Método do shooting linear

Prof. Doherty Andrade

O código Python é baseado no algoritmo de Burden-Faires.

Considere o seguinte problema linear de valor de contorno ou fronteira que desejamos encontrar a solução no intervalo $[a,b]$,

$$(PVF)\begin{cases} y''=p(x)y'+q(x)y+r(x),\ \ x\in(a,b) \\ y(a)=\alpha, \\ y(b)=\beta. \end{cases}$$

Como sabemos se $p(x), q(x)$ e $r(x)$ são contínuas em $[a,b]$ e se $q(x)>0$ para todo $x\in[a,b]$, então o (PVF) acima tem uma única solução.

O método do shooting linear consiste em tomar dois problemas de valor inicial (PVI) associados ao (PVF). Os dois PVI´s são dados por

$$(PVI-1)\begin{cases} y''=p(x)y'+q(x)y+r(x),\ \ x\in(a,b) \\ y(a)=\alpha, \\ y'(a)=0, \end{cases}$$

e

$$(PVI-2)\begin{cases} y''=p(x)y'+q(x)y,\ \ x\in(a,b) \\ y(a)=0, \\ y'(a)=1. \end{cases}$$

Note que nestes problemas, as equações são quase as mesmas, mas as condições iniciais são diferentes.

Os problemas (PVI-1) e (PVI-2) possuem, cada um, uma única solução. Sejam $y_1(x)$ a solução de (PVI-1) e $y_2(x)$ a solução de (PVI-2).

Uma conta fácil mostra que

$$y(x)=y_1(x)+\frac{\beta-y_1(b)}{y_2(b)}y_2(x),\ \ x\in[a,b]$$

é solução do (PVF), desde que $y_2(b)\neq 0$. Como a sua solução é única, então esta é a solução do (PVF).

A expressão acima mostra que a solução $y(x)$ é uma combinação linear entre as duas soluções $y_1(x)$ e $y_2(x)$.

Assim, o método do shooting é baseado na substituição do (PVF) por dois problemas de valor inicial (PVI-1) e (PVI-2).

Usando um método numérico para obter a solução dos (PVI-1) e (PVI-2), teremos as soluções numéricas $(x_k,y_{1,k})$ e $(x_k,y_{2,k})$, respectivamente. Logo, usando a expressão da combinação linear temos que

$$y_k=y_{1,k}+\frac{\beta-y_1(b)}{y_2(b)}y_{2,k},\ \ k=0,\ldots,n.$$

EXEMPLO: Consideremos o PVF

$$\begin{cases} y''=4y'+3y-x,\ \ x\in(0,1) \\ y(0)=2, \\ y(1)=5. \end{cases}$$

Use o método de shooting linear para obter numericamente a sua solução.

Tomemos os dois PVI´s associados

$$(PVI1)\begin{cases} y''=4y'+3y-x,\ \ x\in(0,1) \\ y(0)=2, \\ y'(0)=0. \end{cases} \qquad (PVI2)\begin{cases} y''=4y'+3y,\ \ x\in(0,1) \\ y(0)=0, \\ y'(0)=1. \end{cases}$$

Para resolver numericamente estes dois PVI´s devemos transformar cada um deles em um sistema de EDO´s de primeira ordem. Note que z é apenas uma variável auxiliar.

PVI1: Fazendo $y' = z$ teremos o seguinte sistema

$$\begin{cases} y' = z, \\ z' = F(x,y,z) = 4z + 3y - x, \ x \in (0,1) \\ y(0) = 2, \\ z(0) = 0. \end{cases}$$

Usando o Método de Runge-Kutta de ordem 4 com $h = 0.1$ para resolver este sistema obtemos os seguintes dados:

x_k	y_k	z_k
0.0	2.00000000	0.000
0.10	2.03433760540538833	0.735667694862281896
0.20	2.15959912411865762	1.84803255922480148
0.30	2.42390266340880123	3.56349796703007104
0.40	2.90417333716616755	6.24211871928383833
0.50	3.72315673778426692	10.4566525826116816
0.60	5.07649184172859070	17.1183447548636281
0.70	7.27579887923021040	27.6771132171394712
0.80	10.8172432203556960	44.4400887070362956
0.90	16.4906446961477400	71.0785181243750230
1.00	$\boxed{25.5531193523306364}$	-113.434469030575030

PVI2: Novamente, fazendo $y' = z$ teremos o seguinte sistema

$$\begin{cases} y' = z, \\ z' = F(x,y,z) = 4z + 3y, \ x \in (0,1) \\ y(0) = 0, \\ z(0) = 1. \end{cases}$$

