Tratamento Numérico de Equações diferenciais ordinárias

Alisson Vinicius Gonsales - gonsalesalisson@usp.br
Instituto de Matemática e Estatística - IME-USP
(Dated: Janeiro 2022)

I. Introdução

Testaremos um método linear multipasso inventado para estudar dois problemas modelo: crescimento logístico e o problema do cosseno.
Para um método linear multipasso, temos sempre a relação:

$$\sum_{j=0}^{n} \alpha_j = h \sum_{j=0}^{n} \beta_j f_j \tag{1}$$

Para $\beta_n \neq 0$ temos um método implícito. Para tais métodos, o erro global pode ser estimado através de coeficientes C_p:

$$\tau = h\alpha_k = \sum_p C_p h^p y'(t_k)$$

Para um método de ordem p, devemos ter os coeficientes com índice menor ou igual a p nulos.

II. Escolha do método

Considere um método de três passos, implícito, linear com os coeficientes abaixo:

$$\begin{cases} \alpha_0 = -c \quad \alpha_1 = \frac{m}{n}(c-1) \;\; \alpha_2 = \frac{n-m}{n}(c-1) \quad \alpha_3 = 1 \quad n \in N \quad m = 0, 1, 2, ...n \\ \qquad\qquad \beta_1 = 0 \quad \beta_2 = 0 \end{cases}$$

Com c real. Que $C_0 = 0$ é imediato. Queremos determinar β_0 e β_3 de forma que C_1 e C_2 também sejam nulos para obter um método com ordem de consistência dois. Reescrevendo as equações para os coeficientes C_p obtemos o seguinte sistema de equações:

$$\begin{cases} \beta_0 + \beta_3 \;\; = \frac{m}{n}(c-1) + 2\frac{(n-m)}{n}(c-1) + 3 \\ \qquad 3\beta_3 \;\; = \frac{m}{n}\frac{(c-1)}{2} + 2\frac{(n-m)}{n}(c-1) + \frac{9}{2} \end{cases}$$

Com solução dada por:

$$\beta_0 = \frac{8n-3m}{6n}(c-1) + \frac{3}{2} \quad \beta_3 = \frac{4n-3m}{6n}(c-1) + \frac{3}{2}$$

Temos portanto a forma do método, para uma constante c real, dada por:

$$X_{k+3} = cX_k + \frac{m}{n}(1-c)X_{k+1} + \frac{n-m}{n}(1-c)X_{k+2} + h[(\frac{3m-8n}{6n}(1-c) + \frac{3}{2})f_k + (\frac{3m-4n}{6n}(1-c) + \frac{3}{2})f_{k+3}]$$

Devemos encontrar um valor de c que garanta a estabilidade absoluta do método. Seu primeiro e segundo polinômios característicos serão dados por:

$$\begin{cases} p(r) \;\; = -c - \frac{m}{n}(1-c)r - \frac{n-m}{n}(1-c)r^2 + r^3 \\ \sigma(r) \;\; = \frac{3m-8n}{6n}(1-c) + \frac{3}{2} + \frac{3m-4n}{6n}(1-c)r^3 + \frac{3}{2}r^3 \end{cases}$$

Faremos a escolha c = 0.666, m = 314, n = 340:

```
c = 0.666
m = 314
n = 340
np.roots([1,((m - n)/n)*(c -1),(m/n)*(c -1), -c])
abs(p[0]), abs(p[1]),abs(p[2])
#Output > (0.9809779861171262, 0.8239625704934533, 0.8239625704934533)
```

Vemos que neste caso as raízes de p(r) tem valor absoluto menor do que um, e portanto trata-se de um método absolutamente estável. Seguiremos com esse método a partir de agora.

```
def AlissonMethod():
  c = 0.666
  n = 340
  m = 314
  alphas = [-c, (m/n)*(c-1), (n - m)*(c-1)/n, 1]
  betas = [(8*n - 3*m)*(c-1)/(6*n),0,0, (4*n - 3*m)*(c-1)/(6*n)]
  return [alphas, betas]
```

