

Solução numérica de $y' = f(t, y)$

- Seja $Y(t)$ a solução exata de $Y'(t) = f(t, Y(t))$, $Y(0) = Y^0$.
- Métodos numéricos são receitas para calcular um conjunto de vetores $y^0, y^1, \dots$, que aproximam sistematicamente $Y(0), Y(t_1), Y(t_2), \dots$, para uma sequência de tempos $t_1, t_2, \dots$.
- **Exemplo mais simples:** Método de Euler com passo fixo definido pelo usuário.

$$y^0 = Y^0, \quad t_k = k \Delta t, \quad y^{k+1} = y^k + \Delta t f(t_k, y^k)$$

É um método explícito porque não requer inverter f , como é na versão implícita $y^{k+1} = y^k + \Delta t f(t_{k+1}, y^{k+1})$.

- Nos métodos explícitos, o custo computacional está na avaliação da função f . Para o método de Euler: 1 avaliação por passo.
- Esses métodos se tornam instáveis se $\Delta t > c/J_f$, com $c \simeq 2$ e J_f o maior autovalor em módulo da matriz Jacobiana de f .
- Por J_f depender de y o limite de estabilidade pode apenas ser *adivinhado*.
- Métodos Runge-Kutta constituem bons compromissos entre custo (número de avaliações de f) e precisão ($\|y^k - Y(t_k)\|$).
- A **precisão** é usualmente avaliada pela **ordem do método**:
 $\|y^k - Y(t_k)\| \leq C \Delta t^p$.

Métodos de Runge-Kutta e Tabela de Butcher

$$y^{n+1} = y^n + \Delta t (b_1 k_1 + b_2 k_2 + \dots + b_s k_s)$$

onde

$$k_1 = f(t_n, y^n)$$

$$k_2 = f(t_n + c_2 \Delta t, y^n + \Delta t (a_{21} k_1))$$

$$k_3 = f(t_n + c_3 \Delta t, y^n + \Delta t (a_{31} k_1 + a_{32} k_2))$$

...

0					
c_2	a_{21}				
c_3	a_{31}	a_{32}			
...		...			
	b_1	b_2	...	...	b_s

$$y^{n+1} = y^n + \Delta t (b_1 k_1 + b_2 k_2 + \dots + b_s k_s)$$

onde

$$k_1 = f(t_n, y^n)$$

$$k_2 = f(t_n + c_2 \Delta t, y^n + \Delta t (a_{21} k_1))$$

$$k_3 = f(t_n + c_3 \Delta t, y^n + \Delta t (a_{31} k_1 + a_{32} k_2))$$

$$\dots \quad \dots$$

```
K(:,1)=f(y(:,n),time(n)); ## ydot=f(y,t), notar inversão de ordem
for m=2:nstage
    tt=time(n)+c(m)*dt;
    yy=y(:,n)+dt*K(:,1:m-1)*a(m,1:m-1)';
    K(:,m)=f(yy,tt);
endfor
ynew=y(:,n)+dt*K(:,1:nstage)*b';
time(n+1)=time(n)+dt;
dtime(n+1)=dt;
y(:,n+1)=ynew;
```

Runge-Kutta ordem 2

$$y^{n+1} = y^n + \Delta t \left(\frac{1}{2} k_1 + \frac{1}{2} k_2 \right)$$

$$k_1 = f(t_n, y^n)$$

$$k_2 = f(t_n + \Delta t, y^n + \Delta t k_1)$$

Tabela de Butcher

0	0	0
1	1	0
	$\frac{1}{2}$	$\frac{1}{2}$

```
function [y time]=rk2(f,y0,t0,dt,nt)
```

```
time(1)=t0; y(:,1)=y0;
```

```
nstage=2;a=[0 0;1 0];c=[0; 1];b=[0.5 0.5];
```

```
for n=1:nt
```

```
    K(:,1)=feval(f,y(:,n),time(n));
```

```
    for m=2:nstage
```

```
        tt=time(n)+c(m)*dt;
```

```
        yy=y(:,n)+dt*K(:,1:m-1)*a(m,1:m-1)';
```

```
        K(:,m)=feval(f,yy,tt);
```

```
    endfor
```

```
    y(:,n+1)=y(:,n)+dt*K(:,1:nstage)*b';
```

```
    time(n+1)=time(n)+dt;
```

```
endfor
```

```
function f = foscil(y,t)
```

```
    ome=1.;
```

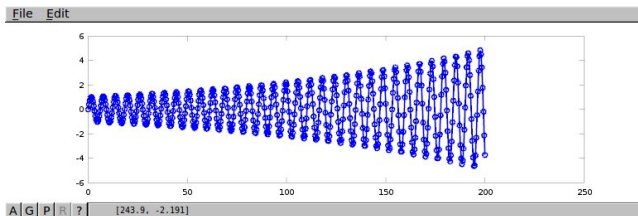
```
    f(1)=y(2);
```

```
    f(2)=-ome*ome*y(1);
```

```
end
```

- Resolvemos $x' = v$, $v' = -x$ com $x(0) = 0$, $v(0) = 1$. A solução exata é $x(t) = \sin t$, $v(t) = \cos t$.
- Matriz jacobiana = $\begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$, então $J_f = 1$ (autovalores $\pm i$), limite de estabilidade $\Delta t < 2$.

```
> [y time]=rk2("foscil",[0;1],0,0.4,500);
> plot(time,y(1,:),"-o","linewidth",2)
```



- A pouca precisão leva a comportamento errado.

