Jones and Bartlett Publishers, Sudbury, U.S.A., 1998.

[6] Villate, J. E. *Introdução aos Sistemas Dinâmicos: uma abordagem prática com Maxima*. Edição do autor, Porto, 2007. (disponível em http://def.fe.up.pt)

[7] Villate, J. E. *Dinâmica e Sistemas Dinâmicos*. Edição do autor, Porto, 2013. (disponível em http://def.fe.up.pt)

Índice