O polinômio de estabilidade do método será dado por:

$$\pi(z, r) = p(r) - z\sigma(r)$$

Podemos analisar a região de estabilidade do método para escolher um bom passo temporal inicial para o próximo item.

```
def plotStabilityRegion(f, n):

  mthd = f #metodo
  z = []  #velores de z da regiao

  #Gerando os valores de theta com modulo 1
  tv = np.linspace(0, 2*math.pi,1000)
  theta = [cmath.exp(1j*ts) for ts in tv]

  #Gerando grid 2-D
  x0,x1 = -2, 9
  y0,y1 = -5,5
  Nx, Ny = 100, 100
  xv    = np.linspace(x0,x1,Nx);
  yv    = np.linspace(y0,y1,Ny);
  [xs,ys] = np.meshgrid(xv,yv);


  zi = xs + 1j*ys

  #Campo escalar nulo inicialmente
  field = xs*0 + ys*0

  for i in range(0,Nx):
    for j in range(0,Ny):
      #Polinomio de estabilidade absoluta
      pi = []
      for k in range(0,n):
        #Criando um array com os coeficientes do polinomio de estabilidade
        pi.append(mthd[0][k] - zi[i][j]*mthd[1][k])
      pi = pi[::-1]

      #Gerando um campo media dos valores abs das raizes
      root = np.sum([abs(d) for d in np.roots(pi)])/(n - 1)
      field[i][j] = root
      pi = 0

  # Achando os valores de z com os polinomios caracter sticos
  for x in theta:
    #primeiro polinomio
    p = 0
    #segundo polinomio
    g = 0
```

```
    for i in range(0,n):
      p += mthd[0][i]*(x**i)
      g += mthd[1][i]*(x**i)
    z.append((p/g))
```

Onde usamos o fato de que se todas as raízes tem valor absoluto inferior a um, sua média também terá.

$$|z_i| \leq 1 \quad \Longrightarrow \quad \frac{1}{n}\sum_{i=0}^{n} |z_i| \leq 1$$

```
  #Construindo o plot
  fig, ax = plt.subplots()
  levels = np.linspace(0,0.95,100);
  b = ax.contourf(xs, ys, field,levels)
  ax.plot([w.real for w in z], [w.imag for w in z],color='green')
  ax.hlines(y=0, xmin=-1.5, xmax=8.2, linewidth=2, color='black',linestyle='dashed')
  ax.grid()
  ax.vlines(x=0, ymin=-5, ymax=5, linewidth=2, color='black',linestyle='dashed')
  ax.vlines(x=6.2, ymin=-5, ymax=5, linewidth=2, color='black',linestyle='dashed')

  ax.set_ylabel('Im(z)')
  ax.set_xlabel('Re(z)')
  ax.set_title('Regi o de estabilidade do m todo')

  plt.show()

#function(metodo, grau do polinomio)
plotStabilityRegion(AlissonMethod(),4)
```

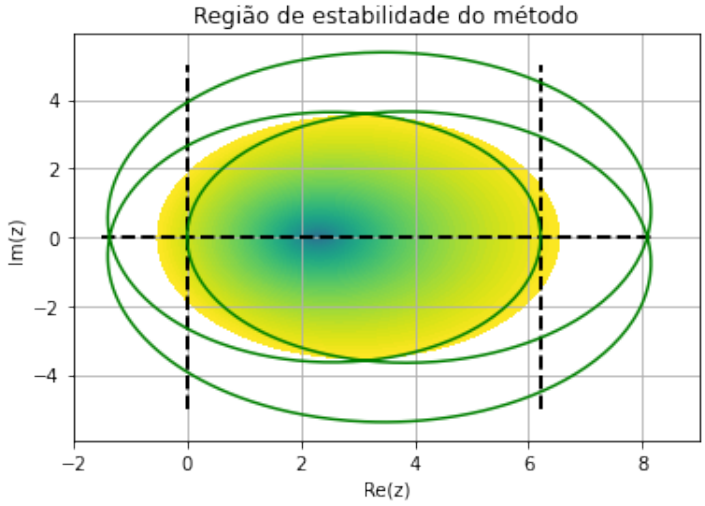

Figura 1: A figura mostra o contorno gerado para o bordo da região de estabilidade do método. As linhas pontilhadas mostram os limites a região elipsoidal central delineada em verde. Essa região está colorida de acordo com o valor absoluto das raízes do polinômio de estabilidade, de 0 até aproximadamente 1

III. Testando a convergência do método

Testaremos o método para discretizar o problema modelo Crescimento Logístico com condição de contorno, dado por:

$$\begin{cases} y' = r\left(1 - \frac{y}{K}\right) y \\ y_0 = 1 \end{cases}$$

O problema admite solução analítica na forma:

$$y(t) = \frac{y_0 K}{[y_0 + (K - y_0)exp(-rt)]}$$

Com uma aplicação do método, após a inicialização, obtemos:

$$X_{k+3} + \alpha_0 X_k + \alpha_1 X_{k+1} + \alpha_2 X_{k+2} - h\beta_0 r(1 - \frac{X_k}{K})X_k - h\beta_3 r(1 - \frac{X_{k+3}}{K})X_{k+3} = 0$$

