3.9 Iteração, mais alguns exemplos

O método iterativo que conhecemos na seção anterior tem uma importância fundamental na física e também em outras disciplinas exatas. Por isso, estudaremos neste item, primeiramente, outra vez dois exemplos que foram já analisados em seções anteriores. No parágrafo 3.9.3 vamos discutir, brevemente, dois algoritmos para isolar iterativamente as raízes reais de uma função f(x).

3.9.1 Queda de uma esfera através de um fluido

Este problema já foi analisado, com bastante detalhe, na seção 2.4. Uma esfera de massa m e raio R cai com velocidade inicial zero a partir de x = 0.
Nós subdividimos a distância H da queda em n intervalos, cada um de comprimento h = H/n.

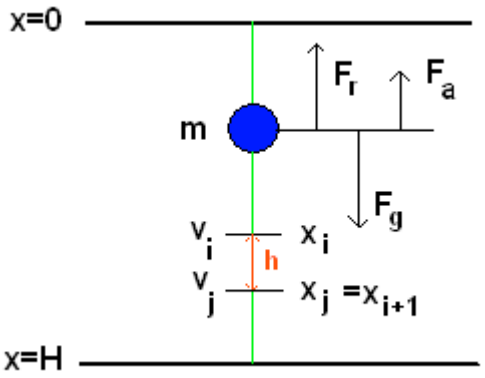


Fig. 3.9-1

Para cada intervalo calculamos a velocidade média segundo $(v_i + v_j)/2$. Ao longo de cada intervalo, consideramos a aceleração como sendo constante. A aceleração no intervalo número j (= intervalo-j) é dada por

$$a_j := (v_j - v_i)/(t_j - t_i) = g[u-((v_i+v_j)/(2v_1))^2] \qquad (1)$$

Compare-a com a equação 2.4-4 no parágrafo 2.4.1, onde encontra também todas as explicações necessárias, por ex. sobre v_1.

O tempo para cair pelo intervalo-j é

$$t_j - t_i = (2h)/(v_i + v_j) \qquad (2)$$

Esta expressão introduzimos em equação (1), juntamente com a abreviatura

$$b := g*h/(2v_1)^2 \qquad (3)$$

Chegamos, assim, à seguinte fórmula de iteração:

$$v_{i+1} = [(v_i^2 + 4buv_1^2(1+b))^{1/2} - bv_i]/(1+ b) \qquad (4)$$

Em vez de v_j temos escrito v_{i+1}, além disso temos $u := 1-\rho/\rho_c$ e $v_1^2 := 8Rg\rho_c/(3C\rho)$, onde ρ = densidade do fluido (1000 kg/m^3 para água), ρ_c = densidade da esfera (800kg/m^3), R = raio (4 mm), C = 0.4 e g = 9.81 m/s^2 .

Na seção 2.4 vimos que esta esfera precisa 0,25 s para cair 0,1975 m e que sua velocidade neste momento é de 1,22989 m/s. Com o seguinte programa, que trabalha com uma exatidão bem menor do que a função **ode** do MuPAD, obtemos, porém, os mesmos resultados.

Para determinar o tempo da queda, temos que somar os tempos parciais t_j , gastos nos n intervalos, ver eq. (2). Este tempo calculamos da seguinte maneira:

$$T = \sum_{j=1}^{n} t_j = \frac{2H}{n} \sum_{i=0}^{n-1} \frac{1}{v_i + v_{i+1}} \qquad (5)$$

- **Reset(): // Queda de uma esfera em água**

```
rc:=7800:rfl:=1000:
R:=0.004:C:=0.4:g:=9.8: H:=0.2:v0:=0:
n:=100:s:=0:
u:=1-rfl/rc:
v1:=8*R*g*rc/(3*rfl*C):
b:=g*H/(2*n*v1):
d:=4*b*u*v1*(1+b):
v(0):=v0:
for i from 1 to n do
v(i):=(sqrt(v(i-1)^2+d)-b*v(i-1))/(1+b):
s:=s+1/(v(i-1)+v(i)):://cálculo da soma
```

```
t(i):=1/(v(i-1)+v(i)):elementos para "sum"
end_for:

print(Unquoted,"o tempo é ", 2*H*s/n, " segundos");

//com "Unquoted" são suprimidas as aspas

print("T =",(2*H/n)*sum(t(j),j=1..n));//soma com "sum"

o tempo é , 0.2521110898, segundos
"T =", 0.2521110898
```

3.9.2 O pêndulo simples com amplitudes arbitrárias

Para este parágrafo, você deveria recapitular a seção 3.2, na qual estudamos a simples equação y"(t) = -sen y(t) com valores iniciais para y(0) e y'(0).

