

Exemplo1

April 11, 2020

1 Introdução

Apresentaremos vários testes para o problema de valor inicial (PVI)

$$\begin{cases} x' = -2tx^2 \\ x(0) = 1 \end{cases}$$

cuja solução exata é

$$x(t) = \frac{1}{1+t^2}.$$

Todas as simulações serão feitas no intervalo $[0, 1]$. Primeiramente, vamos carregar o pacote *DifferentialEquations* e definir o problema:

```
[1]: using DifferentialEquations
f(u,p,t) = -2*t*u^2          # p é para parâmetros, que não há no nosso
    ↪ exemplo, mas acho que a sintaxe exige assim
u0 = 1.0                     # condição inicial
tspan = (0.0,1.0)           # intervalo de tempo para a simulação
prob = ODEProblem(f, u0, tspan) # Definição do problema
```

```
[1]: ODEProblem with uType Float64 and tType Float64. In-
place: false
timespan: (0.0, 1.0)
u0: 1.0
```

As aproximações serão comparadas com os valores da solução exata nos pontos $vt = 0, 0.1, 0.2, \dots, 1$

```
[2]: vt = [i*0.1 for i = 0:10]
u_exata = 1.0./(1 .+ vt.^2)
[vt u_exata]
```

```
[2]: 11×2 Array{Float64,2}:
 0.0  1.0
 0.1  0.990099
 0.2  0.961538
 0.3  0.917431
 0.4  0.862069
 0.5  0.8
```

```

0.6  0.735294
0.7  0.671141
0.8  0.609756
0.9  0.552486
1.0  0.5

```

2 Método de Euler

Primeiramente, usaremos o método de Euler com passos $h = 0.1, 0.01$ e 0.001

```

[3]: sol = solve(prob, Euler(), dt = 0.1)      # h = 0.1
     euler_a = sol.vt
     sol = solve(prob, Euler(), dt = 0.01)     # h = 0.01
     euler_b = sol.vt
     sol = solve(prob, Euler(), dt = 0.001)    # h = 0.001
     euler_c = sol.vt
     [vt euler_a euler_b euler_c u_exata]      # aproximações e solução exata nos
     ↪ pontos escolhidos

```

```

[3]: 11×5 Array{Float64,2}:
 0.0  1.0      1.0      1.0      1.0
 0.1  1.0      0.991069  0.990196  0.990099
 0.2  0.98      0.963301  0.961714  0.961538
 0.3  0.941584  0.919686  0.917655  0.917431
 0.4  0.888389  0.864485  0.862309  0.862069
 0.5  0.82525   0.802293  0.800227  0.8
 0.6  0.757147  0.737271  0.73549   0.735294
 0.7  0.688354  0.672702  0.671296  0.671141
 0.8  0.622018  0.610879  0.609867  0.609756
 0.9  0.560113  0.553198  0.552557  0.552486
 1.0  0.503642  0.500356  0.500035  0.5

```

Vejamos agora os erros, medidos nos pontos selecionados de acordo com

$$erro_x = \max_{0 \leq i \leq 10} |u_{exata}(vt_i) - euler_x(vt_i)|$$

onde $x = a, b$ ou c

```

[4]: erro_a = maximum(abs.(u_exata - euler_a))
     erro_b = maximum(abs.(u_exata - euler_b))
     erro_c = maximum(abs.(u_exata - euler_c))
     [erro_a erro_b erro_c]

