

## 3.1 Equações diferenciais de ordem superior de um e sistemas de equações

As equações diferenciais de segunda ordem são, não só na mecânica, de importância fundamental. Varias vezes vimos que os simples métodos de Euler dão resultados satisfatórios, pelo menos para nossos fines pedagógicos. Estes métodos deixam-se facilmente expandir a equações diferenciais de ordem superior, especialmente para as aplicações da segunda lei de Newton, que invariavelmente pedem a solução de equações de segunda ordem. Também podemos generalizar estes métodos para resolver *sistemas* de equações diferenciais de segunda ordem, como vimos, por exemplo, no cálculo da trajetória do Planeta Mercúrio na última seção.

É somente necessário basear-se nas definições da velocidade  $v$  e da aceleração  $a$  para desenvolver um algoritmo simples e bastante útil. No programa seguinte são incorporadas as definições

$$a(t) \approx (v(t+h) - v(t))/h \quad \text{e} \quad v(t) \approx (y(t+h) - y(t))/h, \quad \text{onde } h = \Delta t. \quad (1)$$

Se  $a$  é conhecida em qualquer instante  $t$ , a velocidade e a posição em  $(t + h)$  podem ser calculadas pelas equações

$$v(t+h) \approx v(t) + h a(t) \quad \text{e} \quad (2)$$

$$y(t+h) \approx y(t) + h v(t) \quad (3)$$

A aceleração  $a$  pode depender explicitamente da posição, da velocidade e do tempo.  $v(t+h)$  é a velocidade no final do intervalo  $dt = h$ . Apesar de a aceleração não ser constante no período  $h$ , supomos que ela não se modifique muito em um intervalo de tempo muito pequeno, e em geral podemos esperar que quanto menor é o intervalo de tempo  $h$ , tanto melhor aproximam as expressões (1) as derivadas exatas  $a(t)$  e  $v(t)$ .

Usando a notação matemática, escrevemos para (2) e (3) as formas.

$$y'(t+h) \approx y'(t) + h y''(t) \quad \text{e} \quad (4)$$

$$y(t+h) \approx y(t) + h y'(t) \quad (5)$$

No programa a seguir resolveremos a equação  $d^2y/dt^2 + 4y^2 = 0$  com as condições iniciais  $y'(0) = v(0) = 0$  e  $y(0) = 2$ . Primeiro calculamos  $a(0) = y''(0)$  e pois o novo valor de  $y$ :  $y(h) = y(0) + h v(0)$ . Para o novo valor da velocidade precisamos do valor de  $a(0)$  que acabamos de determinar.  $v(h) = y'(h) = y'(0) + h a(0)$ . A solução exata do problema é  $y = 2/(4t^2 + 1)$ . Para o cálculo de  $y(t+h)$  usa-se a velocidade no começo do intervalo  $h$ . Por isso, o método é chamado de first-point-method (FPM). No LPM (last-point-method) calcula-se  $x(t+h)$  usando  $v(t+h)$ . (No caso dos processos periódicos, é geralmente melhor determinar a velocidade *antes* da posição, ou seja aplica-se o LPM. Resulta assim que as trajetórias, que deveriam ser fechadas, se fecham efetivamente. Veja mais adiante o Exemplo 2 no parágrafo 3.1.2. Fala-se do método de Euler-Cromer, veja. Am.J.Phys. 52,499, 1984.)

O esquema do cálculo do método de Euler (FPM) será:

Passo	Tempo	Posição	Velocidade	Aceleração
0	$t_0$	$y_0$	$v_0$	$a_0 = f(t_0, y_0, v_0)$
1	$t_1 = t_0 + h$	$y_1 = y_0 + v_0 h$	$v_1 = v_0 + a_0 h$	$a_1 = f(t_1, y_1, v_1)$
2	$t_2 = t_1 + h$	$y_2 = y_1 + v_1 h$	$v_2 = v_1 + a_1 h$	$a_2 = f(t_2, y_2, v_2)$
...				
n	$t_n = t_{n-1} + h$	$y_n = y_{n-1} + v_{n-1} h$	$v_n = v_{n-1} + a_{n-1} h$	$a_n = f(t_n, y_n, v_n)$

Programa 1:

- ```

reset() :
Euler:=proc(h,passos) //Euler simples
begin
t:=0:i:=0:
y:=2:v:=0:
DIGITS:=6:
f:=(t,y)->-4*y(t)^2:

