|  |  |
| --- | --- |
| def metodoHuen(a,b,h,y0,faprox,Ep,itMax):  #bucle de calculos huen  for i in range(1,len(x)):  y[i] = (y[i-1] + h \* faprox(x[i-1],y[i-1]))  it=0  yit=y[i-1]  # \*\*\*\*\* Corrector metodo Trapecio  while ((m.fabs(y[i]-yit)>Ep) and (it<itMax)):  yit=y[i]  y[i] = (y[i-1] + ((h/2) \* ( faprox(x[i-1],y[i-1]) + faprox(x[i],y[i]) )))  print("X="+str(x[i])+" - Y="+str(y[i])+" - It="+str(it)+" error: ",(m.fabs(y[i]-y[i-1])))  it += 1  e[i]= m.fabs(y[i]-y[i-1])  metodoHuen(1,2,0.1,0,faprox,0.01,100) | import numpy as np  import matplotlib.pyplot as plt  import math as m  import numpy.polynomial.polynomial as poly  def f(x,y):  return(1 + ( x - y )\*\*2)  m.e \*\*(3\*x)  print(y)  m.log(1+(x\*\*2))  m.sin(x)  m.cos(x) |
| def metodoEulerRK4(a,b,h,y0,f, freal):  # \*\*\*\*\* metodo de Euler (RK4) \*\*\*\*\*\*  #bucle de calculos  for i in range(1,len(x)):  k1=f(x[i-1],y[i-1])  k2=f(x[i-1]+(h/2),y[i-1]+(h\*k1)/2)  k3=f(x[i-1]+(h/2),y[i-1]+(h\*k2)/2)  k4=f(x[i-1] + h, y[i-1]+(h\*k3))  y[i] = y[i-1] + ((h/6) \* (k1 + 2\*k2 + 2\*k3 + k4))  e[i]= m.fabs(y[i]-y[i-1])  yreal[i] = freal(x[i],y[i])  el[i]= m.fabs(yreal[i]-y[i])  print("error global",i," ",el[i]) | plt.subplot(1, 2, 1)  plt.title("titulo - H="+str(h))  plt.ylabel('y')  plt.xlabel('x')  plt.plot(x,y, 'bo-', label="Y aprox")  plt.plot(x,yreal, 'ro-', label="Y real")  plt.legend()  plt.grid()  plt.subplot(1, 2, 2)  …  plt.show() |
| AdamsBashford(a,b,h,y0, faprox,freal):  # \*\*\*\*\* metodo de Adams Bashfor con (RK1)  y[1] = (y[0] + h \* faprox(x[0],y[0]))  yreal[1] = freal(x[1],y[1])  e[1]= m.fabs(yreal[1]-y[1])  el[1]= m.fabs(y[1]-y[0])  #bucle de calculos para Adans Bashford  for i in range(2,len(x)):  y[i] = y[i-1] + (h/2) \* (3 \* faprox(x[i-1],y[i-1]) - faprox(x[i-2],y[i-2]))  yreal[i] = freal(x[i],y[i])  e[i]= m.fabs(yreal[i]-y[i])  el[i]= m.fabs(y[i]-y[i-1]) | def metodoEulerRK1(a,b,h,y0, f, freal):  # \*\*\*\*\* metodo de Euler (RK1)  n=(b-a)/h  x = np.arange(a,b+h,h)  y = np.arange(a,b+h,h)  yreal = np.arange(a,b+h,h)  e = np.arange(a,b+h,h)  el = np.arange(a,b+h,h)  y[0] = y0  yreal[0] = y0  el[0] = 0  e[0]= m.fabs(yreal[0]-y[0])  for i in range(1,len(x)):  y[i] = y[i-1] + h \* f(x[i-1],y[i-1])  yreal[i] = freal(x[i],y[i])  e[i]= m.fabs(yreal[i]-y[i])  el[i]= m.fabs(y[i]-y[i-1])    metodoEulerRK1(2,3,0.1,1, f, freal) |

|  |  |
| --- | --- |
| def metodoEulerRK1(a,b,h,y10,y20, f1, f2, freal):  # \*\*\*\*\* metodo de Euler (RK1)  n=(b-a)/h  x = np.arange(a,b+h,h)  y1 = np.arange(a,b+h,h)  y2 = np.arange(a,b+h,h)  yreal = np.arange(a,b+h,h)  e = np.arange(a,b+h,h)  el = np.arange(a,b+h,h)  y1[0] = y10  y2[0] = y20  yreal[0] = y10  el[0] = 0  e[0]= 0  for i in range(1,len(x)):  y1[i] = y1[i-1] + h \* f1(x[i-1],y1[i-1],y2[i-1])  y2[i] = y2[i-1] + h \* f2(x[i-1],y1[i-1],y2[i-1])  yreal[i] = freal(x[i],y2[i])  e[i]= m.fabs(yreal[i]-y2[i])  el[i]= m.fabs(y2[i]-y2[i-1])  print(x)  print(y2)  print(yreal)  print(e)  print(el)  def f1(x,y1,y2):  return y2  def f2(x,y1,y2):  return ((3\*y2)-(2\*y1)+x)    def freal(x,y):  return ((m.exp(x))+(0.5\*x)+0.75)  metodoEulerRK1(0,1,0.25,1.75,1.50, f1, f2, freal) | def metodoEulerRK1(a,b,h,y10,y20, f1, f2, fr1, fr2):  # \*\*\*\*\* metodo de Euler (RK1)  n=(b-a)/h  x = np.arange(a,b+h,h)  y1 = np.arange(a,b+h,h)  y2 = np.arange(a,b+h,h)  yr1 = np.arange(a,b+h,h)  yr2 = np.arange(a,b+h,h)  e1 = np.arange(a,b+h,h)  el1 = np.arange(a,b+h,h)  e2 = np.arange(a,b+h,h)  el2 = np.arange(a,b+h,h)  y1[0] = y10 y2[0] = y20 yr2[0] = y20 yr1[0] = y10  el1[0] = 0 e1[0]= 0 e2[0] = 0 el2[0]= 0  for i in range(1,len(x)):  y1[i] = y1[i-1] + h \* (f1(x[i-1],y1[i-1],y2[i-1]))  y2[i] = y2[i-1] + h \* (f2(x[i-1],y1[i-1],y2[i-1]))  yr1[i] = fr1(x[i],y1[i])  yr2[i] = fr2(x[i],y2[i])  e1[i]= m.fabs(yr1[i]-y1[i])  el1[i]= m.fabs(y1[i]-y1[i-1])  e2[i]= m.fabs(yr2[i]-y2[i])  el2[i]= m.fabs(y2[i]-y2[i-1])  print(x) print(y1) print(yr1) print(e1) print(el1) print(x) print(y2) print(yr2) print(e2) print(el2)  def f1(x,y1,y2):  return ((-500\*y1)+(6880\*y2))  def f2(x,y1,y2):  return ((36\*y1)-(500\*y2))  def fr1(x,y):  return (83\*(m.exp(-2\*x)))  def fr2(x,y):  return (6\*(m.exp(-2\*x)))  metodoEulerRK1(0,1,0.001,83.0,6.0, f1, f2, fr1,fr2) |
| def Ajuste(x,y,g):  #Función que calcula el ajuste.  #Parametros: x,y: mediciones. g: grado  coefs = poly.polyfit(x, y, g)  return poly.polyval(x, coefs)  plt.plot(x, y, 'bo', label="Mediciones")  plt.plot(x, Ajuste(x,y,1), 'ro-', label="Ajuste") | x = np.array([0.0, 2.0, 3.0, 5.0]) #medición de x  y = np.array([-1.0, 0.0, 2.0, 1.0]) #medición de y  st = np.array([-1.0, 0.0, 2.0, 1.0]) #medición de y  sr = np.array([-1.0, 0.0, 2.0, 1.0]) #medición de y  f = Ajuste(x,y,2)  for i in range(0,len(x)):  st[i]=(y[i]-np.median(y))\*\*2  for i in range(0,len(x)):  sr[i]=(y[i]-f[i])\*\*2    r= m.sqrt((np.sum(st)-np.sum(sr))/np.sum(st))  r2 = np.corrcoef(x, y)[0,1] #coincide con grado 1 |