Código en python de los algoritmos práctica 3

Javier Sáez, Laura Gómez, Daniel Pozo, Luis Ortega

1. Ejercicio 1- Euler

```
import numpy as num
from decimal import *
import scipy as sci
from numpy.polynomial import polynomial as pol


def euler(f,a,b,n ,y_0):
    h=Decimal((b-a))/Decimal(n)
    vals = []
    vals.append(y_0)
    print ("Indice\t | t |  Aproximado(u) ")
    print("0\t | 0  |\t"+str(y_0))

    for i in range (0, n-1):
        tj =Decimal(a+(i+1)*h)
        x = vals[i] + h*f(tj,Decimal(vals[i]))
        vals.append(x)
        print(str(i+1)+"\t | "+str(tj)+" |"+"\t"+str(x))
        """print("u_",i+1,"=",x)"""


def f(t,x):
    return -x + t + 1

f0 = 1

euler(f,0,1,10,f0)
```

Vamos a ejecutarlo con la función $f = -x + t + 1$ En el intervalo $[0, 1]$ y con $n = 10$. El resultado es:

```
[fjsaezm@fjsaezm practica3]$ python2 1.py
Indice    |  t  |  Aproximado(u)
0         |  0  |       1
1         | 0.1 |       1.01
2         | 0.2 |       1.029
3         | 0.3 |       1.0561
4         | 0.4 |       1.09049
5         | 0.5 |       1.131441
6         | 0.6 |       1.1782969
7         | 0.7 |       1.23046721
8         | 0.8 |       1.287420489
9         | 0.9 |       1.3486784401
```

2. Ejercicio 2 - Taylor de orden r

```
 # -*- coding: utf-8 -*-
import numpy
import itertools as it
from math import factorial
from sympy import symbols, diff, log
def genf(r, f, t ,y, t_s, y_s):
    faux = f
    for i in range(1, r):
        faux = diff(faux, t_s) + diff(faux, y_s)*f
        yield faux.subs(t_s,t).subs(y_s,y).evalf()

def T(t,y,h,f,r, t_s, y_s):

    suma = f.subs(t_s,t).subs(y_s,y).evalf()
    for i,f_i in enumerate(genf(r, f, t ,y, t_s, y_s)):
        suma += ((h**(i+1))/(factorial(i+2)))*f_i

    return suma

def taylor(f,a,b,n,y_0,r, t_s, y_s):
    uj = y_0
    h = float(b - a)/float(n)
    tj = float(a)
```

```
    print ("Indice\t | t |  Aproximado(u) \t| Real(y) \t\t| Error")
    for i in range (1, n+2):
        yield uj
        valor=func_y.subs(t,tj).evalf()
        print (str(i)+"\t | "+str(tj)+"
            |"+str(uj)+"\t|"+str(valor)+" \t| "+str(valor-uj))
        uj = uj + h*T(tj, uj, h, f, r, t_s, y_s)
        tj = tj + h


t, y = symbols( 't y', real = True)

f = y**2/(1+t)

func_y= -1/log(t+1)

a = 1
b = 2
n = 10
y_0 = (-1/log(t+1)).subs(t,1).evalf()

print(f)
for i in taylor(f, a, b, n, y_0, 2, t, y):
    i
```