```

```

[4]: 1×3 Array{Float64,2}:
 0.0263202  0.00241596  0.000239544

```

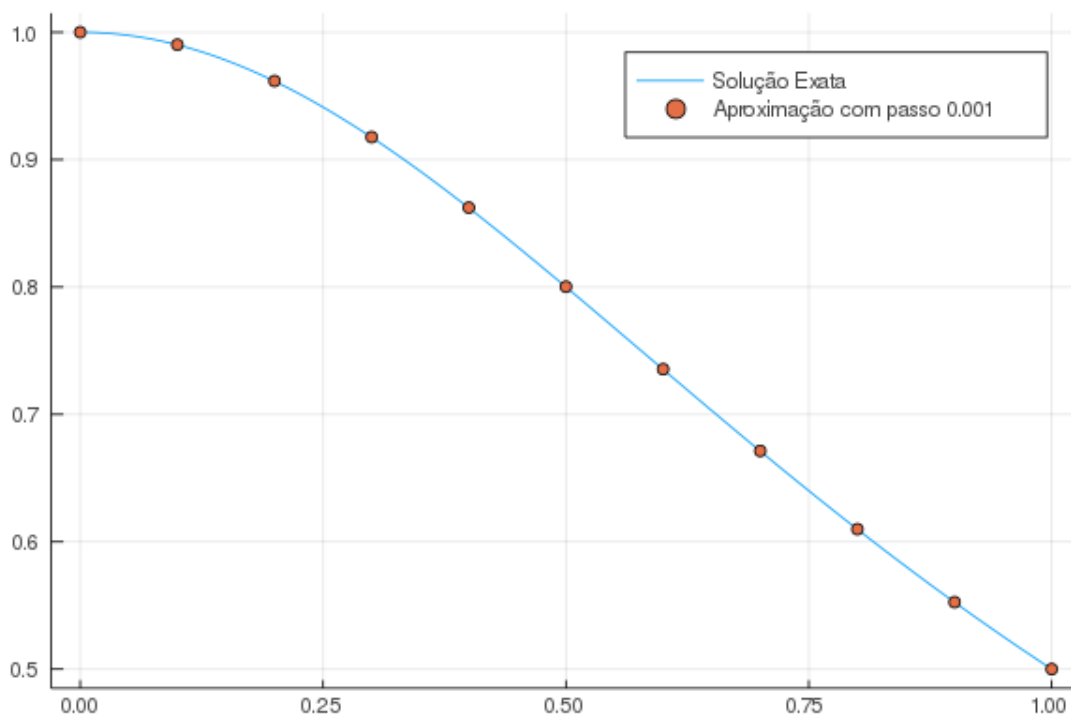
```
[5]: [erro_a/erro_b erro_b/erro_c] # razões entre os erros
```

```
[5]: 1×2 Array{Float64,2}:  
    10.8943  10.0857
```

As simulações sugerem que ao dividirmos h por 10, o erro é dividido aproximadamente por 10, indicando um decaimento proporcional a h . Façamos agora um gráfico, usando a solução exata de 0.001 em 0.001 e a última aproximação nos pontos de comparação

```
[6]: using Plots  
gr(fmt=:png)  
plot(sol.t,t->1.0/(1.0+t^2),label = "Solução Exata")  
scatter!(vt,euler_c,label = "Aproximação com passo 0.001")
```

[6]:



3 Método do Ponto Médio

Usaremos agora o Método do Ponto Médio. Lembre-se que um passo do método para a equação diferencial ordinária

$$x' = f(t, x)$$

pode ser descrito como: se η_n denota a aproximação calculada no instante t_n , então a aproximação η_{n+1} no instante $t_{n+1} = t_n + h$ é obtida por

$$k_1 = f(t_n, \eta_n) \quad (1)$$

$$k_2 = f(t_n + 0.5 h, \eta_n + 0.5 h k_1) \quad (2)$$

$$\eta_{n+1} = \eta_n + h k_2 \quad (3)$$

Apresentaremos resultados para $h = 0.1, 0.05$ e 0.025 , permitindo-nos algumas comparações interessantes

```
[7]: sol = solve(prob, Midpoint(), adaptive = false, dt = 0.1)    # h = 0.1, fixo
      pm_a = sol.vt
      sol = solve(prob, Midpoint(), adaptive = false, dt = 0.05) # h = 0.05, fixo
      pm_b = sol.vt
      sol = solve(prob, Midpoint(), adaptive = false, dt = 0.025) # h = 0.025, fixo
      pm_c = sol.vt
      [pm_a pm_b pm_c u_exata]    # aproximações e solução exata nos pontos escolhidos
```

```
[7]: 11×4 Array{Float64,2}:
      1.0      1.0      1.0      1.0
      0.99      0.990075  0.990093  0.990099
      0.961176  0.961451  0.961517  0.961538
      0.916742  0.917266  0.917391  0.917431
      0.861104  0.86184   0.862013  0.862069
      0.798887  0.799739  0.799937  0.8
      0.73418   0.735035  0.735232  0.735294
      0.670145  0.670911  0.671086  0.671141
      0.608952  0.609573  0.609712  0.609756
      0.551905  0.552355  0.552455  0.552486
      0.499638  0.49992   0.499981  0.5
```

```
[8]: erro_a = maximum(abs.(u_exata - pm_a))
      erro_b = maximum(abs.(u_exata - pm_b))
      erro_c = maximum(abs.(u_exata - pm_c))
      [erro_a erro_b erro_c]
```

```
[8]: 1×3 Array{Float64,2}:
      0.00111446  0.000261065  6.33162e-5
```

```
[9]: [erro_a/erro_b erro_b/erro_c]    # razões entre os erros
```

```
[9]: 1×2 Array{Float64,2}:
      4.2689  4.12319
```