Nem MuPAD nem pessoa viva sabe resolver esta equação em forma "fechada". Uma solução aproximada obtém-se somente por meios numéricos iterativos. Um método iterativo muito simples vamos desenvolver neste item. Trata-se, na realidade, de uma simples derivação do método anterior, pois é só o caso de uma queda com vínculo.

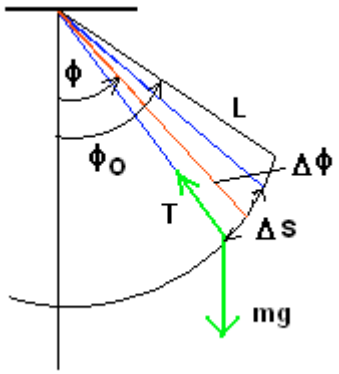


Fig. 3.9-2

Agora subdividimos, como mostrado na Fig. 3.9-2, a amplitude φ_o em n partes de igual tamanho $\Delta\varphi = \varphi_o/n$.

O pêndulo precisa Δt segundos para percorrer o ângulo $\Delta\varphi = \Delta s/L$. A soma de todos os elementos Δt dá o período $T := T_o \cdot K_o$, onde K_o é um fator de correção, dependendo do ângulo φ_o, e $T_o = 2\pi\,(L/g)^{1/2}$ é o período do pêndulo simples. O fator K, introduzido no parágrafo 3.2.2, é em $\pi/2 = 1.570796$ vezes maior do que nosso K_o.

Suponhamos que a aceleração tangencial é constante num intervalo Δt. Temos

$$a_t = (v_{i+1} - v_i)/\Delta t = g\cdot\mathrm{sen}\varphi \qquad (6)$$

A velocidade média no intervalo Δt é $(v_i + v_{i+1})/2$, e o arco, passado pelo pêndulo em Δt segundos será $\Delta s = (v_i + v_{i+1})\,\Delta t/2 = L\,\Delta\varphi$. Assim, obtemos

$$v_{i+1} = v_i + g\,\Delta t\,\mathrm{sen}\varphi \qquad (7)$$
$$\Delta t = 2\,L\,\Delta\varphi/(v_i + v_{i+1}) \qquad (8)$$

Substituindo Δt da primeira equação pelo Δt da segunda, resulta a seguinte fórmula de iteração para a velocidade

$$v_{i+1} = (v_i^2 + 2\,L\,\Delta\varphi\,g\,\mathrm{sen}\varphi)^{1/2} \qquad (9)$$

A soma de todos os Δt entre $\varphi = \varphi_o$ e $\varphi = 0$ proporciona o tempo T/4, e o período completo é

$$T = \frac{8L\phi_o}{n}\sum_{\phi_o}^{0}\frac{1}{v_i + v_{i+1}} \qquad (10)$$

O fator de correção vem dado por

$$K_0 = \frac{T}{T_0} = \frac{4\phi_0}{n\pi}\sqrt{gL}\sum_{\phi_o}^{0}\frac{1}{v_i + v_{i+1}} \qquad (11)$$

K_o depende, aparentemente, de g e L. Mas isso não é o caso, pois, se introduzirmos uma grandeza u sem dimensão como

$$v := u\,(g\,L)^{1/2}, \qquad (12)$$

podemos eliminar $(g\,L)^{1/2}$ e nós obtemos $v_i + v_{i+1} = (g\,L)^{1/2}\,(u_i + u_{i+1})$, onde pusemos

$$u_{i+1} := (u_i^2 + 2\,\Delta\varphi\,\mathrm{sen}\varphi)^{1/2}.$$

Finalmente, obtemos

$$K_0 = \frac{4\phi_0}{n\pi} \sum_{\phi_0}^{0} \frac{1}{u_i + u_{i+1}} \qquad (13)$$

- ```
  reset():pêndulo com iteração
  fi1:=30://ângulo em graus
  pi:=float(PI):
  fi0:=float(fi1*pi/180):
  DIGITS:=8:
  n:=500: //muitos subdivisões!
  for i from 1 to n do

  dfi:=fi0/n:
  fi2:=fi0-dfi/2://ver Fig.3.9-2
  v(0):=0:t(0):=0:
  b:=2*dfi*sin(fi2-(i-1)*dfi):
  v(i):=sqrt(v(i-1)^2+b):
  t(i):=1/(v(i-1)+v(i)):

  end_for:

  K0:=4*fi0/(n*pi)*sum(t(j),j=1..n);

  K:=K0*pi/2// para comparar com a tabela na seção 3.2
  ```

```
1.0174244  (= K0)

1.5981664  (= K)
```