Usando o Método de Runge-Kutta de ordem 4 com $h = 0.1$ para resolver este sistema obtemos os seguintes dados:

x_k	y_k	z_k
0.0	0.00	1.00000
0.10	0.123570239017721614	1.51154227870370983
0.20	0.312484805372823193	2.33056859314248577
0.30	0.605867619049519246	3.63859440281483382
0.40	1.06594747108753918	5.72448994458101446
0.50	1.79172360443169798	9.04796616645809770
0.60	2.94071025308004241	14.3405935524235133
0.70	4.76354051447866400	22.7665622675180722
0.80	7.65903473271118518	36.1785153833488877
$.90$	12.2618264177168932	57.5246363686053286
1.00	$\boxed{19.5818114520269404}$	91.4964947316431960

Das tabelas anteriores obtemos que $y_1(b) \approx y_{1,10} = 25.55311935$ e $y_2(b) \approx y_{2,10} = 19.58181$. Eles estão marcados em um box. Agora usando a equação (???) e as tabelas anteriores obtemos (novamente, z_k é apenas auxiliar), obtemos

x_k	y_k
0.000	2.00000000
0.10	1.904637966
0.200	1.831614274
0.300	1.787982462
0.400	1.785352147
0.500	1.842559064
0.600	1.989914844
0.700	2.275974580
0.800	2.778300788
0.900	3.62060035
1.000	5.00000000

Esta é a solução numérica do problema de valor de fronteira do exemplo.

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
from numpy import zeros, abs
from numpy import abs, sin, log
import matplotlib.pyplot as plt
```

In [2]:

```
# baseado  em Burden-Faires
# Método do Shooting linear com Runge-Kutta de ordem 4

"""
# INPUT : y'' = p(x)y' + q(x)y + r(x) ,  y(a) = alpha, y(b) = beta
         p(x)
         q(x)
         r(x)
         a,b
         alpha,beta
# OUTPUT : approximations w1,i to y(xi)
                         u1 e u2 para o PVI 1
                         v1 e v2 parao PVI 2
"""

def shooting_linear(p,q,r,a,b,alpha, beta, N):

    u1 = np.zeros(N+1)
    u2 = np.zeros(N+1)

    v1 = np.zeros(N+1)
    v2 = np.zeros(N+1)

    W1 = np.zeros(N+1)

    h = (b-a)/N
    u1[0] = alpha
    u2[0] = 0

    v1[0] = 0
    v2[0] = 1

    for i in range(1,N+1):#Runge-Kutta para sistemas
        x = a + (i-1)*h
        k11 = h*u2[i-1]
        k12 = h*(p(x)*u2[i-1]+q(x)*u1[i-1]+r(x))
        k21 = h*(u2[i-1] + (1/2)*k12)
        k22 = h*(p(x + h/2)*(u2[i-1] + (1/2)*k12) + q(x+h/2)*(u1[i-1]+(1/2)*k11)+r(x+h/
2))
        k31 = h*(u2[i-1] + (1/2)*k22)
        k32 = h*(p(x + h/2)*(u2[i-1] + (1/2)*k22) + q(x+h/2)*(u1[i-1]+(1/2)*k21)+r(x+h/
2))
        k41 = h*(u2[i-1]+k32)
        k42 = h*(p(x + h)*(u2[i-1] + k32) + q(x+h)*(u1[i-1]+ k31)+r(x+h))

        u1[i] = u1[i-1] + (k11 + 2*k21 + 2*k31 + k41)/6
        u2[i] = u2[i-1] + (k12 + 2*k22 + 2*k32 + k42)/6

        kp11 = h*v2[i-1]
        kp12 = h*(p(x)*v2[i-1] +q(x)*v1[i-1])
        kp21 = h*(v2[i-1] + (1/2)*kp12)
        kp22 = h*(p(x+h/2)*(v2[i-1]+ (1/2)*kp12) + q(x+h/2)*(v1[i-1]+ (1/2)*kp11))
        kp31 = h*(v2[i-1] + (1/2)*kp22)
        kp32 = h*(p(x+h/2)*(v2[i-1]+(1/2)*kp22)+ q(x+h/2)*(v1[i-1] +(1/2)*kp21))
        kp41 = h*(v2[i-1] + kp32)
        kp42 = h*(p(x+h)*(v2[i-1]+kp32)+ q(x+h)*(v1[i-1] + kp31))

        v1[i] = v1[i-1] + (1/6)*(kp11 + 2*kp21 + 2*kp31 + kp41)
        v2[i] = v2[i-1] + (1/6)*(kp12 + 2*kp22 + 2*kp32 + kp42)
```