Runge-Kutta de ordem 4

■ Tabela de Butcher:

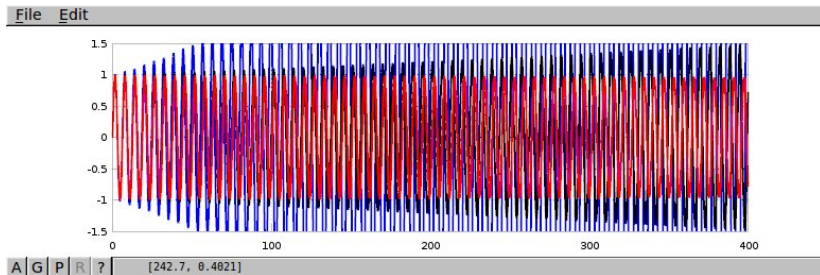
0	0	0	0	0
$\frac{1}{2}$	$\frac{1}{2}$	0	0	0
$\frac{1}{2}$	0	$\frac{1}{2}$	0	0
1	0	0	1	0
	$\frac{1}{6}$	$\frac{2}{6}$	$\frac{2}{6}$	$\frac{1}{6}$

■ Mesmo código do slide anterior, com

```
nstage=4;  
a=[0 0 0 0;1/2 0 0 0;0 1/2 0 0;0 0 1 0];  
c=[0; 1/2; 1/2; 1];  
b=[1/6 2/6 2/6 1/6];
```

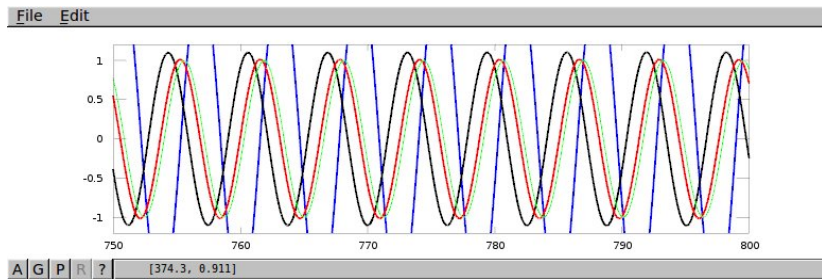
Comparamos RK2 com $\Delta t = 0.4$ e 0.2 , e RK4 com $\Delta t = 0.4$.

```
>[y1 time1]=rk2("foscil",[0;1],0,0.4,1000);  
>[y2 time2]=rk2("foscil",[0;1],0,0.2,2000);  
>[y4 time4]=rk4("foscil",[0;1],0,0.4,1000);  
> plot(time1,y1(1,:),"-b","linewidth",2,...  
time2,y2(1,:),"-k","linewidth",2,...  
time4,y4(1,:),"-r","linewidth",2)  
> axis([0 400 -1.5 1.5])
```



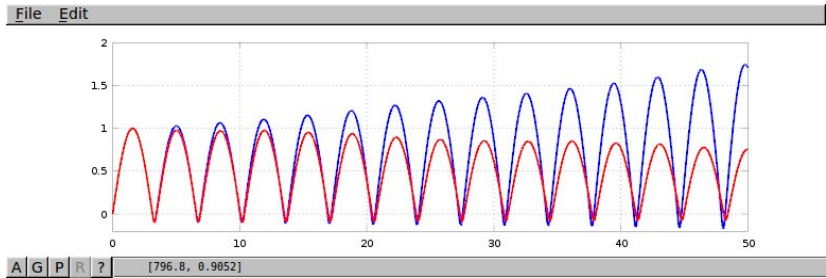
Qual é o Δt tal que a $t = 800$ a amplitude continua sendo $\simeq 1$ com RK2?

```
>[y1 time1]=rk2("foscil",[0;1],0,0.2,4000);  
>[y2 time2]=rk2("foscil",[0;1],0,0.1,8000);  
>[y4 time4]=rk2("foscil",[0;1],0,0.05,16000);  
>plot(time1,y1(1,:),"-b","linewidth",2,time2,y2(1,:),"-k",...  
"linewidth",2,time4,y4(1,:),"-r","linewidth",2,...  
time1,sin(time1),"-g","linewidth",1)  
> axis([750 800 -1.2 1.2])
```