Para Δt suficientemente pequeno, a precisão do método de iteração pode produzir resultados com atı três ou quatro dígitos decimais corretos. (Devido a limitações de máquinas e dos limites inerentes dos mesmos métodos, não adianta sempre reduzir o valor do intervalo Δt. Com a redução de Δt, não obtém-se, necessariamente, valores cada vez mais precisos.)

Para aumentar a precisão temos que aplicar métodos com ordens de precisão mais altas, como os métodos implantados em MuPAD.
```

3.9.3 Métodos iterativos para encontrar os zeros de uma função.

Nas mais diversas áreas das ciências exatas ocorre, freqüentemente, o problema de determinar os zeros de uma função f(x) (ou as raízes de uma equação f(x) = 0). Os zeros reais podemos obter, com precisão limitada, da representação gráfica da função, pois, os zeros reais são representados pelas abscissas dos pontos onde uma curva intercepta o eixo-x.

Para a maioria das funções existem unicamente métodos numéricos para encontrar os seus zeros (ou as raízes das equações correspondentes). Estes métodos são, normalmente, iterativos. Eles partem de uma aproximação inicial para o zero e, em seguida, refinam este valor iterativamente até se obter uma aproximação para o zero correto (que não se conhece) com uma precisão ε prefixada.

A análise gráfica da função f(x) proporciona, em geral, um valor inicial adequado para começar com a iteração. Ver a figura 3.9-3.

Dos muitos métodos existentes, vamos estudar aqui somente o **Método de Newton** e o **Método da bissecção**.

Geometricamente, no primeiro método traçamos a reta tangente no ponto $(x_0,f(x_0))$ e, em seguida, tomamos **o zero da tangente como o novo valor aproximado** para o zero da função f(x). Assim iterando, obtemos uma seqüência $\{x_i\}$ de valores que se aproximam da raiz exata da equação f(x) = 0.

Os elementos x_i determinamos, neste método, por a seguinte fórmula de iteração:

$$x_{i+1} = x_i - f(x_i)/f'(x_i) \qquad (14)$$

(A equação da tangente é dada por $t(x) = f(x_i) + f'(x_i)(x-x_i)$.
Isolando a variável x na equação **t(x) := 0**, resulta $x = x_i - f(x_i)/f'(x_i)$.
Finalmente, pomos $x_{i+1} := x$, para obter a equação (14))

Exemplo:

Determine uma seqüência de valores x_i que converge para a raiz exata da equação

$x^3 - 2x - 5 = 0$, e ilustre os primeiros passos da iteração por representações gráficos.

Esta equação é chamada segundo o matemático inglês *John Wallis* (1616-1703).

Ela é utilizada para testar diversos métodos de iteração, e o seu zero foi calculado com mais de 4000 dígitos decimais corretos. Você tem que calcular somente um valor aproximado com 15 casas decimais corretas.

Solução:

- ```
  f:=x->x^3-2*x-5:// John Wallis (1616-1703)
  g:=plot::Function2d(f(x),x=-1..3,Color=RGB::Red):
  plot(g)
  ```

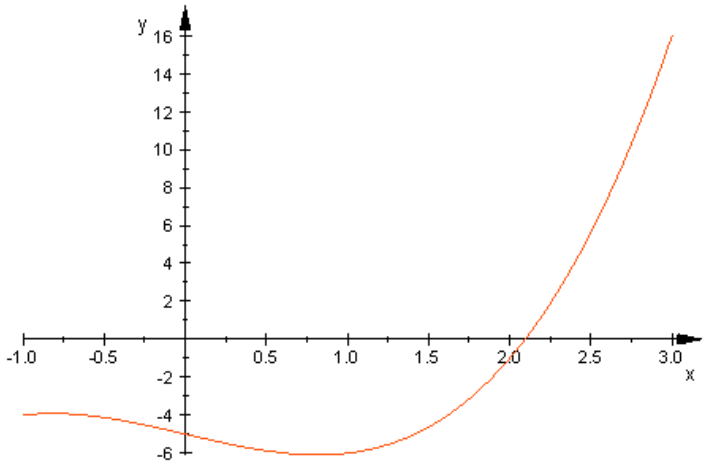


Fig. 3.9-3

O zero da função está perto de 2,1. Para o seguinte passo utilizamos, porém, x0 = 2,5, para mostrar que, neste caso, não é necessário usar o melhor valor possível como ponto inicial da iteração.