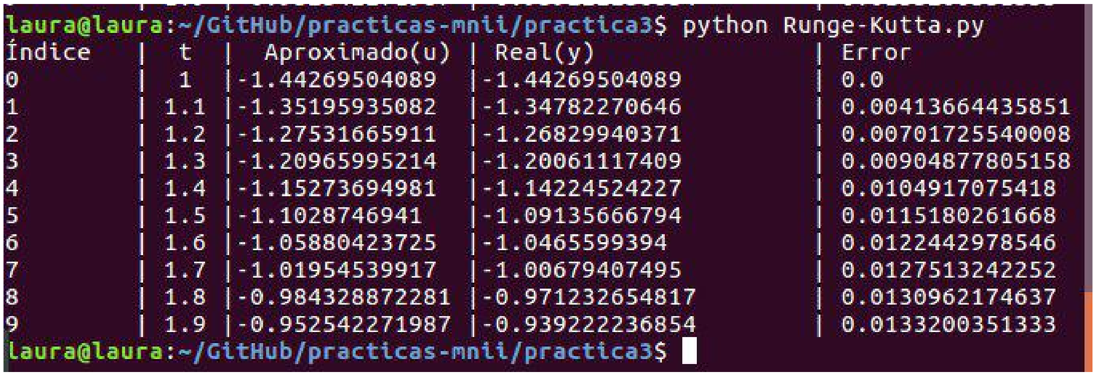
Y si lo ejecutamos con $f = y^2/(1+t)$, y $[a, b] = [1, 2]$, $n = 10$ y $r = 2$ obtenemos:

```
[fjsaezm@fjsaezm practica3]$ python2 Taylor.py
y**2/(t + 1)
Índice    |  t  |  Aproximado(u)          | Real(y)                | Error
1         | 1.0 |-1.44269504088896        |-1.44269504088896       | 0
2         | 1.1 |-1.34873525483283        |-1.34782270646418       | 0.000912548368647625
3         | 1.2 |-1.26973794720399        |-1.26829940370903       | 0.00143854349496197
4         | 1.3 |-1.20234967136413        |-1.20061117409314       | 0.00173849727099218
5         | 1.4 |-1.14414771293982        |-1.14224524227158       | 0.00190247066824467
6         | 1.5 |-1.09333961444766        |-1.09135666793729       | 0.00198294651037179
7         | 1.6 |-1.04857141738952        |-1.04655993939590       | 0.00201147799362111
8         | 1.7 |-1.00880160222078        |-1.00679407494966       | 0.00200752727111775
9         | 1.8 |-0.973216005606330       |-0.971232654817011      | 0.00198335078931944
10        | 1.9 |-0.941169031915503       |-0.939222236853531      | 0.00194679506197271
11        | 2.0 |-0.912142172279551       |-0.910239226626837      | 0.00190294565271365
```

3. Ejercicio 5 - Runge-Kutta

```python
# -*- coding: utf-8 -*-

import math

def RungeKutta(f,a,b,n ,y_0):
    h=float(b-a)/n
    vals = []
    vals.append(y_0)
    valor=y(1)
    print ("Indice\t | t | Aproximado(u) | Real(y) \t\t| Error")
    print ("0\t | 1 |"+str(y_0)+"\t|"+str(valor)+" \t|
        "+str(valor-y_0))
    """
        NOTA: Ahora mismo los valores tj se machacan con cada
            interacion y los
        valores aproximados u se almacenan en el vector vals.
    """
    for i in range (0, n-1):
        tj =a+(i+1)*h
        Ki = []
        Ki.append(f(tj,vals[i]))
        Ki.append(f(tj+h/2,vals[i]+(h/2)*Ki[0]))
        Ki.append(f(tj+h/2,vals[i]+(h/2)*Ki[1]))
        Ki.append(f(tj+h,vals[i]+h*Ki[2]))
        x = vals[i] + (h/6)*(Ki[0]+2*Ki[1]+2*Ki[2]+Ki[3])
        valor=y(tj)
        vals.append(x)
        print (str(i+1)+"\t | "+str(tj)+"
            |"+str(x)+"\t|"+str(valor)+" \t| "+str(valor-x))


def f(t,x):
    return math.pow(x,2)/(1+t);

def y(t):
    return -1/math.log(t+1)

"""En t=1"""
f0 = -1/(math.log(2))

RungeKutta(f,1,2,10,f0)
```

Si ejecutamos este ejercicio obtenemos:



```
laura@laura:~/GitHub/practicas-mnii/practica3$ python Runge-Kutta.py
Índice    |  t  |  Aproximado(u) | Real(y)                | Error
0         |  1  |-1.44269504089  |-1.44269504089          | 0.0
1         | 1.1 |-1.35195935082  |-1.34782270646          | 0.00413664435851
2         | 1.2 |-1.27531665911  |-1.26829940371          | 0.00701725540008
3         | 1.3 |-1.20965995214  |-1.20061117409          | 0.00904877805158
4         | 1.4 |-1.15273694981  |-1.14224524227          | 0.0104917075418
5         | 1.5 |-1.1028746941   |-1.09135666794          | 0.0115180261668
6         | 1.6 |-1.05880423725  |-1.0465599394           | 0.0122442978546
7         | 1.7 |-1.01954539917  |-1.00679407495          | 0.0127513242252
8         | 1.8 |-0.984328872281 |-0.971232654817         | 0.0130962174637
9         | 1.9 |-0.952542271987 |-0.939222236854         | 0.0133200351333
laura@laura:~/GitHub/practicas-mnii/practica3$
```