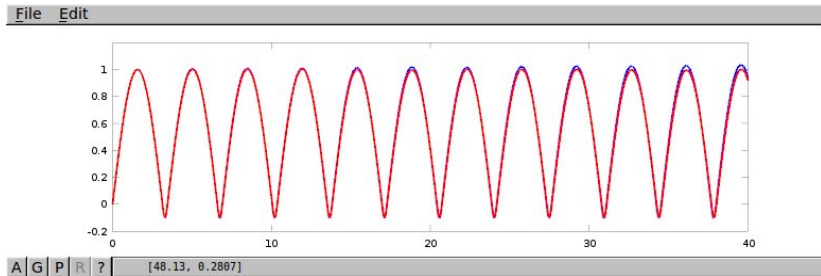


Massa-mola com parede

```
function f = foscilw(y,t)
ome=1.;f(1)=y(2);
if (y(1)>0) f(2)=-ome*ome*y(1);
else f(2)=-100*ome*ome*y(1);
endif
end
>[y1 time1]=rk2("foscilw",[0;1],0,0.1,500);
>[y2 time2]=rk4("foscilw",[0;1],0,0.2,250);
>plot(time1,y1(1,:),"-b","linewidth",2,time2,y2(1,:),"-r",...
"linewidth",2)
> axis([0 50 -.2 2])
```



```
>[y1 time1]=rk2("foscilw",[0;1],0,0.02,2000);  
>[y2 time2]=rk4("foscilw",[0;1],0,0.04,1000);  
>plot(time1,y1(1,:),"-b","linewidth",2,time2,y2(1,:),"-r",...  
"linewidth",2)  
> axis([0 40 -.2 1.2])
```



Ajuste automático de passo

Ideia geral: Em cada passo de tempo,

- Calcular y^{n+1} com **dois** métodos de diferente ordem.
Exemplo: RK2 $\rightarrow y^{n+1}$, RK4 $\rightarrow z^{n+1}$.
- Estimar o erro como a diferença desses resultados.

$$e = \|y^{n+1} - z^{n+1}\|$$

- Se (erro > tolerância) reduzir Δt e voltar a 1.
- Predizer um Δt adequado para próximos passos. Vamos supor que o esquema de menor ordem é de ordem p , o “passo ideal” Δt_* daria erro igual a tolerância.

$$e \simeq C \Delta t^{p+1}, \quad \epsilon \simeq C \Delta t_*^{p+1} \Rightarrow \Delta t_* \simeq \Delta t \left(\frac{\epsilon}{e} \right)^{\frac{1}{p+1}}$$

$$\Delta t \leftarrow \max(0.5\Delta t, \min(2\Delta t, 0.9\Delta t_*))$$

Embedded Runge-Kutta methods (ERKM)

- Para ajustar automaticamente Δt são precisos **2 métodos** de diferente ordem.
- Em geral, esses dois métodos requerem avaliações **em pontos diferentes**.
- ERKM consegue construir os dois métodos **maximizando o número de pontos comuns**.
- São **pares otimizados de métodos RK**. Os mais utilizados são os de **Fehlberg**, de **Dormand-Prince**, entre outros. Matlab e Octave utilizam ERKM.

Método de Dormand-Prince

0	0	0	0	0	0	0	0
$\frac{1}{5}$	$\frac{1}{5}$	0	0	0	0	0	0
$\frac{3}{10}$	$\frac{3}{40}$	$\frac{9}{40}$	0	0	0	0	0
$\frac{4}{5}$	$\frac{44}{45}$	$-\frac{56}{15}$	$\frac{32}{9}$	0	0	0	0
$\frac{8}{9}$	$\frac{19372}{6561}$	$-\frac{5360}{2187}$	$\frac{64448}{6561}$	$-\frac{212}{729}$	0	0	0
1	$\frac{9017}{3168}$	$-\frac{355}{33}$	$\frac{46732}{5247}$	$\frac{49}{176}$	$-\frac{5103}{18656}$	0	0
1	$\frac{35}{384}$	0	$\frac{500}{1113}$	$\frac{125}{192}$	$-\frac{2187}{6784}$	$\frac{11}{84}$	0
	$\frac{35}{384}$	0	$\frac{500}{1113}$	$\frac{125}{192}$	$-\frac{2187}{6784}$	$\frac{11}{84}$	0
	$\frac{5179}{57600}$	0	$\frac{7571}{16695}$	$\frac{393}{640}$	$-\frac{92097}{339200}$	$\frac{187}{2100}$	$\frac{1}{40}$

Há 2 vetores b , o primeiro é de ordem 4, o segundo de ordem 5.

Utilizando lsode (Octave)

- A função `lsode` implementa as melhoras discutidas.
- Se especifica a função e os tempos em que se deseja a solução, Δt é automático.

```
T=[0:0.2:40];  
[X, istate, MSG]=lsode("foscilw",[0 1],T);  
plot(T,X,"linewidth",2)
```