```

```

for i from 0 to passos do
Y(i):=y:T(i):=t:
if (i=0)or (modp(i,2)=0)then //se mostra cada segundo valor
print("i=",i,"t= ",t,"y= ",y,"v= ",v,"a ",f(t,y)):
end_if:

a:=f(t,y):
y:=y+v*h:
v:=v+a*h:
t:=t+h:
end_for:
plot(plot::Point2d(T(i),Y(i))
$ i=1..passos,Color=RGB::Blue):
end_proc:
Euler(0.05,10)

```

```

"i=", 0, "t= ", 0, "y= ", 2, "v= ", 0, "a ", -16
"i=", 2, "t= ", 0.1, "y= ", 1.96, "v= ", -1.6, "a ", -15.3664
"i=", 4, "t= ", 0.2, "y= ", 1.76158, "v= ", -3.0752, "a ", -
12.4127
"i=", 6, "t= ", 0.3, "y= ", 1.42303, "v= ", -4.21286, "a ", -
8.10008
"i=", 8, "t= ", 0.4, "y= ", 0.981496, "v= ", -4.91184, "a ", -
3.85334
"i=", 10, "t= ", 0.5, "y= ", 0.480679, "v= ", -5.21282, "a ", -
0.924211

```

Também os métodos melhorados de Euler se deixam estender para equações superiores. No programa a seguir combinamos o método de Euler com o de Heun, ou seja, tomamos Euler para calcular a velocidade usando  $v(t+h) = v(t) + h a(t)$  e Heun para determinar a posição:  $y(t+h) = y(t) + (v(t) + v(t+h)) \cdot h/2$ . Neste Programa incluímos uma tabela: `tabela:=array(1..passos,1..4):` e a fórmula da solução analítica: `F0:=2/(4*t^2+1).`

## Programa 2:

```

• reset()://Método Euler_Heun
  Euler_Heun:=proc(h,passos)
    local i,t,y,v,f,F0;
    begin
      t:=0:y:=2:v:=0:
      DIGITS:=8:
      f:=(t,y)->-4*y^2:
      tabela:=array(1..passos,1..4):
      for i from 1 to passos do
        va:=v:// va = velocidade anterior, v = vel. atual
        a:=f(t,y):
        v:=va+h*a:
        y:=y+(h/2)*(va+v):
        t:=t+h:
        F0:=2/(4*t^2+1)//solução analítca

        tabela[i,1]:=t:
        tabela[i,2]:=v:
        tabela[i,3]:=y:
        tabela[i,4]:=F0:
      end_for:
      return(tabela)
    end_proc:
  Euler_Heun(0.05,10)

```

**Solução:**

| <b>t</b> | <b>v</b>    | <b>y</b>   | <b>y<sub>exato</sub></b> |
|----------|-------------|------------|--------------------------|
| 0.05,    | -0.8,       | 1.98,      | 1.980198                 |
| 0.1,     | -1.58408,   | 1.920398,  | 1.9230769                |
| 0.15,    | -2.3216657, | 1.8227544, | 1.8348624                |
| 0.2,     | -2.9861524, | 1.6900589, | 1.7241379                |
| 0.25,    | -3.5574122, | 1.5264698, | 1.6                      |
| 0.3,     | -4.0234342, | 1.3369486, | 1.4705882                |

Observe como os valores numéricos começam a desviar-se fortemente dos valores exatas a partir de  $t = 0.2$

### Programa 3: (Feynman)

R.P.Feynman utiliza um método simples, quase como Euler, mas melhor, para calcular a órbita do planeta Mercúrio (veja The Feynman Lectures of Physics, Vol. I, Sec.9-7). Feynman utiliza a velocidade no centro do intervalo  $h$  e parte da equação  $y(t+h) \approx y(t) + h v(t+h/2)$ . Segundo Feynman, calcula-se um valor aproximado para  $v(t+h/2)$  por meio de

$$a(t) = y''(t) \approx (y'(t+h/2) - y'(t-h/2))/h$$

Obtemos assim  $v(t+h/2) \approx v(t - h/2) + h a(t)$ . Mas, quando  $t = 0$ , estamos em frente de um problema, pois não conhecemos  $v(-h/2)$ . Bem, para este caso de  $t = 0$  aproximamos a aceleração com a fórmula  $a(0) \approx (v(h/2) - v(0))/ (h/2)$  o que nos dá para  $v(h/2)$  a seguinte equação de arranque

$$v(h/2) = v(0) + a(0) h/2$$

que usamos uma vez só. A posição calculamos sempre depois de passos inteiros, mas a velocidade será sempre determinada no centro de um passo.