Ou seja, chegamos a equação de segundo grau em X_{k+3}:

$$\frac{h\beta_3 r}{K} X_{k+3}^2 + (1 - h\beta_3 r)X_{k+3} + \alpha_0 X_k + \alpha_1 X_{k+1} + \alpha_2 X_{k+2} - h\beta_0 r(1 - \frac{X_k}{K})X_k = 0$$

Que possui duas soluções. Tomaremos a solução mais conveniente para estudar o problema no intervalo (0,20) x (1, y(20))

$$X_{k+3} = \frac{K}{h\beta_3 r}(1 - h\beta_3 r)(-1 + [1 - 4\frac{h\beta_3 r}{K(1 - h\beta_3 r)^2}(\alpha_0 X_k + \alpha_1 X_{k+1} + \alpha_2 X_{k+2} - h\beta_0 r(1 - \frac{X_k}{K})X_k)]^{1/2})$$

```
#Funcao com o metodo
def AlissonMethod(vt, vy_0 , dt, initmethod, ASolution):

 #Parametros e coeficientes do metodo
  c = 0.666
  m = 314
  n = 340
  a0 = -c
  a1 = (m/n)*(c - 1)
  a2 = ((n-m)/n)*(c - 1)
  b0 = ((8*n - 3*m)/(6*n))*(c - 1) + 3/2
  b3 = ((-3*m + 4*n)/(6*n))*(c - 1) + 3/2
  npass = 3
  K = 100000
  r = 2
  err = [0]

  #Constantes da solucao da eq. 2 grau
  A = (dt*b3*r)/K
  B = (1 - dt*b3*r)
  #condicao inicial
  vy = vy_0.copy()

  #inicializacao
  vy, err = initmethod(vt,vy,dt,npass,err,ASolution)

  for k in range(3, len(vt)):
    #Constante C da solucao da eq. 2 grau
    C = a0*vy[k-3] + a1*vy[k-2] + a2*vy[k-1] - dt*b0*vy[k-3]*r*(1 - (vy[k-3]/K))
    vy.append((-B + ((B**2) - (4*A*C))**0.5 )/(2*A))
    err.append(abs(ASolution[len(vy)-1] - vy[len(vy)-1]))
  return np.array([vy,err])
```

A. Inicialização utilizando o método de Euler

n	$h = \frac{20-0}{n}$	$\|e(t,h)\|$	$q = \|e(t,2h)/e(t,h)\|$	$log_2(q)$ (Ordem)
4	5.00000e+00	1.79297e+04		
8	2.50000e+00	6.79801e+04	2.63750e-01	-1.92276e+00
16	1.25000e+00	7.19123e+04	9.45319e-01	-8.11261e-02
32	6.25000e-01	9.64606e+04	7.45509e-01	-4.23702e-01
64	3.12500e-01	7.02756e+04	1.37260e+00	4.56915e-01
128	1.56250e-01	2.18604e+04	3.21475e+00	1.68471e+00
256	7.81250e-02	7.46946e+03	2.92663e+00	1.54924e+00
512	3.90625e-02	2.85347e+03	2.61768e+00	1.38829e+00
1024	1.95312e-02	1.20284e+03	2.37228e+00	1.24627e+00
2048	9.76562e-03	5.45055e+02	2.20682e+00	1.14197e+00
4096	4.88281e-03	2.58358e+02	2.10969e+00	1.07703e+00
8192	2.44141e-03	1.25628e+02	2.05653e+00	1.04021e+00
16384	1.22070e-03	6.19236e+01	2.02876e+00	1.02060e+00

Tabela I: Valores do erro global máximo e estimativas da ordem de convergência do método para a aproximação obtida com inicialização pelo método de Euler.

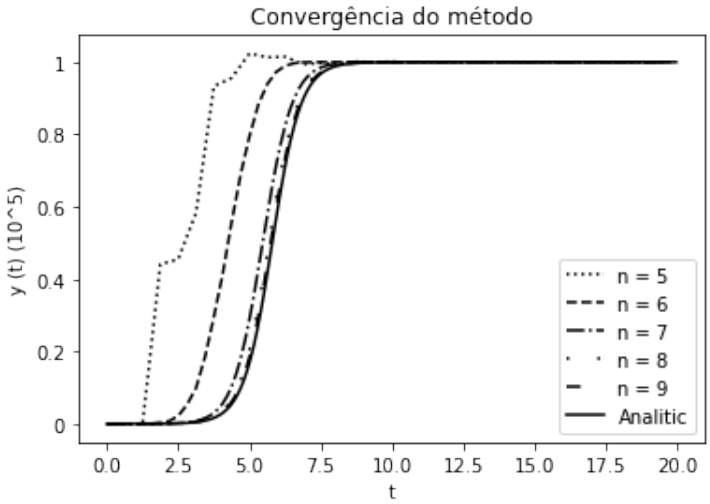

Figura 2: Convergência dos gráficos das aproximações para o gráfico da solução exata com a diminuição do passo de integração para inicialização pelo método de Euler

Vemos pela tabela e imagem acima que as aproximações convergem para a solução exata do problema com ordem 1. A princípio, a inicialização não-consistente gerada pelo método de Euler deveria fazer o método ter o efeito de pequenas perturbações nas aproximações iniciais. No entanto, vemos que o erro não diminui proporcionalmente ao passo de integração, então é justo se perguntar se a diminuição do passo com inicialização pelo método de Euler é suficiente para convergência nesse caso. Obteremos mais informações com o método a seguir.

```
#Metodo Euler expl cito
def Euler(vt_i, vyi_0, dt_0, npass,err, ASolution):
  #Condicao inicial
  vyi_1 = vyi_0.copy()
```

```
#Laco de repeticao
for k in range(0,npass-1):
  vyi_1.append(vyi_1[k] + dt_0*model(vt_i[k],vyi_1[k]))

  #Erro global
  err.append(abs(ASolution[len(vyi_1)-1] - vyi_1[len(vyi_1)-1]))
return vyi_1, err
```

B. Inicialização utilizando o método RK22

n	$h = \frac{20-0}{n}$	$\|e(t,h)\|$	$q = \|e(t,2h)/e(t,h)\|$	$log_2(q)$ (Ordem)
4	5.00000e+00	1.43452e+04		
8	2.50000e+00	6.79820e+04	2.11015e-01	-2.24458e+00
16	1.25000e+00	7.19130e+04	9.45337e-01	-8.10997e-02
32	6.25000e-01	9.64617e+04	7.45508e-01	-4.23705e-01
64	3.12500e-01	7.08945e+04	1.36064e+00	4.44283e-01
128	1.56250e-01	2.27239e+04	3.11982e+00	1.64146e+00
256	7.81250e-02	7.75365e+03	2.93073e+00	1.55126e+00
512	3.90625e-02	2.93212e+03	2.64438e+00	1.40293e+00
1024	1.95312e-02	1.22342e+03	2.39665e+00	1.26102e+00
2048	9.76562e-03	5.50314e+02	2.22313e+00	1.15259e+00
4096	4.88281e-03	2.59686e+02	2.11915e+00	1.08349e+00
8192	2.44141e-03	1.25962e+02	2.06163e+00	1.04378e+00
16384	1.22070e-03	6.20076e+01	2.03140e+00	1.02247e+00

Tabela II: Valores do erro global máximo e estimativas da ordem de convergência do método para a aproximação obtida com inicialização pelo método de Runge Kutta.

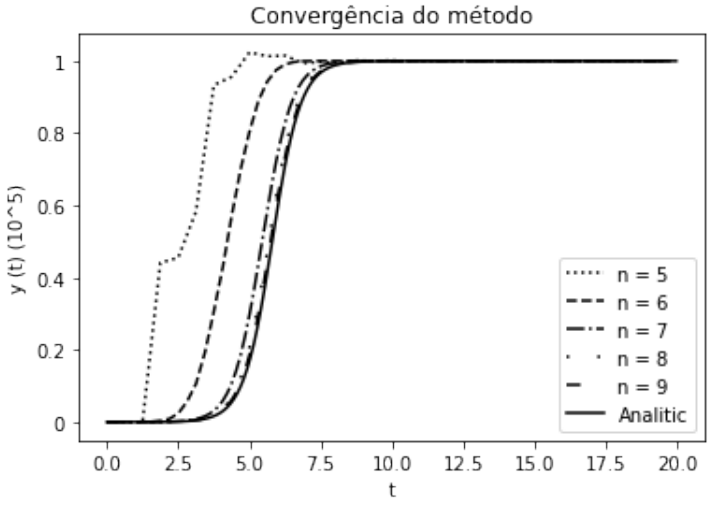

Figura 3: Convergência dos gráficos das aproximações para o gráfico da solução exata com a diminuição do passo de integração para inicialização pelo método RK22

Vemos pela tabela e imagem acima que as aproximações convergem para a solução exata com ordem de convergência 1. Dado que o método tem ordem 2 com inicialização também com ordem 2, isso pode significar que a solução exata é

menos suave do que o esperado e sendo assim, deveremos analisar a continuidade de suas derivadas. Antes, testamos com inicialização dada pela solução exata.

```
#Metodo de Runge Kutta de 2 estagios e ordem 2
def RungeKutta22(vt_i, vyi_0, dt_0, npass,err,ASolution):
  #Condicao inicial
  vyi_1 = vyi_0.copy()
  #Laco de repeticao
  for k in range(0,npass-1):
    #kappa1
    v0 = model(vt_i[k],vyi_1[k])
    vyi_1.append(vyi_1[k] + dt_0*model(vt_i[k] + 0.5*dt_0,vyi_1[k] + 0.5*dt_0*v0))
    #erro global
    err.append(abs(ASolution[len(vyi_1)-1] - vyi_1[len(vyi_1)-1]))

  return vyi_1, err
```

C. Inicialização utilizando a Solução analítica

n	$h = \frac{20-0}{n}$	$\|e(t,h)\|$	$q = \|e(t,2h)/e(t,h)\|$	$log_2(q)$ (Ordem)
4	5.00000e+00	6.92917e+03		
8	2.50000e+00	6.79803e+04	1.01929e-01	-3.29436e+00
16	1.25000e+00	7.19120e+04	9.45326e-01	-8.11156e-02
32	6.25000e-01	9.64601e+04	7.45510e-01	-4.23701e-01
64	3.12500e-01	6.87113e+04	1.40385e+00	4.89386e-01
128	1.56250e-01	1.80096e+04	3.81525e+00	1.93178e+00
256	7.81250e-02	5.01056e+03	3.59434e+00	1.84572e+00
512	3.90625e-02	1.50491e+03	3.32948e+00	1.73530e+00
1024	1.95312e-02	5.00233e+02	3.00842e+00	1.58900e+00
2048	9.76562e-03	1.86752e+02	2.67860e+00	1.42148e+00
4096	4.88281e-03	7.74911e+01	2.40998e+00	1.26902e+00
8192	2.44141e-03	3.47695e+01	2.22871e+00	1.15621e+00
16384	1.22070e-03	1.63885e+01	2.12157e+00	1.08513e+00

Tabela III: Valores do erro global máximo e estimativas da ordem de convergência do método para a aproximação obtida com inicialização pela solução analítica.

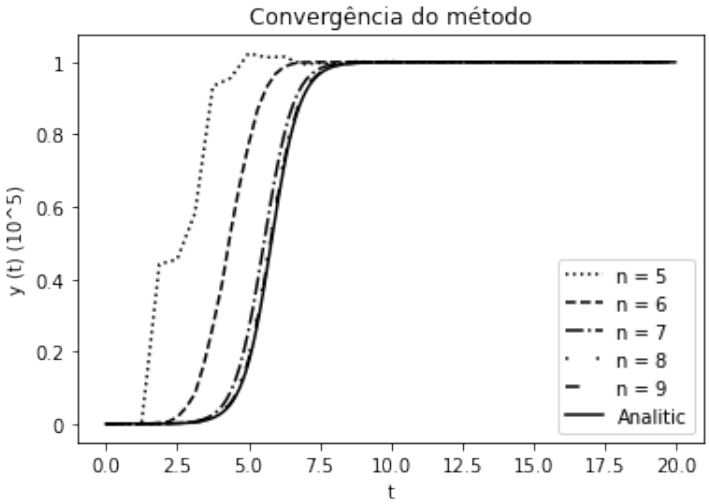

Figura 4: Convergência dos gráficos das aproximações para o gráfico da solução exata com a diminuição do passo de integração para inicialização utilizando a solução analítica

Vemos que o método realmente converge com ordem 1. Analisando a continuidade nas derivadas, observamos:

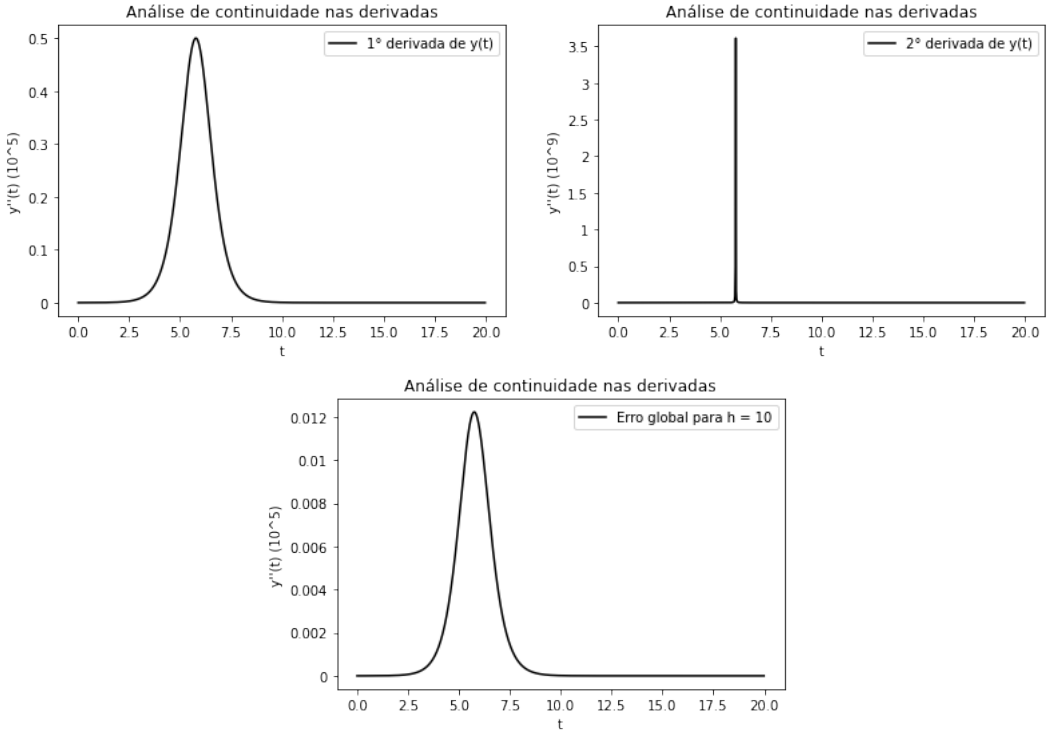

Figura 5: Análise da continuidade das derivadas da solução exata e erro global máximo para n=10 o expoente do número de partições em base 2

Observa-se que a segunda derivada da solução exata é descontínua e o erro global máximo tem um perfil proporcional à contribuição da primeira derivada, indicando que nosso problema de fato deveria assumir convergência de ordem 1.

```
#Metodo de inicializacao usando a solucao analitica
def SAinit(vt_i, vyi_0, dt_0, npass,err,ASolution):
  K = 100000
  r = 2
```

```
#Condicao inicial
vyi_1 = vyi_0.copy()

#Laco de repeticao
for k in range(0,npass-1):
  vyi_1.append((vyi_0[0]*K)/(vyi_0[0] + (K-vyi_0[0])*math.exp(-r*vt_i[k])))

  #Erro global
  err.append(abs(ASolution[len(vyi_1)-1] - vyi_1[len(vyi_1)-1]))

return vyi_1, err
```

IV. Propagação de modos computacionais

Testaremos agora o problema modelo dado por:

$$y'(t) = -sin(t)$$

Utilizamos aqui uma inicialização oscilante do tipo:

$$y_{k+1} = y_k + (-1)^k h$$

Que introduz um erro de amplitude h na aproximação. Testaremos 3 métodos lineares distintos e veremos se há a propagação desses modos.

A. Método escolhido

Utilizando como preditor o método de Euler, com 6 iterações de ponto fixo, foi testado o método escolhido para a inicialização apontada anteriormente.

n	$h = \frac{20-0}{n}$	$\|e(t,h)\|$	$q = \|e(t,2h)/e(t,h)\|$	$log_2(q)$ (Ordem)
64	3.43750e-01	4.02253e-01		
128	1.71875e-01	1.86609e-01		
256	8.59375e-02	8.96279e-02	2.08204e+00	1.05800e+00
512	4.29688e-02	4.38918e-02	2.04202e+00	1.03000e+00
1024	2.14844e-02	2.17152e-02	2.02125e+00	1.01525e+00
2048	1.07422e-02	1.07999e-02	2.01068e+00	1.00769e+00
4096	5.37109e-03	5.38552e-03	2.00536e+00	1.00386e+00
8192	2.68555e-03	2.68915e-03	2.00268e+00	1.00193e+00
16384	1.34277e-03	1.34367e-03	2.00134e+00	1.00097e+00

Tabela IV: Tabela de convergência para o método escolhido com preditor do método de Euler e inicialização acima. Vemos que há convergência de ordem 1

A convergência com ordem 1 era esperada, dada o erro introduzido nos passos iniciais que é da ordem de h. Esperamos observar esse comportamento nos outros dois métodos também. Na figura abaixo vemos que os gráficos das aproximações convergem para o da solução analítica do problema.

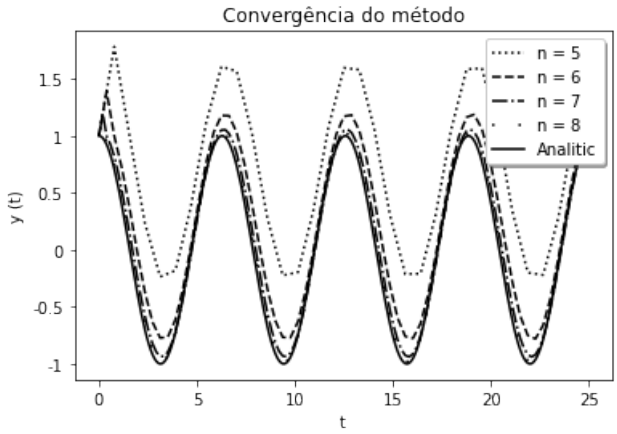

Figura 6: Gráfico indicando a convergência das aproximações para a solução exata do problema com a diminuição do passo de integração. Aqui, n representa o expoente do número de partições na base 2

```
def AlissonMethod2(vt,vy_0, dt, npf, preditor, model,analiticS):
  vy = vy_0.copy()


  #Par metros do metodo
  c = 0.666
  m = 314
  n = 340
  a0 = -c
  a1 = (m/n)*(c - 1)
  a2 = ((n-m)/n)*(c - 1)
  b0 = ((8*n - 3*m)/(6*n))*(c - 1) + 3/2
  b3 = ((-3*m + 4*n)/(6*n))*(c - 1) + 3/2

  C3 = (a1 + 8*a2 + 27)/6 - (9*b3)/2

  err = []
  #Numero de passos
  npass = 3

  err = [0]
  #Chute inicial
  vy,err = Modo(vt, vy, dt, npass,err,analiticS)
  yPontoFixo = []

  for k in range(3,len(vt)):
    pred = preditor(vt, vy, dt, 2, model, k,err,analiticS)
    vy, C3_pred, err = pred

    if(k>3):
      yPf = vy[len(vy)-1] + (C3_pred/(C3_pred - C3))*(yPontoFixo[npf-1] - yPontoFixo[0])
    else:
      yPf = vy[len(vy)-1]

    yPontoFixo = [yPf]
    #iteracoes de ponto fixo
    for s in range(0,npf):
      yPf = -a0*vy[k-3] - a1*vy[k-2] - a2*vy[k-1] + dt*(b0*model(vt[k-3],vy[k-3]) + b3*model(vt[k
          ], yPf))
      if(s>0):
        yPontoFixo.append(yPf + (yPf - yPontoFixo[0])*(C3/(C3_pred - C3)))
    vy[len(vy)-1] = yPontoFixo[npf-1]
    err[len(err)-1] = abs(analiticS[k] - vy[k])

  return np.array([vy, err])
```

B. Leapfrog

n	$h = \frac{20-0}{n}$	$\|e(t,h)\|$	$q = \|e(t,2h)/e(t,h)\|$	$log_2(q)$ (Ordem)
64	3.43750e-01	4.02253e-01		
128	1.71875e-01	1.86609e-01		
256	8.59375e-02	8.96279e-02	2.08204e+00	1.05800e+00
512	4.29688e-02	4.38918e-02	2.04202e+00	1.03000e+00
1024	2.14844e-02	2.17152e-02	2.02125e+00	1.01525e+00
2048	1.07422e-02	1.07999e-02	2.01068e+00	1.00769e+00
4096	5.37109e-03	5.38552e-03	2.00536e+00	1.00386e+00
8192	2.68555e-03	2.68915e-03	2.00268e+00	1.00193e+00
16384	1.34277e-03	1.34367e-03	2.00134e+00	1.00097e+00

Tabela V: Tabela de convergência do problema modelo estudado para o método Leapfrog. Vemos que há convergência de ordem 1

Na tabela acima é possível notar que os erros máximos cometidos e a ordem de convergência possuem os mesmos valores da tabela anterior para o método escolhido. Isto era esperado, dado que estamos utilizando a mesma inicialização para ambos. Vemos novamente que os valores o erro máximo são proporcionais a h.

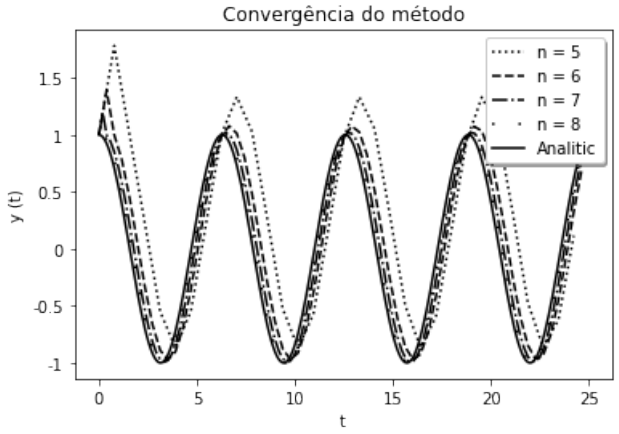

Figura 7: Convergência dos gráficos das soluções aproximadas para o da solução analítica com a diminuição do passo de integração. Aqui, n representa o exoente do número de partições em base 2

```
def Leapfrog(vt,vy_0,dt,model,ASolution):
  vy = vy_0.copy()
  err = [0]
  npass = 3
  err = []
  vy, err = Modo(vt,vy,dt,npass, err, ASolution)
  vf = [model(vt[0], vy[0]),model(vt[1],vy[1]),model(vt[2],vy[2])]
  for k in range(3, len(vt)):
    vy.append(vy[k-1] + dt*vf[k-1] + 0.5*dt*dt*(a(vt[k-1],vy[k-1])))
    vf.append(model(vt[k],vy[k]))
    err.append(abs(vy[k] - ASolution[k]))

  return vy, err
```

C. Adams Bashford 3

n	$h = \frac{20-0}{n}$	$\lvert e(t,h)\rvert$	$q = \lvert e(t,2h)/e(t,h)\rvert$	$log_2(q)$ (Ordem)
64	3.43750e-01	4.37740e-01		
128	1.71875e-01	1.86609e-01		
256	8.59375e-02	8.96279e-02	2.08204e+00	1.05800e+00
512	4.29688e-02	4.38918e-02	2.04202e+00	1.03000e+00
1024	2.14844e-02	2.17152e-02	2.02125e+00	1.01525e+00
2048	1.07422e-02	1.07999e-02	2.01068e+00	1.00769e+00
4096	5.37109e-03	5.38552e-03	2.00536e+00	1.00386e+00
8192	2.68555e-03	2.68915e-03	2.00268e+00	1.00193e+00
16384	1.34277e-03	1.34367e-03	2.00134e+00	1.00097e+00

Tabela VI: Tabela de convergência para o problema modelo estudado usando o método de Adams Bashford de terceira ordem. Vemos que há convergência de primeira ordem

Da tabela acima, podemos constatar que o erro máximo cometido para qualquer método de ordem maior que 1 será o erro da inicialização e teremos convergência de ordem 1, apenas um método de ordem 0 apresentaria um comportamento distinto. Uma inicialização com amplitudes proporcionais à potências maiores de h teria um efeito de uma perturbação. A imagem abaixo mostra a convergência dos gráficos para os da solução analítica.

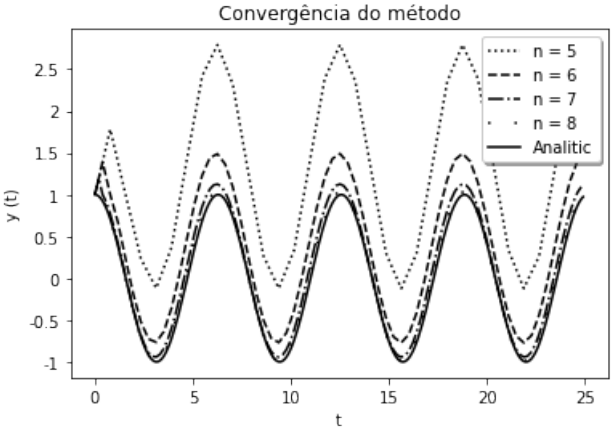

Figura 8: Convergência dos gráficos das soluções aproximadas para o gráfico da solução exata com a diminuição do passo de integração

Fica claro na figura acima que para n=5 temos um erro muito maior comparado aos erros da aproximação pelos métodos anteriores. Isto pode estar ligado a propriedades de estabilidade do método. Na verdade, vemos que para $n = 5$:

n	$h = \frac{20-0}{n}$	$\lvert e(t,h)\rvert$
32	6.87500e-01	1.44378e+00

Tabela VII: Valor do erro máximo cometido quando o expoente do número de partições em base 2 é 5

O erro máximo é maior que o passo de integração em uma ordem de grandeza. Isso pode estar ligado a propriedades como estabilidade absoluta, dado que com a diminuição do passo caímos na zona de proporcionalidade do erro.

```
def AdamsBashford(vt,vy_0, dt,model, analiticS):
  vy = vy_0.copy()
  npass = 3
  err = [0]
  vy,err = Modo(vt, vy, dt, npass,err,analiticS)
  f0 = model(vt[0], vy[0])
  f1 = model(vt[1], vy[1])
  f2 = model(vt[2], vy[2])

  for k in range(3, len(vt)):
    vy.append(vy[len(vy)-1] + dt*(5*f0 - 16*f1 +23*f2)/12)
    f1 = f2
    f0 = f1
    f2 = model(vt[k],vy[k])
    err.append(abs(analiticS[k] - vy[k]))
  return vy,err
```

V. Referências