- ```
  x0:=2.5:
  tang:=f'(x0)*(x-x0)+f(x0):
  t1:=plot::Function2d(tang(x),x =1.5..3.5):
  plot(g,t1)
  ```

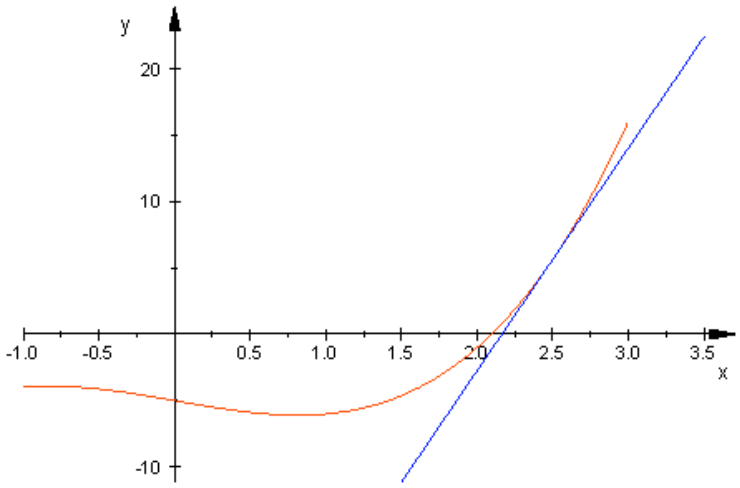


Fig. 3.9-4

Vemos que o zero da reta tangente pode servir como valor inicial do próximo passo da iteração. Isso é equivalente ao uso da equação (14):

- ```
  x1:=float(x0-f(x0)/f'(x0));

  float(f(x0));

  float(f(x1));
  ```

```
2.164179104

5.625

0.8079451262
```

Vemos que $f(x_1)$, com x_1 = 2,164179104, está mais perto de zero do que o valor anterior com x_0 = 2,5. Na Fig. 3.9-5 traçamos a função f(x) juntamente com a tangente no ponto $(x_1,f(x_1))$.

- ```
  tang:=f'(x1)*(x-x1)+f(x1):
  t2:=plot::Function2d(tang(x),x =1.5..2.5):
  plot(g,t2)
  ```

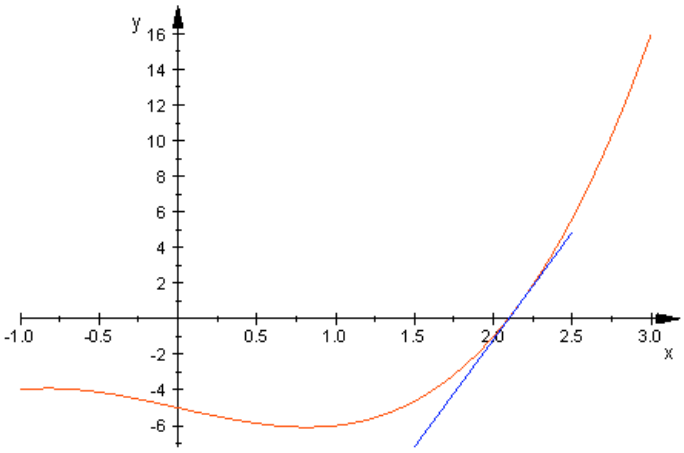


Fig. 3.9-5

O próximo valor, x_2 , está ainda mais perto do valor buscado do que x_1:

- ```
  x2:=float(x1-f(x1)/f'(x1));

  float(f(x2));
  ```

```
2.097135356

0.02888172122
```

Finalmente, incorporamos a fórmula (14) num programa, para calcular, sem gráficos, n valores da seqüência $\{x_i\}$ com 25 casas decimais. Como valor inicial usamos $x_0 = 3$.

- ```
  f:=x->x^3-2*x-5:

  DIGITS:=25:
  n:=7:
  x[0]:=3:
  for i from 0 to n do //n=número máximo de iterações

  x[i+1]:=x[i]-f(x[i])/f'(x[i]);
  print(float(x[i]),float(f(x[i])));

  end_for;
  ```

Os valores na primeira coluna são os elementos x_i , os da segunda coluna são os valores da função de Wallis.

```
                                3.0, 16.0

                             2.36, 3.424256

  2.12719678015881649080822 4, 0.37109984624719729714211 12

 2.09513603693363411505338 1, 0.00652662595368705080876 5816

                    2.094551673824267730291679,
                0.000002146143144271548405971071

                    2.094551481542347406130902,
            0.000000000000232321403204771757030 6866

 2.094551481542326591482387, 2.722390729583987529002929e-27

 2.094551481542326591482387, 3.738289469450879297692366e-55
```

Por meio do mesmo programa e com a mesma aproximação inicial de $x_0 = 3$ podemos rapidamente encontrar os zeros das seguintes equações:

1. x - sen x - 0,5 = 0 (1,4973003...)
2. $e^{-x} + x/5 - 1 = 0$ (4,9651...)

O outro método tem o nome do matemático austríaco **Bolzano** (1781-1848). Este método, também chamado de **Método da bissecção**, consiste em buscar um intervalo [a,b] que contém uma raiz da equação f(x) = 0, e reduzir a amplitude deste intervalo até atingir a precisão requerida. A redução dos intervalos se faz dividindo sucessivamente o intervalo atual ao meio.

Para escrever um programa para o método de Bolzano, começamos com valores iniciais para a e b. O centro do intervalo é x = (a+b)/2, ou seja, x é simplesmente a média aritmética entre a e b.

Seja f(a) o valor da função no ponto x = a. Se f(x), com x = (a+b)/2, for zero, então temos também f(a)·f(x) = 0, e x seria a raiz da equação em estudo, p. ex., da equação $e^{-x} + x/5 - 1 = 0$.

Normalmente, x =(a+b)/2 não vai ser a raiz x_0 buscada.
Se x_0 fica no lado **esquerdo** de x, então temos **f(a)·f(x) < 0** e nós continuamos com nossa busca só no intervalo [a,x], ou seja, poremos b:=x.

Em seguida dividimos esse novo intervalo em duas partes iguais.

Se x_0 fica no lado **direito** de x, então temos **f(a)·f(x) > 0** e nós continuamos com nossa busca só no intervalo [x,b], ou seja, vamos por a:=x. Em seguida dividimos esse novo intervalo em duas partes iguais.

Se desejarmos encontrar uma raiz com precisão $|x_0 - x| < \varepsilon$, entyo temos que efetuar, no mínimo, N divisões, onde o número N é calculado pela relação

$$N > (\log(b-a)-\log(\varepsilon))/\log(2). \qquad (15)$$

Exemplo: Se o intervalo inicial [a,b] for [4,6], temos que efetuar no mínimo 11 iterações, para obter $|x_0 - x| <= b-a < 10^{-3}$, ou seja com precisão de 3 dígitos decimais. De fato, devemos efetuar 12 iterações.

Isso significa que a convergência do método da bissecção é muito lenta, especialmente, se o intervalo inicial for muito grande e se ε for muito pequeno. O número de iterações é, geralmente, muito grande, se queremos um bom valor aproximado para a raiz da equação em estudo.

Para escrever um programa para o MuPAD temos que mudar um pouco a nossa estratégia, pois parece que MuPAD tem problemas com condições que incluem > ou < numa comparação. Então temos que usar <>, e isso significa uma outra formulação do algoritmo.
Se f(a) e f(x) têm sinais diferentes, então a raiz tem que estar entre a e x. Continuamos, neste caso, com o mesmo valor de a e substituimos b por x, ou seja b:=x.

Se f(a) e f(x) têm sinais iguais, então vamos manter b como limite direito e usaremos x como o novo limite esquerdo.

Um intervalo com $f(a_0) = f(b_0)$ deve ser excluido desde o princípio:

```
if sign(f(a[0]))= sign(f(b[0]))then

print(Unquoted,"O intervalo nao serve!")
```

O que acabo de expor está incorporado no seguinte **Programa** para o algorítmo da **bissecção** . Os passos da iteração mostram primeiro o limite esquerdo, pois a média aritmética entre a e b, logo o limite b e a amplitude do intervalo [x_0, x], ou seja a distância entre os limites. Ao final do programa foi acrescentado o código para traçar os intervalos.

- ```
  reset():://Bolzano (método da bissecção)

  f:= x-> exp(-x)+x/5-1:
  a[0]:= 4: b[0]:=6:
  DIGITS:=25:
  eps:=0.001:
  N:=12:

  if sign(f(a[0]))= sign(f(b[0]))then
  print(Unquoted,"O intervalo nao serve!")

  else

  for i from 1 to N do
  x[i]:=(a[i-1]+b[i-1])/2:

  if sign(f(a[i-1]))<> sign(f(x[i]))
  then b[i]:=x[i];a[i]:=a[i-1]:

  else

  a[i]:=x[i]:b[i]:=b[i-1]:

  end_if:

  dist:=float(abs(a[i]-b[i])):
  print(float(a[i-1]),float(x[i]),float(b[i-1]),dist);

  if dist < eps then break

    end_if:
  end_for:
  ```
  ```

```
//traçado dos intervalos
for k from 0 to 10 do
l[k]:=plot::Line2d([a[k],k],[b[k],k]):
end_for:
plot(l[k] $ k=0..10)// $ é um operador para os índices
/*traça os intervalos de baixo para cima, ver o seguinte
programa para a seqüência inversa*/
end_if:
```

a	centro	b	dist
4.0,	5.0,	6.0,	1.0
4.0,	4.5,	5.0,	0.5
4.5,	4.75,	5.0,	0.25
4.75,	4.875,	5.0,	0.125
4.875,	4.9375,	5.0,	0.0625
4.9375,	4.96875,	5.0,	0.03125
4.9375,	4.953125,	4.96875,	0.015625
4.953125,	4.9609375,	4.96875,	0.0078125

O valor de `4.9658203125` tem uma precisão de só 3 dígitos decimais: 4.965

```
//traçar os intervalos traça os intervalos de cima para
baixo


for k from 0 to 5 do

l[k]:=plot::Line2d([a[k],5-k],[b[k],5-k],Color=RGB::Red):

end_for:

plot(l[k] $ k=0..5)
end_if:
```

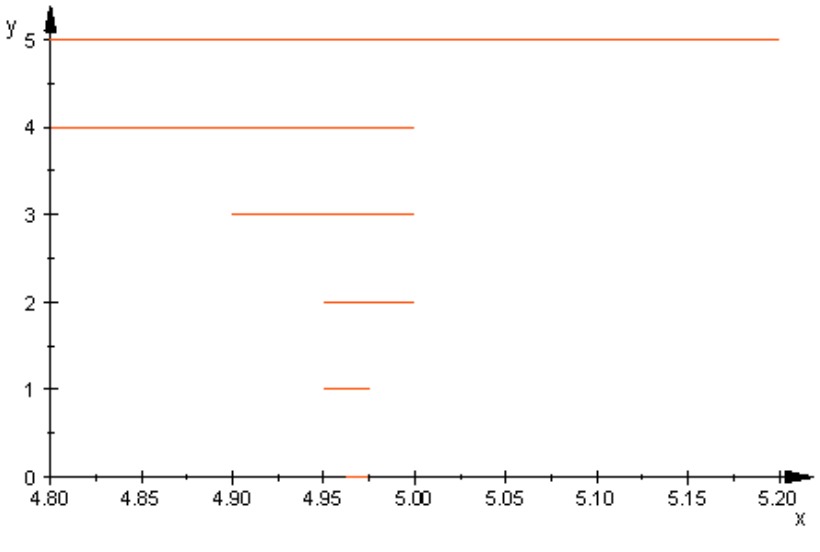


Fig. 3.9-7

Kai Gehrs e Vera Verspohl utilizam em *Analysis mit MuPAD*, SciFace Software, uma lista dos intervalos juntamente com a função **`plot::Group2d`** com a qual pode-se fazer o gráfico de um grupo de objetos gráficos.
Para construir a lista dos intervalos a traçar, é usada a instrução **`append`** (*acrescentar* elementos à uma lista). Por meio da função **`op`** puxamos os elementos da lista **`intervalos`** .

```
intervalos :=[]://lista vazia para começar
for j from 0 to N do
intervalos:=append(intervalos,plot::Line2d([a[j],j],[b[j],j])):

 //os intervalos a traçar são acrescentados à lista "intervalos"
end_for:
inter:=plot::Group2d(op(intervalos)):

//grupo dos elementos da lista "intervalos"

plot(inter):
```