4. Ejercicio 7

```
  import numpy as num
import scipy as sci
from decimal import *
from numpy.polynomial import polynomial as pol

def pMedio(f,a,b,n,y_0):
    h=Decimal((b-a))/Decimal(n)
    vals = []
    vals.append(y_0)
    print ("Indice\t | t |  Aproximado(u) ")
    print ("0\t | 0  |\t"+str(y_0))
    vals.append(euler1(f,a,b,n,y_0))
    print ("1\t | "+ str(h) + " |\t"+str(vals[1]))
    for i in range (2, n):
        tj = a+(i*h)
        x = vals[i-2] + Decimal(2*h*f(tj,vals[i-1]))
        vals.append(x)
        print(str(i)+"\t | "+str(tj)+" |\t"+str(x))



def euler1(f,a,b,n,y_0):
    h=Decimal((b-a))/Decimal(n)
    tj = Decimal(a+h)
    x = y_0 + h*f(tj,y_0)
    return x

def y(t,x):
   return -x*t + math.pow(t,2)/2 + t
```

```
def f(t,x):
    return -x + t + 1

f0 = 1

pMedio(f,0,1,10,f0)
```

Si lo ejecutamos con la misma función y otros parámetros del ejercicio 1, obtenemos:

```
[fjsaezm@fjsaezm practica3]$ python2 7.py
Indice    |  t  |  Aproximado(u)
0         |  0  |       1
1         | 0.1 |       1.01
2         | 0.2 |       1.038
3         | 0.3 |       1.0624
4         | 0.4 |       1.10552
5         | 0.5 |       1.141296
6         | 0.6 |       1.1972608
7         | 0.7 |       1.24184384
8         | 0.8 |       1.308892032
9         | 0.9 |       1.3600654336
```

5. Ejercicio 8- Adams-Bashforth

```
# PROGRAMA 8
# -*- coding: utf-8 -*-

from math import fabs, exp
from scipy.interpolate import lagrange
from scipy.integrate import quad

a = 0.0
b = 1.0
n = 10
k = 4

# Funcion f de dos variables
f = lambda t,y: -y + t + 1.0
```

```
# Solucion exacta
sol_exacta = True
y = lambda t: exp(-t) + t

# Lista con las aproximaciones u_{0},..,u_{k-1}
# (en caso de no tener la solucion exacta)
inicial = []

h = (b - a) / n
t = [a + j * h for j in range(n + 1)]
u = [0 for i in range(n + 1)] # Lista "vacia" con n+1 posiciones

def integrate_interpolation_polynomial(j):
   x = []
   y = []
   for i in range(k):
      x.append(t[j-i])
      y.append(f(x[i], u[j-i]))

   poly = lagrange(x,y)
   return quad(poly, t[j], t[j+1])[0]

def adams_bashforth(j):
   if j < k:
      return u[j]

   u_j = adams_bashforth(j-1) +
       integrate_interpolation_polynomial(j-1)
   u[j] = u_j
   return u_j

"""
Main
"""

if sol_exacta:
   for i in range(k):
      u[i] = y(t[i])
else:
   for i in range(k):
      u[i] = inicial[i]

adams_bashforth(n)
print("Las", n, "aproximaciones son: ")
for item in u:
```

```
    print(item)
if sol_exacta:
    print("Los errores son: ")
    for j in range(n + 1):
        print(fabs(u[j] - y(t[j])))
```

Y si lo ejecutamos con $f = -y + t + 1$,$[a, b] = [0, 1]$,$n = 10$, $k = 4$ obtenemos:

```
[fjsaezm@fjsaezm practica3]$ python2 8-adams-bashforth.py
('Las', 10, 'aproximaciones son: ')
1.0
1.00483741804
1.01873075308
1.04081822068
1.07032291996
1.10653547546
1.14881840771
1.19659339344
1.24933815637
1.3065796139
1.36788995796
Los errores son:
0.0
0.0
0.0
0.0
2.87392431164e-06
4.81575091049e-06
6.77161793572e-06
8.08965308807e-06
9.19225663676e-06
9.95416030869e-06
1.05167855848e-05
[fjsaezm@fjsaezm practica3]$
```