As simulações sugerem que ao dividirmos h por 2, o erro é dividido aproximadamente por 4, indicando um decaimento proporcional a h^2 . Este ganho de precisão vem com um custo. Note que um passo do Método do ponto médio usa duas avaliações do campo f , enquanto que um passo do Método de Euler usa apenas uma. É natural questionar se este custo adicional compensa. Observe que o erro estimado para o Método de

Euler com $h = 0.001$ é aproximadamente 2.4×10^{-4} e o erro para o Método do Ponto Médio com $h = 0.05$ é comparável, sendo aproximadamente 2.6×10^{-4} . O Método de Euler usou 1000 passos e portanto 1000 avaliações de f , enquanto que o Método do Ponto Médio usou 20 passos e conseqüentemente 40 avaliações de f , o que é uma economia significativa para esta ordem de grandeza do erro.

4 Método Runge-Kutta Clássico

O famoso Método Runge-Kutta Clássico (RK4 daqui em diante) é um método de ordem 4 (o erro decai proporcionalmente a h^4) cujo avanço no tempo é descrito por

$$k_1 = f(t_n, \eta_n) \quad (4)$$

$$k_2 = f(t_n + 0.5 h, \eta_n + 0.5 h k_1) \quad (5)$$

$$k_3 = f(t_n + 0.5 h, \eta_n + 0.5 h k_2) \quad (6)$$

$$k_4 = f(t_n + h, \eta_n + h k_3) \quad (7)$$

$$\eta_{n+1} = \eta_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \quad (8)$$

Note que os parâmetros são simples pois o método foi desenvolvido para ser usado em uma época que não havia computadores. Vejamos os resultados numéricos.

```
[10]: sol = solve(prob,RK4(),adaptive=false,dt=0.1)    # h = 0.1 fixo
      rk4_a = sol.(vt)
      sol = solve(prob,RK4(),adaptive=false,dt=0.05)  # h = 0.05 fixo
      rk4_b = sol.(vt)
      sol = solve(prob,RK4(),adaptive=false,dt=0.025) # h = 0.025 fixo
      rk4_c = sol.(vt)
      # O comando abaixo é para mostrar em formato não compacto, pois a precisão é
      ↪ alta
      show(IOContext(stdout, :compact=>false), "text/plain", [rk4_a rk4_b rk4_c,
      ↪ u_exata])
```

11×4 Array{Float64,2}:

1.0	1.0	1.0	1.0
0.9900989249501665	0.9900990047470648	0.9900990095850809	0.9900990099009901
0.9615381436580871	0.9615384431173571	0.9615384604316574	0.9615384615384615
0.9174305975195713	0.9174311594495312	0.9174311907018797	0.9174311926605504
0.8620681834882848	0.8620689234704948	0.8620689630855644	0.8620689655172413
0.7999992090185386	0.7999999593811166	0.7999999977111145	0.8
0.7352935002790218	0.7352940884180565	0.735294116082253	0.7352941176470588
0.6711406195178697	0.6711409281606623	0.6711409391195511	0.6711409395973154
0.6097561191882678	0.6097561058751381	0.6097560982701216	0.6097560975609756
0.552486529856882	0.5524862142982742	0.5524861896333003	0.5524861878453039
0.5000006022105239	0.500000040931104	0.5000000026414391	0.5

```
[11]: erro_a = maximum(abs.(u_exata - rk4_a))
      erro_b = maximum(abs.(u_exata - rk4_b))
      erro_c = maximum(abs.(u_exata - rk4_c))
      [erro_a erro_b erro_c]
```

```
[11]: 1×3 Array{Float64,2}:
      7.90981e-7  4.20467e-8  2.64144e-9
```

```
[12]: [erro_a/erro_b erro_b/erro_c]  # razões entre os erros
```

```
[12]: 1×2 Array{Float64,2}:
      18.812  15.9181
```

Sendo um método de ordem 4, o fator deve ser aproximadamente 16, pois estamos dividindo h por 2. Para uma comparação com o Método do Ponto Médio, usaremos neste último um passo que garanta um erro comparável com o obtido para o RK4 com $h = 0.1$

```
[13]: sol = solve(prob, Midpoint(), adaptive = false, dt = 0.0025)  # h = 1/400
      pm = sol.vt
      maximum(abs.(u_exata-pm))  # erro
```

```
[13]: 6.166297614740301e-7
```

Temos então 800 avaliações de f para o Método do Ponto Médio contra 40 avaliações para o método RK4.

```
[ ]:
```