A adaptação ao nosso programa pode-se fazer como segue:

```
reset():
f:= x-> exp(-x)+x/5-1:
a[0]:= 4: b[0]:=6:
p:=plot::Function(5*f(x),x = 4..6, Color=RGB::Green):
//f foi amplificado para mais claridade no gráfico
DIGITS:=25:

eps:=0.001:
N:=10:

if sign(f(a[0]))= sign(f(b[0]))then
print(Unquoted,"O intervalo não serve!")
else

for i from 1 to N do
x[i]:=(a[i-1]+b[i-1])/2:
if sign(f(a[i-1]))<> sign(f(x[i]))
then b[i]:=x[i];a[i]:=a[i-1]:

else

a[i]:=x[i]:b[i]:=b[i-1]:

end_if:
end_for:


//vem aqui o novo código:

intervalos :=[]://lista vazia
for j from 0 to N do
intervalos:=append(intervalos,
plot::Line2d([a[j],j],[b[j],j])):

//os intervalos a traçar são acrescentados à lista
"intervalos"

end_for:
inter:=plot::Group2d(op(intervalos)):

//grupo dos elementos da lista "intervalos"

plot(p,inter):
// traça a funçaõ 5*f juntamente com os intervalos
end_if:
```

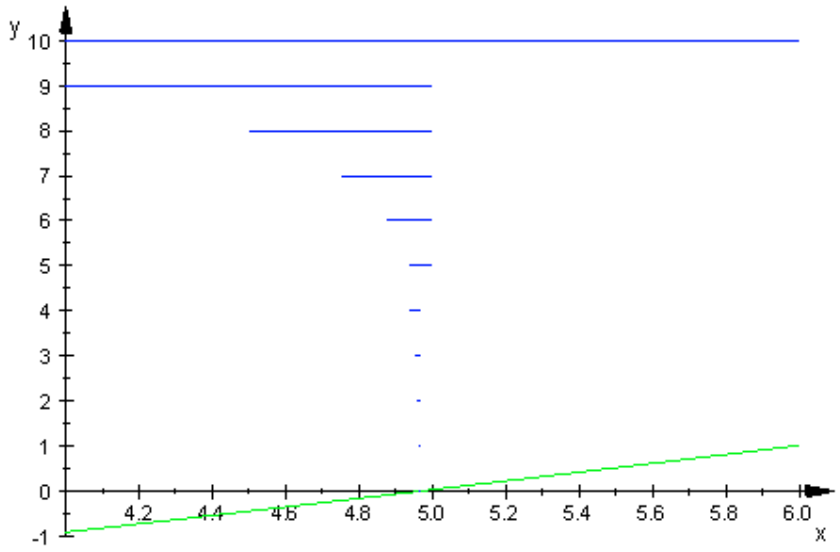
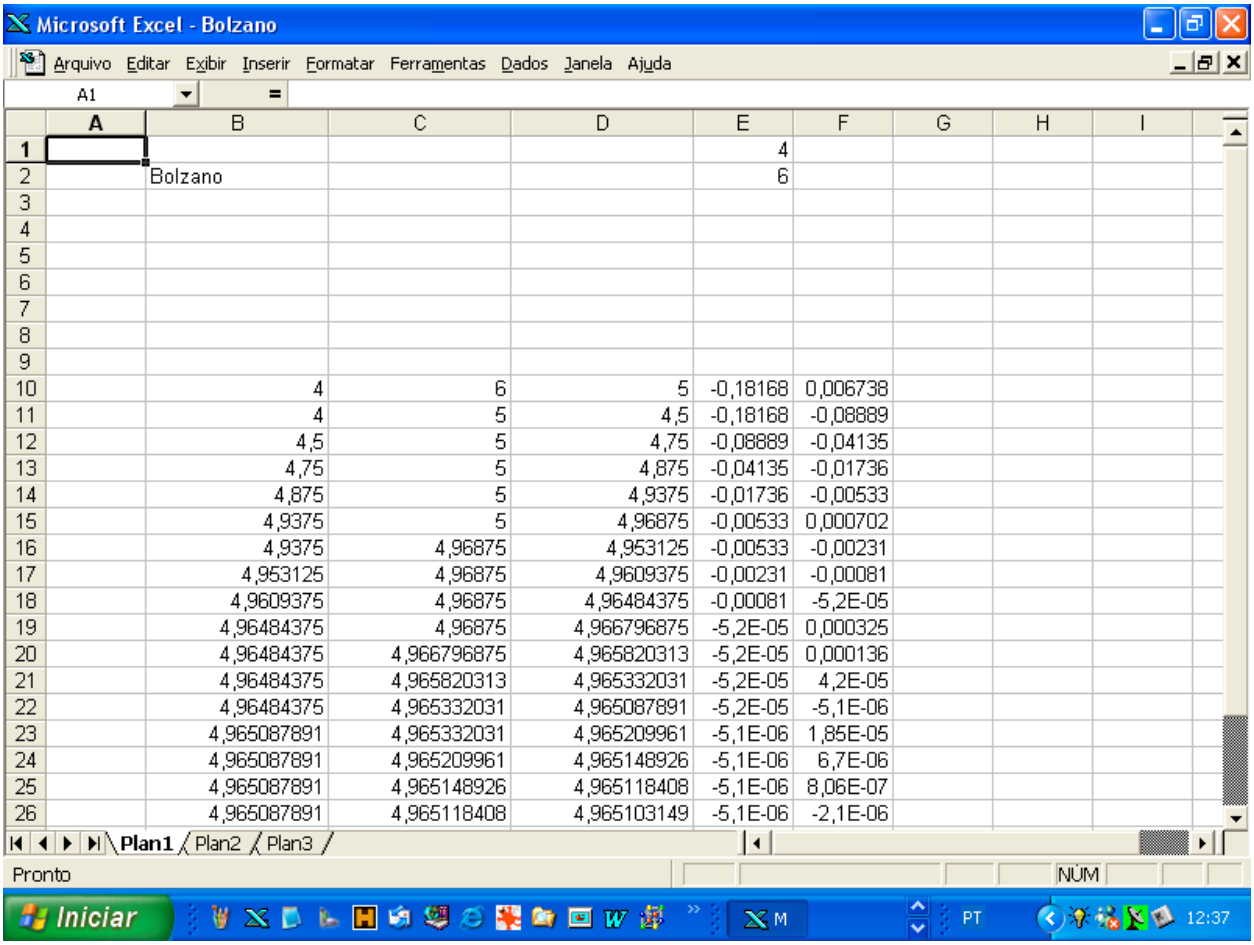


Fig. 3.9-7

Alias, a implementação do algoritmo a bissecção no **Excel** é bem simples. Utilizando a fórmulação com **f(a)·f(x) < 0** e **f(a)·f(x) > 0**, podemos proceder da seguinte maneira:

1. Ampliar as colunas B, C, D até 16 dígitos
2. Colocar o valor de a em E1 e o de b em E2.
3. B10 :**+E$1**; C10: **+E$2**; D10: **(B10+C10)/2**
   E10: **@EXP(-B10)+B10/5-1** (= f(a))
   F10: **@EXP(-D10)+D10/5-1** (= f(x))
4. Se quisermos fazer 30 iterações, temos que copiar
   D10, E10, F10 até D30, E30, F30
5. B11: **@Se(E10*F10>0;D10;B10** (a:=x)
   C11: **@Se(E10*F10<0;D10;C10** (b:=x)
6. Copiar B11, C11 até B30..C30

Resultados:



	A	B	C	D	E	F	G	H	I
1					4				
2		Bolzano			6				
3									
4									
5									
6									
7									
8									
9									
10		4	6	5	-0,18168	0,006738			
11		4	5	4,5	-0,18168	-0,08889			
12		4,5	5	4,75	-0,08889	-0,04135			
13		4,75	5	4,875	-0,04135	-0,01736			
14		4,875	5	4,9375	-0,01736	-0,00533			
15		4,9375	5	4,96875	-0,00533	0,000702			
16		4,9375	4,96875	4,953125	-0,00533	-0,00231			
17		4,953125	4,96875	4,9609375	-0,00231	-0,00081			
18		4,9609375	4,96875	4,96484375	-0,00081	-5,2E-05			
19		4,96484375	4,96875	4,966796875	-5,2E-05	0,000325			
20		4,96484375	4,966796875	4,965820313	-5,2E-05	0,000136			
21		4,96484375	4,965820313	4,965332031	-5,2E-05	4,2E-05			
22		4,96484375	4,965332031	4,965087891	-5,2E-05	-5,1E-06			
23		4,965087891	4,965332031	4,965209961	-5,1E-06	1,85E-05			
24		4,965087891	4,965209961	4,965148926	-5,1E-06	6,7E-06			
25		4,965087891	4,965148926	4,965118408	-5,1E-06	8,06E-07			
26		4,965087891	4,965118408	4,965103149	-5,1E-06	-2,1E-06			