Método do Shooting Não Linear

Doherty Andrade

www.metodosnumericos.com.br

1 Método do shooting não linear

Script em Python é baseado no algoritmo de Burden-Faires.

O método do shooting para problemas de contorno de segunda ordem não linear

$$PVF(1)\begin{cases} y'' = f(x,y,y'), a \leq x \leq b \\ y(a) = \alpha, \quad \text{e} \\ y(b) = \beta. \end{cases}$$

é semelhante ao método do shooting linear, exceto que a solução para problemas não lineares não pode ser expressa como uma combinação linear de soluções de dois problemas de valor inicial. Em vez disso, aproximamos a solução para o problema de contorno pela utilização de soluções de uma sequência de problemas de valor inicial envolvendo um parâmetro t.

Esses problemas têm a forma

$$PVI(2)\begin{cases} y'' = f(x,y,y'), a \leq x \leq b \\ y(a) = \alpha, \quad \text{e} \\ y'(a) = t. \end{cases}$$

Fazemos isso pela escolha dos parâmetros $t = t_k$ de modo a assegurar que

$$\lim_{k\to\infty} y(b,t_k) = y(b) = \beta,$$

em que $y(x,t_k)$ denota a solução para o problema de valor inicial (2) com $t = t_k$ e $y(x)$ denota a solução problema de contorno (1).

O resultado que assegura a existência de solução para o PFV(2) é dado pelo seguinte teorema:

TEOREMA: Suponha que a função f no PVF (2) seja contínua no conjunto

$$D = \{(x,y,y'); a \leq x \leq b, -\infty < y < \infty, -\infty < y' < \infty\}.$$

e que as derivadas parciais f_x e f_y também sejam contínuas em D. Se

(i) $f_x(x,y,y') > 0$ em D, e

(2i) existir contante $M \geq 0$ tal que

$$|f(x,y,y')| \leq M, \forall (x,y,y') \in D$$

então o PFV (2) tem uma única solução.

Para determinar o parâmetro t_k, suponha que a função f satisfaz às condições do teorema acima. Se $y(x,t)$ denota a solução do PVI (2), em seguida determinamos t como

$$y(b,t) - \beta = 0.$$

Esta é uma equação não-linear e devemos utilizar um dos métodos conhecidos para resolvê-la.

Se usarmos o método da secante, precisaremos de duas aproximações iniciais t_0 e t_1 e gerar os termos restantes da sequência por

$$t_k = t_{k-1} - \frac{(y(b,t_{k-1}) - \beta)\,(t_{k-1} - t_{k-2})}{y(b,t_{k-1}) - y(b,t_{k-2})}, k = 2,3,\ldots.$$

Se usarmos o método de Newton-Raphson para gerar a sequência t_k, precisaremos de uma aproximação inciail t_0 e gerar os demais termos da sequência por

$$\text{(MNR) } t_k = t_{k-1} - \frac{y(b,t_{k-1}) - \beta}{\frac{dy}{dt}(b,t_{k-1})}, k = 1,2,3,\ldots. \tag{1}$$

Note que este enfoque exige o conhecimento da derivada $\frac{dy}{dt}(b,t_{k-1})$ o que é uma dificuldadeuma vez que uma representação explícita de $y(b,t)$ não é conhecida, conhecendo-se apenas valores $y(b,t_0)$, $y(b,t_1)$,...,$y(b,t_{k-1})$.

Para contornar esta dificuldade vamos derivar com relação t a equação

$$y'' = f(x,y,y')$$

do PVI (2). Usando que x e t são independentes, $\frac{\partial x}{\partial t} = 0$, usando as condições iniciais obtemos que

$$\frac{\partial y}{\partial t}(a,t) = 0$$

e

$$\frac{\partial y'}{\partial t}(a,t) = 1,$$

podemos simplificar e obter

$$\frac{\partial y''}{\partial t}(x,t) = \frac{\partial f}{\partial y}(x,y,y')\frac{\partial y}{\partial t}(x,t) + \frac{\partial f}{\partial y'}(x,y,y')\frac{\partial y'}{\partial t}(x,t)$$

com as condições iniciais

$$\frac{\partial y}{\partial t}(a,t) = 0$$

e

$$\frac{\partial y'}{\partial t}(a,t) = 1.$$

Para simplificar a notação, vamos usar $z(x,t)$ para denotar $\frac{\partial y}{\partial t}(x,t)$. Admitindo que a ordem de derivação com relação a x e a t pode ser trocada, a equação acima fica

$$PVI(3)\begin{cases} z''(x,t) = \frac{\partial f}{\partial y}(x,y,y')z(x,t) + \frac{\partial f}{\partial y'}(x,y,y')z'(x,t), a \le x \le b, \\ z(a,t) = 0, \\ z'(a,t) = 1. \end{cases}$$

Assim, o método de Newton-Raphson exige que dois problemas de valor inicial sejam resolvidos a cada iteração, PVI(2) e PVI(3). Então da equação (MNR) fica

$$t_k = t_{k-1} - \frac{y(b, t_{k-1}) - \beta}{z(b, t_{k-1})}. \tag{2}$$

Os problemas de valor inicial não podem, em geral, serem resolvidos explicitamente, portanto, há a necessidade de algum método numérico que pode ser Runge-Kutta de ordem 4 para aproximar ambas as soluções que são necessárias no método de Newton-Raphson.