Como exemplo utilizamos outra vez a equação  $y''(t) = f(t,y,y') = - 4[y(t)]^2$  com  $y(0) = 2$  e  $y'(0) = v(0) = 0$ .

- ```

Feynman:=proc(h,passos)// para y''=f(t,y,v)
local i,t0,t,y0,y,v0,v,f;
begin

i:=0:
t0:=0:y0:=2:v0:=0:
t:=t0:y:=y0:v:=v0:
DIGITS:=6:

f:=-4*y^2:

print("t=",t,"y=",y,"y'=",v,"a=",f):
//passo médio inicial (intial halfstep)para v
v:=v + f*h/2:
//y é calculado com h
y:=y+v*h:

```

```

for i from 1 to passos do
print("t=",t+h/2,"v=",v):
t:=t+h:
f:=-4*y^2:
print("t=",t,"y=",y,"a=",f):
v:=v+f*h:
y:=y+v*h:
end_for:
end_proc:

```

**Feynman(0.05,6)**

```

"t=", 0, "y=", 2, "y'=", 0, "a=", -16
"t=", 0.025, "v=", -0.4
"t=", 0.05, "y=", 1.98, "a=", -15.6816
"t=", 0.075, "v=", -1.18408
"t=", 0.1, "y=", 1.9208, "a=", -14.7578
"t=", 0.125, "v=", -1.92197
"t=", 0.15, "y=", 1.8247, "a=", -13.3181
"t=", 0.175, "v=", -2.58788
"t=", 0.2, "y=", 1.6953, "a=", -11.4962

```

#### Programa 4: (Runge-Kutta)

As fórmulas de **Runge-Kutta** podem ser aplicadas às equações diferenciais de segunda ordem, bem como àquelas de ordem superior.

Para isso, recorreremos ao algoritmo apresentado em 2.1, onde somente temos que acrescentar as relações que se referem à aceleração (= derivada segunda).

$$t_{n+1} = t_n + h$$

$$y_{n+1} = y_n + h\langle v \rangle$$

$$v_{n+1} = v_n + h\langle a \rangle$$

onde

$$\langle v \rangle := (v_1 + 2v_2 + 2v_3 + v_4) / 6$$

$$\langle a \rangle := (a_1 + 2a_2 + 2a_3 + a_4) / 6$$

As derivadas calculam-se usando o seguinte esquema:

$v_1 := v$	$a_1 := f(t, y, v)$
$v_2 := v + a_1 h / 2$	$a_2 := f(t + h/2, y + v_1 h / 2, v_2)$
$v_3 := v + a_2 h / 2$	$a_3 := f(t + h/2, y + v_2 h / 2, v_3)$
$v_4 := v + a_3 h$	$a_4 := f(t + h, y + v_3 h, v_4)$

Uma implementação desse esquema podemos realizar facilmente:

- ```

reset() //Runge-Kutta y''(t,y,y'); a:=y'', v:=y'
t:=0:y:=-1:v:=0:
h:=0.1:

DIGITS:=7:

f:=(t,y,v)->3*v-2*y:
for i from 1 to 10 do
v1:=v:
a1:=f(t,y,v):
v2:=v+a1*h/2:
a2:=f(t+h/2,y+v1*h/2,v2):
v3:=v+a2*h/2:
a3:=f(t+h/2,y+v2*h/2,v3):
v4:=v+h*a3:
a4:=f(t+h,y+v3*h,v4):
y:=y+h*(v1+2*v2+2*v3+v4)/6:
v:=v+h*(a1+2*a2+2*a3+a4)/6:
t:=t+h:

F0:=exp(2*t)-2*exp(t) //solução analítica
print(t,v,y,F0,F0-y):
end_for:

```

**Solução:**

0.1, 0.2324583, -0.9889417, -0.9889391, 0.000002589542  
 0.2, 0.5408308, -0.9509872, -0.9509808, 0.000006363494  
 0.3, 0.9444959, -0.8776105, -0.8775988, 0.00001172395  
 0.4, 1.467393, -0.7581277, -0.7581085, 0.00001918874  
 0.5, 2.139061, -0.5791901, -0.5791607, 0.00002942863

**3.1.2 Redução de  $y''$  a duas eqs. de primeira ordem**

Nas seções anteriores aprendemos métodos para resolver equações diferenciais da primeira ordem. Também podemos estender estes métodos sobre *sistemas* de primeira ordem. Resulta que estes métodos podem ser usados também para equações de ordem superior a um, tal como  $d^2x/dt^2 = f(t,x(t),x'(t))$ , pois esta equação pode, por exemplo, ser reduzida a um sistema de duas equações da primeira ordem  $y' = f(t,x,y)$  e  $x' = y$ . Por exemplo:

A equação para um pêndulo  $x'' + \text{sen}(x) = 0$ ,  $x(0) = 0$ ,  $x'(0) = 1$  pode ser transformada em  $x' = y$  e  $y' = x'' = -\text{sen}(x)$ .

Bastará então considerar unicamente sistemas de equações de primeira ordem. Mas, as vezes será mais útil e simples resolver a equação superior por um procedimento direto, ou seja, sem desdobrar a equação em duas equações da primeira ordem. A extensão de nossos métodos numéricos a equações de ordem superior a um é "straightforward", como veremos no seguinte parágrafo.

Usaremos  $t$  para designar a variável independente e  $x, y, z, \dots$  para as variáveis dependentes. Na Física,  $t$  significa normalmente o tempo. Mas, um *verdadeiro* sistema de duas equações acopladas e de primeira ordem é raro na Física. No entanto, na ecologia existem muitos modelos matemáticos que se baseiam em sistemas da primeira ordem. Muito famoso é o sistema de Volterra-Lotka, que vamos estudar mais tarde.

Suponhamos que temos o seguinte sistema

$$x'(t) = f(t,x,y) \quad (1a)$$

$$y'(t) = g(t,x,y) \quad (1b)$$

com as condições iniciais  $x(t_0) = x_0$  e  $y(t_0) = y_0$ .

Queremos determinar valores aproximados  $x_1, x_2, x_3 \dots y_1, y_2, y_3 \dots$  para as soluções exatas  $x(t)$  e  $y(t)$  nos pontos  $t_i = t_0 + i h$ .

Agora é só necessário aplicar, por exemplo, a fórmula de Euler  $y_{i+1} = y_i + h \cdot f(x_i, y_i)$  ao nosso sistema (1):

$$x_{i+1} = x_i + h f(t_i, x_i, y_i) = x_i + h x'_i \quad (2a)$$

$$y_{i+1} = y_i + h g(t_i, x_i, y_i) = y_i + h y'_i \quad (2b)$$

### Exemplo 1:

Determinar valores aproximados de  $x(t)$  e  $y(t)$  para o sistema

$$x'(t) = f(t,x,y) = x - 4y$$

$$y'(t) = g(t,x,y) = -x + y \quad \text{com } x(0) = 1 \text{ e } y(0) = 0.$$

**Solução:** ( $h = 0.1$ )

$$t=0$$

$$f = 1 - 4 \cdot 0 = 1$$

$$g = -1 + 0 = -1$$

$$x_1 = 1 + 0.1 \cdot 1 = 1.1$$

$$y_1 = 0 + 0.1 \cdot (-1) = -0.1$$

$$t=0.1$$

$$f = 1.1 - 4 \cdot (-0.1) = 1.1 + 0.4 = 1.5$$

$$g = -1.1 + (-0.1) = -1.2$$

$$x_2 = 1.1 + 0.1 \cdot 1.5 = 1.25$$

$$y_2 = -0.1 + 0.1 \cdot (-1.2) = -0.22, \text{ compare agora o programa:}$$

- `reset()://sistema de  $x' := f(t, x, y)$ ,  $y' := g(t, x, y)$ , EULER`  
`h:=0.1:`  
`t:=0:`  
`x:=1:y:=0:`  
`DIGITS:=6:`  
`for i from 0 to 5 do`  
`f:=x-4*y:`  
`g:=-x+y:`  
`x:=x+h*f:`  
`y:=y+h*g:`  
`t:=t+h:`  
`print(t,x,y):`  
`end_for:`

| <b>t</b> | <b>x</b> | <b>y</b> |
|----------|----------|----------|
| 0.1,     | 1.1,     | -0.1     |
| 0.2,     | 1.25,    | -0.22    |
| 0.3,     | 1.463,   | -0.367   |
| 0.4,     | 1.7561,  | -0.55    |
| 0.5,     | 2.15171, | -0.78061 |
| 0.6,     | 2.67912, | -1.07384 |

### Exemplo 2: (Raposas e coelhos)

O matemático italiano V. Volterra (1860-1940) e, independente de ele, A.J.Lotka (1925) desenvolveram o seguinte sistema para calcular a evolução do número de coelhos e raposas num determinado ecossistema, ano após ano:

$$x' = ax - bxy; \quad y' = -cy + dxy$$

$x'(t)$  é a taxa de crescimento dos coelhos e  $y'(t)$  a taxa para as raposas. Escreva um programa que simule a evolução dos animais (modelo do presa-predador). Os constantes serão  $a = 2$ ,  $b = 0.01$ ,  $c = 1$ ,  $d = b$ ;  $h = 0.02$ . Os números iniciais de coelhos e raposas será 100 e 60.

### Solução:

- ```

reset() //coelhos e raposas
x:=100:y:=60:h:=0.02:t:=0:
a:=2:b:=0.01:
c:=1:d:=0.01:
DIGITS:=6:
for i from 0 to 5 do
f:=a*x-b*x*y:
g:=-c*y+d*x*y:
x:=x+h*f:
y:=y+h*g:
t:=t+h:
print(t,x,y) //tempo, coelhos, raposas
end_for:

```

### Solução:

```

0.02, 102.8, 60.0
0.04, 105.678, 60.0336
0.06, 108.637, 60.1018
0.08, 111.676, 60.2056
0.1, 114.799, 60.3462
0.12, 118.005, 60.5248

```

- ```

reset()://coelhos e raposas -com gráfico
x:=100:y:=60:h:=0.05:t:=0:
a:=2:b:=0.01:
c:=1:d:=0.01:
DIGITS:=6:
for i from 0 to 100 do
X(i):=x:Y(i):=y:T(i):=t:
f:=a*x-b*x*y:
x:=x+h*f://assim se fecha a trajetória
g:=-c*y+d*x*y:
y:=y+h*g:
t:=t+h:
//print(t,x,y):
end_for:

plot(plot::Point2d(X(i),Y(i),Color=RGB::Red) $
i=0..100,ViewingBox=[0..400,0..600],Scaling=Constrained):

```

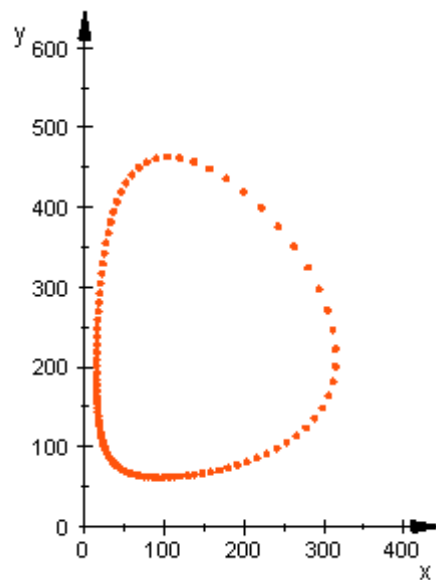


Fig.: 3.1-1

Nesta versão do programa anterior interpolamos a equação para  $x$  entre as equações para  $f$  e  $g$ . Desta forma, foi usado no cálculo de  $g$  sempre o valor de  $x$  atualizado. O resultado é uma trajetória fechada. Este fenómeno pode-se observar com muitos sistemas periódicos quando são tratados com o método de Euler. O centro da trajetória tem as coordenadas  $(c/d; a/b)$ . O período pode ser calculado com

$$T = 2\pi/(ad)^{1/2}$$

### Exemplo 3:

Transforme a equação de Van der Pol  $y''(x) - m(1-y^2)y'(x) + y = 0$  num sistema de duas equações diferenciais de primeira ordem.  $m$  é um parâmetro maior do que zero.

### Solução:

Para fazer a transformação, vamos usar outra vez uma mudança de variáveis:

$$y_1(x) = y(x) \quad \text{e} \quad y_2(x) = y'(x).$$

Teremos então o sistema

$$\begin{aligned} y_1' &= y_2 \\ y_2' &= m(1-y_1^2) y_2 - y_1. \end{aligned}$$

Para resolver o problema, usando um dos nossos métodos, podemos tomar  $m=2$  e as condições iniciais  $y(0) = 2$  e  $y'(0) = 0$ .

A equação de Van der Pol é uma equação homogênea, pois o lado direito é zero. No próximo exemplo consideramos uma equação linear de segunda ordem com uma função  $F(t)$  "de excitação" no lado direito. No capítulo 6 que trata de oscilações encontramos esta equação quando falamos sobre oscilações excitadas, onde  $F(t)$  tem a forma  $F(t) = F_0 \cos(\omega t)$ .

**Exemplo 4:**

Transforme a equação do oscilador harmônico

$$mx''(t) + rx'(t) + kx(t) = F(t)$$

num sistema composto por duas equações diferenciais de primeira ordem. As condições iniciais são  $x(t_0) = x_0$  e  $x'(t_0) = u_0$ .

**Solução:**

Outra vez fazemos a mudança  $x'(t) = y(t)$  e  $x''(t) = y'(t)$  com que a equação de segunda ordem torna-se:

$$x'(t) = y \quad \text{e} \quad y'(t) = f(t,x,y)$$

com as condições iniciais  $x(t_0) = x_0$  e  $y(t_0) = y_0$ . **Por exemplo:**

$$x''(t) - 2x'(t) + 2x(t) = e^{2t} \text{sen}(t)$$

Para esta equação utilizaremos o método de **Runge-Kutta** com

$x'(t) = y(t)$  ( $=f(t,x,y)$ );  $y$  representa a derivada  $dx/dt$  (= velocidade)

$y'(t) = 2y(t) - 2x(t) + e^{2t}\text{sen}(t)$  ( $=g(t,x,y)$ );  $y'$  é a derivada segunda  $x''$  (=aceleração)

$t_0=0$ ;  $x_0 = -0.4$ ;  $y_0 = -0.6$  ( $=x'(0) = v(0)$ )

A solução analítica é  $x(t) = 0.2 e^{2t} (\text{sent} - 2 \text{cost})$

O algoritmo de **RK** para uma equação  $y' = f(x,y)$ , veja 2.1, é facilmente modificado para resolver um sistema de duas equações de primeira ordem, é só acrescentar uma segunda função  $g(t,x,y)$ , ou seja agora temos  $x' = f(t,x,y)$  e  $y' = g(t,x,y)$ .

- `reset()`://Runge-Kutta sistema  $x'=f(t,x,y);y'=g(t,x,y)$

```

t:=0:x:=-0.4:y:=-0.6:
h:=0.1:
DIGITS:=5:

f:=(t,x,y)->y:
g:=(t,x,y)->2*y-2*x+exp(2*t)*sin(t):

for i from 1 to 5 do

v1:=f(t,x,y)// velocidade
a1:=g(t,x,y)// aceleração
v2:=f(t+h/2,x+v1*h/2,y+a1*h/2):
a2:=g(t+h/2,x+v1*h/2,y+a1*h/2):
v3:=f(t+h/2,x+v2*h/2,y+a2*h/2):
a3:=g(t+h/2,x+v2*h/2,y+a2*h/2):
v4:=f(t+h,x+v3*h,y+a3*h):
a4:=g(t+h,x+v3*h,y+a3*h):

x:=x+h*(v1+2*v2+2*v3+v4)/6:
y:=y+h*(a1+2*a2+2*a3+a4)/6:

t:=t+h:

F0:=0.2*exp(2*t)*(sin(t)-2*cos(t))://sol. analítica

print(t,F0,x,y)// y é a derivada e x

end_for:

```

| t    | val. exato | x         | y=dx/dt  |
|------|------------|-----------|----------|
| 0.1, | -0.46173,  | -0.46173, | -0.63163 |
| 0.2, | -0.52556,  | -0.52556, | -0.64015 |
| 0.3, | -0.5886,   | -0.5886,  | -0.61366 |
| 0.4, | -0.64661,  | -0.64661, | -0.53658 |
| 0.5, | -0.69356,  | -0.69357, | -0.38874 |

O método de MuPAD aplicamos já na seção 3.8, Programa 3. Aqui podemos compará-lo com o programa de Runge-Kutta. O MuPAD possui vários operadores para a resolução de EDO's, aqui vemos dois para a resolução da nossa equação

$$x'' - 2x' + 2x = e^{2t} \sin t:$$

- `reset() //MuPAD para sistema x'=f(t,x,y);y'=f(t,x,y)`  
`x0:=-0.4;y0:=-0.6:`

```
IVP:={x'(t)=y(t),y'(t)=-
2*x(t)+2*y(t)+exp(2*t)*sin(t),x(0)=x0,y(0)=y0}:
fields:=[x(t),y(t)]:
ivp:=numeric::ode2vectorfield(IVP,fields):
```

```
Y:=numeric::odesolve2(ivp):
```

```
Y(0.5); //x e y =x' depois de 0.5 s
```

```
[-0.6935639464, -0.3887390548]
```

**Outra versão:**

- `reset() :`  
`f:=(t,Y)->[Y[2],2*Y[2]-2*Y[1]+exp(2*t)*sin(t)]:`  
`t0:=0:Y0:=[-0.4,-0.6]:`  
`Y:=numeric::odesolve2(f,t0,Y0):`

```
Y(0.5)
```

```
[-0.6935639464, -0.3887390548]
```

### 3.1.3 Sistemas de duas equações de segunda ordem

#### Euler, Feynman, Runge-Kutta

Os sistemas de equações diferenciais de segunda ordem surgem em muitos problemas da Física, por exemplo, as equações de movimento são em geral equações diferenciais de segunda ordem, até mesmo, sistemas de tais equações. As equações de movimento de um planeta são  $x'' = -x/r^3$  e  $y'' = -y/r^3$ , onde  $r^2 = x^2 + y^2$ . Sua forma geral é  $x'' = f(t,x,x',y,y')$  e  $y'' = g(t,x,x',y,y')$  com as condições iniciais  $x(t_0) = x_0$ ;  $x'(t_0) = x'_0$ ;  $y(t_0) = y_0$ ; e  $y'(t_0) = y'_0$ .

Os algoritmos para uma equação só devemos, agora, aplicar a duas equações. Não será preciso escrever todos os detalhes, mas vou dar-lhes aqui os programas para os algoritmos de Euler, Feynman e Runge-Kutta. Na vida real utilizamos, obviamente, MuPAD com seus excelentes métodos embutidos.

Euler:

```

• reset() :

Euler:=proc(h,passos)//movimento do Mercúrio

begin
t:=0:i:=0:
x:=0.30779:y:=0:u:=0:v:=1.9772:
//x(0):=0.5:y(0):=0:u(0):=0:v(0):=1.63 segundo Feynman

DIGITS:=6:

r:=(x,y)->sqrt(x^2+y^2):
F:=(x,y)->-x/r(x,y)^3:
G:=(x,y)->-y/r(x,y)^3:

```

```

for i from 0 to passos do
X(i):=x:Y(i):=y:T(i):=t:
if (i=0)or (modp(i,10)=0)then

print("i=",i,"t= ",t,"x= ",x,"y= ",y):

end_if:

u:=u+F(x,y)*h://last-point-method (Cromer)
v:=v+G(x,y)*h:
x:=x+u*h:
y:=y+v*h:
t:=t+h:
end_for:

plot(plot::Point2d(X(i),Y(i))
$ i=1..passos,Color=RGB::Blue):

end_proc:

Euler(0.01,151)