```
        i = i+1

    for k in range(0,N+1):#montagem da soluçao
        W1[k] = u1[k] + ((beta-u1[N])*v1[k])/(v1[N])
        k = k +1
    print('As aproximações  são:',W1)
```

In [3]:

```
a = 1         # start point
b = 2         # end point
alpha = 1     # boundary condition
beta  = 2      # boundary condition
N = 99        # number of subintervals

def p(x):
    p = -(2/x)
    return p

def q(x):
    q =  (2/x**2)
    return q

def r(x):
    r =  math.sin(math.log(x))
    return r

shooting_linear(p,q,r,a,b,alpha,beta,N)
```

```
As aproximações  são: [1.         1.00836192 1.01675894 1.02519074 1.03365
699 1.04215742
 1.05069178 1.05925987 1.0678615  1.0764965  1.08516474 1.0938661
 1.10260049 1.11136783 1.12016808 1.12900118 1.13786713 1.1467659
 1.15569752 1.16466199 1.17365936 1.18268966 1.19175295 1.2008493
 1.20997877 1.21914146 1.22833746 1.23756685 1.24682976 1.25612628
 1.26545655 1.27482068 1.2842188  1.29365106 1.30311758 1.31261851
 1.322154   1.3317242  1.34132927 1.35096936 1.36064462 1.37035523
 1.38010135 1.38988315 1.39970079 1.40955444 1.41944429 1.42937049
 1.43933323 1.44933267 1.45936901 1.46944241 1.47955305 1.48970112
 1.49988679 1.51011023 1.52037164 1.53067118 1.54100904 1.5513854
 1.56180044 1.57225434 1.58274727 1.59327941 1.60385095 1.61446205
 1.62511291 1.63580368 1.64653456 1.65730571 1.66811731 1.67896953
 1.68986254 1.70079653 1.71177165 1.72278809 1.733846   1.74494556
 1.75608693 1.76727028 1.77849578 1.7897636  1.80107388 1.8124268
 1.82382252 1.8352612  1.84674299 1.85826806 1.86983655 1.88144863
 1.89310446 1.90480417 1.91654793 1.92833589 1.94016819 1.95204498
 1.96396641 1.97593263 1.98794378 2.         ]
```

Exemplo: Consideremos o PVF

$$\begin{cases} y'' = 4y' + 3y - x, \ x \in (0,1) \\ y(0) = 2, \\ y(1) = 5. \end{cases}$$

Use o método de shooting linear para obter numericamente a sua solução.

In [4]:

```
a = 0          # start point
b = 1          # end point
alpha = 2      # boundary condition
beta  = 5       # boundary condition
N = 99         # number of subintervals

def p(x):
    p = 4
    return p

def q(x):
    q =  3
    return q

def r(x):
    r =  -x
    return r

shooting_linear(p,q,r,a,b,alpha,beta,N)
```

```
As aproximações  são: [2.         1.9894904  1.97916787 1.96903586 1.95909
8   1.94935814
 1.93982035 1.93048894 1.92136844 1.91246366 1.90377964 1.89532174
 1.88709557 1.87910708 1.87136252 1.86386846 1.85663185 1.84966
 1.84296058 1.83654168 1.83041183 1.82457995 1.81905548 1.81384831
 1.80896883 1.80442799 1.80023728 1.79640878 1.79295516 1.78988976
 1.78722658 1.78498031 1.7831664  1.78180104 1.78090126 1.78048492
 1.78057077 1.78117849 1.78232874 1.7840432  1.78634461 1.78925683
 1.79280492 1.79701514 1.80191507 1.80753361 1.8139011  1.82104935
 1.82901175 1.83782329 1.84752068 1.85814243 1.86972892 1.8823225
 1.89596755 1.91071066 1.92660065 1.94368872 1.96202855 1.98167644
 2.00269139 2.0251353  2.04907302 2.0745726  2.10170533 2.13054597
 2.16117291 2.1936683  2.22811829 2.26461318 2.30324763 2.34412087
 2.38733692 2.43300484 2.48123892 2.532159   2.58589067 2.64256558
 2.70232175 2.76530383 2.83166346 2.90155957 2.97515876 3.05263565
 3.13417326 3.21996346 3.31020732 3.40511561 3.50490925 3.60981979
 3.72008992 3.83597403 3.95773872 4.08566345 4.22004112 4.3611787
 4.50939795 4.66503612 4.82844665 5.        ]
```

Fim - ok

In [ ]: