MODELAGEM, CONTROLE E DISCRETIZAÇÕES PARA SISTEMAS DISTRIBUÍDOS

Danilo Borges da Silva (bolsista do PIBIC/ICV),Dr. Alexandro Marinho Oliveira(Orientador, Depto de Matemática - UFPI (Campus Ministro Reis Veloso))

Na atividade proposta pelo meu plano de trabalho desenvolvi algoritmos para solucionar equações diferenciais ordinárias utilizando os métodos de solução numérica de Euler,Euler Melhorado e Runge-Kutta de 4^a ordem, como o auxilio de um software livre com grande poder de processamento chamado Python.

1 Introdução

Sabe-se que nem todas as equações diferenciais são passivas de soluções algébricas, recorrendo aos metódos apreendidos em um curso de Equações Diferenciais Ordinárias(EDOs), para solucionar esta deficiência foram desenvolvidos métodos que se aproximassem do resultado real da equação em determinado ponto, dentre vários métodos citarei um dos mais eficientes métodos para resolução de EDOs conhecido como Runge-Kutta de 4^a ordem, e o algoritmo escrito em Python que descreve este método.

2 Resultados Preliminares

Antes de vermos como se descreve o algoritmo vejamos como é o método. **Método de Runge-Kutta de Quarta Ordem** O **procedimento de Runge-Kutta de quarta ordem** consiste em encontrar constantes apropriadas de tal forma que a fórmula:

$$y_{n+1} = y_n + ak_1 + bk_2 + ck_3 + dk_4$$

, onde

$$k_1 = hf(x_n, y_n)$$
$$k_2 = hf(x_n + \alpha_1 h, y_n + \beta_1 k_1)$$
$$k_3 = hf(x_n + \alpha_2 h, y_n + \beta_2 k_1 + \beta_3 k_2)$$
$$k_3 = hf(x_n + \alpha_3 h, y_n + \beta_4 k_1 + \beta_5 k_2 + \beta_6 k_3)$$

coincida com um polinômio de Taylor de grau quatro. Isso resulta em 11 equações e 13 incógnitas. O conjunto de valores mais comumente usado para as constantes dá valor ao seguinte resultado:

$$y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4),$$
$$k_1 = hf(x_n, y_n)$$
$$k_2 = hf(x_n + \frac{1}{2}h, y_n + \frac{1}{2}k_1) \quad (1)$$
$$k_3 = hf(x_n + \frac{1}{2}h, y_n + \frac{1}{2}k_2)$$
$$k_4 = hf(x_n + h, y_n + k_3) \quad (2)$$

Observemos que as fórmulas em 2, que k_2 depende de k_1, k_3 depende de k_2 e k_4 de k_3. Além disso, k_2 e k_3 envolvem aproximações às inclinações no ponto médio do intervalo entre x_n e x_{n+1}. Vemos que o método segue certos passos para a sua resolução, ou seja, caberá de forma apropriada a um algoritmo de programação.

3 Resultado

Veremos agora como construir o método de Runge-Kutta de 4^a ordem com o python.
Exempo Considere o problema de valor inicial $y' = 2xy$, y(1)=1. Use o método de Runge-Kutta para obter uma aproximação de y(1,5) usando primeiramente h=0,1 e, depois, h=0,05. Usaremos agora uma comparação entre o valor adquirido com o método e o valor real da edo que neste caso é $y = e^{-1}e^{x^2}$, dividindo os resultados adiquiridos com os nomes xn[1], yn[2], vlr exato[3], erro abs[4] e erro rel[5]. Abaixo na figura 1 está o código utilizado para a resolução do problema em python:

```python
import numpy
import matplotlib
from pylab import *

#exemplo utilizando a seguinte equação:
#     Seja o problema de valor inicial y'=2xy, y(1)=1, usando para obter uma aproximação
#     de y(1,5) usando h=0,1 e depois h=0,05.

#1° Determinaremos a função

def f(y,x):
    """
        Função que sera determinada a equação
    """
    return 2*x*y
#2° Determinamos a função do metodo de runge-kutta 2a ordem
def ff(y,x,h):
    """
        Método de runge-kutta 4a ordem
    """
    k1=h*f(x,y)
    k2=h*(f(x+(0.5)*h,y+(0.5)*k1))
    k3=h*(f(x+(0.5)*h,y+(0.5)*k2))
    k4=h*(f(x+h,y+k3))
    return (y+((1/6.0)*(k1+(2*k2)+(2*k3)+k4)))
#3° Faremos a função exata da edo que no caso y=exp(-1)*exp(x^2)
def freal(x):
    return numpy.exp(-1)*numpy.exp(x*x)
#4° Iniciaremos as variáveis
condicao=1.5
h=0.1
ord=[1]
abs=[1]
exato=[1]
i=1
print 'Para h=0.1:'
print '     xn        yn       vlr exato   erro abs  erro rel'
print '%.6f  |  %.6f  | %.6f  | %.6f | %.6f' %(abs[-1] , ord[-1], exato[-1],numpy.absolute((ord[-1]-exato[-1])),numpy.absolute(((((ord[-1]-exato[-1]))/exato[-1]*100))))
#executa as interações do método para o caso h=0.1
while (i<1.49):
    ord.append(ff(ord[-1],abs[-1],h))#adiciona cada ordenada adiquirida com o método na minha lista de ordenadas
    exato.append(freal(abs[-1]+h))   #adiciona cada ordenada exata da função
    abs.append(abs[-1]+h)            #adiciona cada abscissa na minha lista de abscissas
    i=abs[-1]
    print '%.6f  |  %.6f  | %.6f  | %.6f | %.6f' %(abs[-1] , ord[-1], exato[-1],numpy.absolute((ord[-1]-exato[-1])),numpy.absolute(((((ord[-1]-exato[-1]))/exato[-1]*100))))

h=0.05
ord2=[1]
abs2=[1]
exato2=[1]
i=1
print '\nPara h=0.05:'
print '     xn        yn       vlr exato   erro abs  erro rel'
print '%.6f  |  %.6f  | %.6f  | %.6f | %.6f' %(abs2[-1] , ord2[-1], exato2[-1],numpy.absolute((ord2[-1]-exato2[-1])),numpy.absolute(((((ord2[-1]-exato2[-1]))/exato2[-1]*100))))
#executa as interações do método para o caso h=0.05
while (i<1.49):
    ord2.append(ff(ord2[-1],abs2[-1],h))#adiciona cada ordenada adiquirida com o método na minha lista de ordenadas
    exato2.append(freal(abs2[-1]+h))    #adiciona cada ordenada exata da função
    abs2.append(abs2[-1]+h)             #adiciona cada abscissa na minha lista de abscissas
    i=abs2[-1]
    print '%.6f  |  %.6f  | %.6f  | %.6f | %.6f' %(abs2[-1] , ord2[-1], exato2[-1],numpy.absolute((ord2[-1]-exato2[-1])),numpy.absolute(((((ord2[-1]-exato2[-1]))/exato2[-1]*100))))

#na função abaixo serao exibidos 3 funções uma com o h=0.1 com cor vermelha, outra com h=0.05 com cor azul e por fim a função com valor exato na cor preta
plot(abs,ord,'red',abs2,ord2,'blue',abs2,exato2,'black')
grid(True)
show()
```

Figura 1: código exemplo 03

[1] os n valores de x
[2] os n valores de x
[3] valor exato para o determinado xn
[4] erro absoluto
[5] erro relativo

Ao executar o código veremos os resultados que o programa geraram um grafico e no prompt de comando os resultados numéricos em forma de tabela, o gráfico dos pontos adquiridos com o h=0.1 (vermelho), com h=0.05 (azul) e o valor exato (preto), observe bem a aproximação que este método faz no gráfico, e vemos que os resultados colhidos mostram uma exatidão entre três a quatro casas decimais: Vemos que não precisamos recorrer a softwares pagos para realizarmos estas operações, sendo que além do python ser livre ele conta com um série de pacotes que facilitam no desenvolvimento de qualquer tipo de aplicação.

4 Referências

BORGES, Luiz Eduardo. **Python para Desenvolvedores**. Rio de Janeiro: Edição do Autor, 1ªEdição: 2009.

BORGES, Luiz Eduardo. **Python para Desenvolvedores**. Rio de Janeiro: Edição do Autor, 2ªEdição: 2010.

ZILL, Dennis G. **Equações Diferenciais com Aplicações em Modelagem**. São Paulo: Pioneira Thompson Learning, 2003.

Informações sobre Matplotlib:
<http://matplotlib.sourceforge.net/>. Acesso em 04 de agosto de 2010.

Informações sobre Numpy:
<http://numpy.scipy.org/>. Acesso em 04 de agosto de 2010.

Informações sobre Python:
<http://www.python.org/>. Acesso em 04 de agosto de 2010.

nformações sobre Scipy:
<http://www.scipy.org/>. Acesso em 04 de agosto de 2010.