```

**Feynman:** (método do passo meio, halfstep)

- `Feynman:=proc(h,passos)//Mercúrio`  
`begin`  
`DIGITS:=6:`  
`t0:=0:i:=0:`  
`x0:=0.5:y0:=0:u0:=0:v0:=1.63://dados do Feynman`  
`t:=t0:x:=x0:y:=y0:u:=u0:v:=v0:`  
`//intial halfstep`  
`r:=sqrt(x^2+y^2):`  
`F:=-x/r^3://9.81*(0.8728-(u/2.028)^2):`  
`G:=-y/r^3:`  
`print("t=",t,"x=", x,"y=",y):`  
`t:=h/2:`  
`u:=u + F*h/2:`  
`v:=v + G*h/2:`  
`print("t=",t,"u=", u,"v=",v):`  
`t:=h:`

```

x:=x+u*h:
y:=y+v*h:
print("t=",t,"x=", x,"y=",y):
for i from 1 to passos do
r:=sqrt(x^2+y^2):
F:=-x/r^3:
G:=-y/r^3:
t:=t+h/2:
u:=u + F*h:
v:=v + G*h:
print("t=",t,"u=",u,"v=",v);
t:=t+h/2:
x:=x+u*h:
y:=y+v*h:
print("t=",t,"x=",x,"y=",y);
end_for:

end_proc:

Feynman(0.1,6)

"t=", 0, "x=", 0.5, "y=", 0
"t=", 0.05, "u=", -0.2, "v=", 1.63
"t=", 0.1, "x=", 0.48, "y=", 0.163
"t=", 0.15, "u=", -0.568485, "v=", 1.50487
"t=", 0.2, "x=", 0.423151, "y=", 0.313487

```

- `reset()`://RK sistema de 2 eq. dif. de segunda ordem

```

t0:=0:
x0:=1:
y0:=0:
u0:=0:
v0:=0:
h:=0.2:
DIGITS:=5:

t:=t0:x:=x0:u:=u0:y:=y0:v:=v0:

```

```

f:=(x,y,u,v)->-x+v:
g:=(x,y,u,v)->-y+u:

for i from 1 to 5 do

f1:=f(x,y,u,v):
g1:=g(x,y,u,v):

t:=t0+h/2:
x:=x0+u*h/2:y:=y0+v*h/2:
u:=u0+f1*h/2:v:=v0+g1*h/2:
f2:=f(x,y,u,v):g2:=g(x,y,u,v):

x:=x0+u*h/2:y:=y0+v*h/2:
u:=u0+f2*h/2:v:=v0+g2*h/2:
f3:=f(x,y,u,v):g3:=g(x,y,u,v):

t:=t0+h:
x:=x0+u*h:y:=y0+v*h:
u:=u0+f3*h:v:=v0+g3*h:
f4:=f(x,y,u,v):g4:=g(x,y,u,v):

x:=x0+h*u0+h*h*(f1+f2+f3)/6:
y:=y0+h*v0+h*h*(g1+g2+g3)/6:
u:=u0+h*(f1+2*f2+2*f3+f4)/6:
v:=v0+h*(g1+2*g2+2*g3+g4)/6:

print(t,x,y,u,v):

t0:=t:x0:=x:y0:=y:u0:=u:v0:=v:

end_for:

```

```

0.2, 0.98, -0.0013333, -0.2, -0.019933
0.4, 0.92, -0.010587, -0.39992, -0.078933
0.6, 0.82006, -0.03536, -0.59936, -0.1746
0.8, 0.68035, -0.082614, -0.79732, -0.30294
1.0, 0.50135, -0.15835, -0.99187, -0.45836

```

Todos os métodos até agora discutidos são chamados de "single-step" (passo único). Isso quer dizer: Quando se conhece a solução  $x(t)$  para um instante  $t$  determinado, se pode, então, calcular  $x(t+h)$ , sem necessidade de conhecer também valores da solução para instantes anteriores a  $t$ .

Mas, nos chamados métodos "multi-step" (passo múltiplo) se faz também uso de valores anteriores a  $t$ , a saber:  $x(t-h)$ ,  $x(t-2h)$ , ... Tais métodos precisam, ao começo, de um método "single-step" para calcular alguns valores iniciais para arrancar o algoritmo.

Os métodos de passo múltiplo mais populares provêm de ADAMS-BASHFORD, MILNE e de HAMMING. O do ADAMS foi desenvolvido em 1855, baseando-se em idéias do BASHFORD. Anos depois, o método caiu no olvido até, no começo do século XX, foi redescoberto pelo matemático norueguês STRÖMER.

A fórmula de recorrência de ADAMS para  $x'(t) = f(x(t))$  tem a seguinte forma:

$$x(t+h) = x(t) + h/24 \cdot (55 f(x(t)) - f(x(t-h)) + 37 f(x(t-2h)) - 9 f(x(t-3h)))$$

Antes de aplicar esta fórmula, calcula-se os valores necessários para o arranque pelo método de Runge-Kutta.

O método de HAMMING é muito exato e estavel e é por isso usado com freqüência.