Observe que cada PVI deve ser escrito como um sistema de EDOS de primeira ordem e então utilizar o Runge-Kutta.

2 Exemplo

Considere o PVF dado a seguir:

$$\begin{cases} y'' = \frac{1}{8}\left(32 + 2x^3 - yy'\right), 1 \leq x \leq 3 \\ y(1) = 17, \\ y(3) = \frac{43}{3} \end{cases}$$

cuja solução exata é $y(x) = x^2 + \frac{16}{x}$.

Vamos aplicar o método do shooting não linear programada a seguir.

```
In [1]: import numpy as np  # biblioteca básica de matemática
        import sympy as sp
        from sympy import Symbol
        import scipy.linalg   # SciPy biblioteca de algebra linear
        import math

In [2]: from numpy import zeros, abs

        def shoot_nonlinear(a,b,alpha, beta, n, tol, M):

            w1 = zeros(n+1)
            w2 = zeros(n+1)
            h = (b-a)/n
            k = 1
            TK = (beta - alpha)/(b - a)

            print("i""    x" "      " "W1""      " "W2")
            while k <= M:

                w1[0] = alpha
                w2[0] = TK
                u1    = 0
                u2    = 1

                for i in range(1,n+1):
```

```
            x = a + (i-1)*h      #step 5

            t = x + 0.5*(h)

            k11 = h*w2[i-1]      #step 6

            k12 = h*f(x,w1[i-1],w2[i-1])
            k21 = h*(w2[i-1] + (1/2)*k12)
            k22 = h*f(t, w1[i-1] + (1/2)*k11, w2[i-1] + (1/2)*k12)
            k31 = h*(w2[i-1] + (1/2)*k22)
            k32 = h*f(t, w1[i-1] + (1/2)*k21, w2[i-1] + (1/2)*k22)
            t   = x + h
            k41 = h*(w2[i-1]+k32)
            k42 = h*f(t, w1[i-1] + k31, w2[i-1] + k32)
            w1[i] = w1[i-1] + (k11 + 2*k21 + 2*k31 + k41)/6
            w2[i] = w2[i-1] + (k12 + 2*k22 + 2*k32 + k42)/6
            kp11 = h*u2
            kp12 = h*(fy(x,w1[i-1],w2[i-1])*u1 + fyp(x,w1[i-1], w2[i-1])*u2)
            t    = x + 0.5*(h)
            kp21 = h*(u2 + (1/2)*kp12)
            kp22 = h*((fy(t, w1[i-1],w2[i-1])*(u1 + (1/2)*kp11)) + fyp(x+h/2, w1[i-1],w2
            kp31 = h*(u2 + (1/2)*kp22)
            kp32 = h*((fy(t, w1[i-1],w2[i-1])*(u1 + (1/2)*kp21)) + fyp(x+h/2, w1[i-1],w2
            t    = x + h
            kp41 = h*(u2 + kp32)
            kp42 = h*(fy(t, w1[i-1], w2[i-1])*(u1+kp31) + fyp(x + h, w1[i-1], w2[i-1])*(
            u1 = u1 + (1/6)*(kp11 + 2*kp21 + 2*kp31 + kp41)
            u2 = u2 + (1/6)*(kp12 + 2*kp22 + 2*kp32 + kp42)


        r = abs(w1[n] - beta)
        #print(r)
        if r < tol:
            for i in range(0,n+1):
                x = a + i*h
                print("%.2f %.2f %.4f %.4f" %(i,x,w1[i],w2[i]))
            return

        TK = TK -(w1[n]-beta)/u1

        k = k+1


    print("Número máximo de iterações foi excedido")
    return
```

Exemplo

```
In [3]: def f(x,y,yp):
```

```
        fx = (1/8)*(32 + 2*x**3 -y*yp)
        return fx

    def fy(xp,z,zp):
        fyy = -(1/8)*(zp)
        return fyy

    def fyp(xpp,zpp,zppp):
        fypp = -(1/8)*(zpp)
        return fypp

    a = 1          # ponto inicial
    b = 3          # ponto final
    alpha = 17     # condição de fronteira
    beta = 43/3    # condição de fronteira
    N = 20         # número de subintervalos
    M = 10         # Número máximo de iterações
    tol = 0.00001 # tolerância


    shoot_nonlinear(a,b,alpha,beta,N,tol,M)
```

```
i    x      W1      W2
0.00 1.00 17.0000 -14.0002
1.00 1.10 15.7555 -11.0233
2.00 1.20 14.7734 -8.7113
3.00 1.30 13.9978 -6.8676
4.00 1.40 13.3886 -5.3634
5.00 1.50 12.9167 -4.1112
6.00 1.60 12.5600 -3.0501
7.00 1.70 12.3018 -2.1364
8.00 1.80 12.1289 -1.3384
9.00 1.90 12.0311 -0.6322
10.00 2.00 12.0000 -0.0001
11.00 2.10 12.0291 0.5718
12.00 2.20 12.1127 1.0942
13.00 2.30 12.2465 1.5754
14.00 2.40 12.4267 2.0222
15.00 2.50 12.6500 2.4400
16.00 2.60 12.9138 2.8331
17.00 2.70 13.2159 3.2052
18.00 2.80 13.5543 3.5592
19.00 2.90 13.9272 3.8975
20.00 3.00 14.3333 4.2222
```

```
In [ ]:
```