Solución de Ecuaciones Diferenciales Ordinarias con Python

Martínez Cerda Mario Antonio

A 12 de Marzo de 2021

1 Introducción

Una ecuación diferencial es una ecuación que involucra una variable dependiente y sus derivadas con respecto a una o más variables independientes. Muchas de las leyes de la naturaleza, en Física, Química, Biología y Astronomía; encuentran la forma más natural de ser expresadas en el lenguaje de las ecuaciones diferenciales. Estas ecuaciones no sólo tienen aplicaciones en las ciencias físicas, sino que también abundan sus aplicaciones en las ciencias aplicadas como ser ingeniería, finanzas y economía.

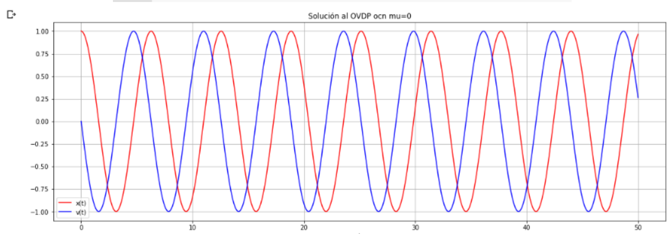
Las ecuaciones diferenciales ordinarias se llaman así, debido a que todas las derivadas involucradas son tomadas con respecto a una única y misma variable independiente.
A lo largo del documento veremos la solución a EDO´s con Python, para fortalecer y mejorar estos cálculos gracias a la programación, que serán de gran utilidad, para problemas futuros.

2 Ejercicios y evidencias

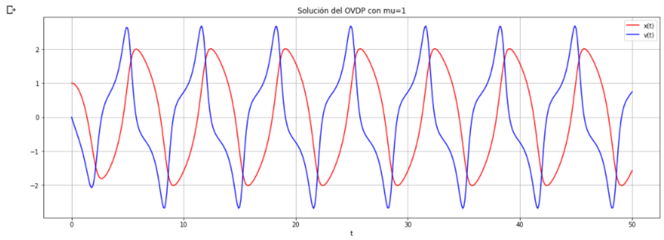
2.1 Ejercicio 1

Resuelva la ecuación diferencial del oscilador de Van der Pol
$\frac{d^2x}{dt^2} - \mu(1-x^2)\frac{dx}{dt} + x = 0$
Donde x es la posición y μ es un parámetro de la parte no lineal.
Resuelva el caso para $\mu = 0, 1, 2, 3, 4$ y grafique las soluciones para un tiempo de integración de $t = [0, 50]$.

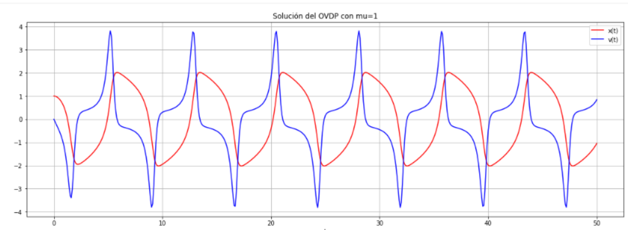
```python
#Oscilador de VAN DER POL
from scipy.integrate import solve_ivp, odeint
#Definiendo la función.
def OVDP(y,t,mu):
  x, v = y
  dydt = [v, mu*(1 - x**2)*v - x]
  return dydt


#Para mu = 0
#Tiempo de integración de 0 a 50.
t = np.linspace(0,50,501)
#Coeficiente mu = 0 para este caso.
mu= 0
#Las condiciones iniciales
t_0 = 0.0
y_0 = [1.0, 0.0]

Solve = odeint( OVDP, y_0, t, args=(mu,))
y1 = Solve [:,0]
y2 = Solve [:,1]

#Graficando
plt.figure(figsize = (18,6))
plt.plot(t,y1,"r",label = "x(t)")
plt.plot(t,y2,"b",label= "v(t)")
plt.legend(loc = "best")
plt.title("Solución al OVDP ocn mu=0")
plt.xlabel("t")
plt.grid()
plt.show()
```



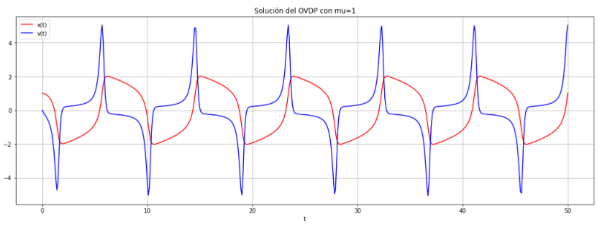

```python
#Para mu = 1
#Tiempo de integración
t = np.linspace( 0, 50, 501 )
#Mu = 1
mu = 1
#COndiciones iniciales
t_0 = 0.0
y_0 = [1.0, 0.0]
Solve = odeint( OVDP, y_0, t, args=(mu,) )
y1 = Solve [:,0]
y2 = Solve [:,1]
#Graficando.
plt.figure( figsize = (18,6) )
plt.plot( t, y1, "r", label = "x(t)" )
plt.plot( t, y2, "b", label = "v(t)" )
plt.legend( loc = "best" )
plt.title( "Solución del OVDP con mu=1" )
plt.xlabel( "t" )

plt.grid()
plt.show()
```



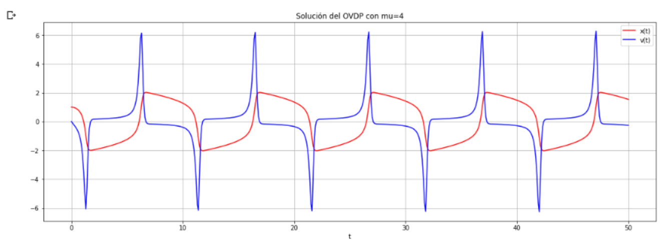

```python
#Para mu = 2
#Tiempo de integración
t = np.linspace( 0, 50, 501 )
#Mu = 1
mu = 2
#COndiciones iniciales
t_0 = 0.0
y_0 = [1.0, 0.0]
Solve = odeint( OVDP, y_0, t, args=(mu,) )
y1 = Solve [:,0]
y2 = Solve [:,1]
#Graficando.
plt.figure( figsize = (18,6) )
plt.plot( t, y1, "r", label = "x(t)" )
plt.plot( t, y2, "b", label = "v(t)" )
plt.legend( loc = "best" )
plt.title( "Solución del OVDP con mu=2" )
plt.xlabel( "t" )

plt.grid()
plt.show()
```




```python
#Para mu = 3
#Tiempo de integración
t = np.linspace( 0, 50, 501 )
#Mu = 1
mu = 3
#COndiciones iniciales
t_0 = 0.0
y_0 = [1.0, 0.0]
Solve = odeint( OVDP, y_0, t, args=(mu,) )
y1 = Solve [:,0]
y2 = Solve [:,1]
#Graficando.
plt.figure( figsize = (18,6) )
plt.plot( t, y1, "r", label = "x(t)" )
plt.plot( t, y2, "b", label = "v(t)" )
plt.legend( loc = "best" )
plt.title( "Solución del OVDP con mu=3" )
plt.xlabel( "t" )

plt.grid()
plt.show()
```




```
#Para mu = 4
#Tiempo de integración
t = np.linspace( 0, 50, 501 )
#Mu = 1
mu = 4
#COndiciones iniciales
t_0 = 0.0
y_0 = [1.0, 0.0]
Solve = odeint( OVDP, y_0, t, args=(mu,) )
y1 = Solve [:,0]
y2 = Solve [:,1]
#Graficando.
plt.figure( figsize = (18,6) )
plt.plot( t, y1, "r", label = "x(t)" )
plt.plot( t, y2, "b", label = "v(t)" )
plt.legend( loc = "best" )
plt.title( "Solución del OVDP con mu=4" )
plt.xlabel( "t" )

plt.grid()
plt.show()
```



2.2 Ejercicio 2

Siguiendo con el ejemplo anterior del oscilador de Van de Pol, reproduce la gráfica del plano fase (θ, ω) que aparece en la Wikipedia para distintos valores de μ y se reproduce abajo.

```
#Agregamos el oscilador de VAN DER POL nuevamente.
#Oscilador de VAN DER POL
from scipy.integrate import solve_ivp, odeint
#Definiendo la función.
def OVDP(y,t,mu):
  x, v = y
  dydt = [v, mu*(1 - x**2)*v - x]
  return dydt
```

Primeramente reproduciremos en cada celda para los distintos valores de mu.

```
#Para mu = 0.01

t = np.linspace( -7, 7, 501 )
#Mu = 0.01
mu = 0.01
#COndiciones iniciales
t_0 = 0.0
y_0 = [1.0, 0.0]
Solve = odeint( OVDP, y_0, t, args=(mu,) )
y1_0_01 = Solve [:,0]
y2_0_01 = Solve [:,1]
```

```
#Para mu = 0.1

t = np.linspace( -7, 7, 501 )
#Mu = 0.1
mu = 0.1
#COndiciones iniciales
t_0 = 0.0
y_0 = [1.0, 0.0]
Solve = odeint( OVDP, y_0, t, args=(mu,) )
y1_0_1 = Solve [:,0]
y2_0_1 = Solve [:,1]
```

```
t = np.linspace( -7, 7, 501 )
#Mu = 0.5
mu = 0.5
#COndiciones iniciales
t_0 = 0.0
y_0 = [1.0, 0.0]
Solve = odeint( OVDP, y_0, t, args=(mu,) )
y1_0_5 = Solve [:,0]
y2_0_5 = Solve [:,1]
```

```
t = np.linspace( -7, 7, 501 )
#Mu = 1.0
mu = 1.0
#COndiciones iniciales
t_0 = 0.0
y_0 = [1.0, 0.0]
Solve = odeint( OVDP, y_0, t, args=(mu,) )
y1_1 = Solve [:,0]
y2_1 = Solve [:,1]
```

```
t = np.linspace( -7, 7, 501 )
#Mu = 1.5
mu = 1.5
#COndiciones iniciales
t_0 = 0.0
y_0 = [1.0, 0.0]
Solve = odeint( OVDP, y_0, t, args=(mu,) )
y1_1_5 = Solve [:,0]
y2_1_5 = Solve [:,1]
```

```
[ ]
t = np.linspace( -7, 7, 501 )
#Mu = 2.0
mu = 2.0
#COndiciones iniciales
t_0 = 0.0
y_0 = [1.0, 0.0]
Solve = odeint( OVDP, y_0, t, args=(mu,) )
y1_2 = Solve [:,0]
y2_2 = Solve [:,1]
```

```
[ ]
t = np.linspace( -7, 7, 501 )
#Mu = 2.5
mu = 2.5
#COndiciones iniciales
t_0 = 0.0
y_0 = [1.0, 0.0]
Solve = odeint( OVDP, y_0, t, args=(mu,) )
y1_2_5 = Solve [:,0]
y2_2_5 = Solve [:,1]
```

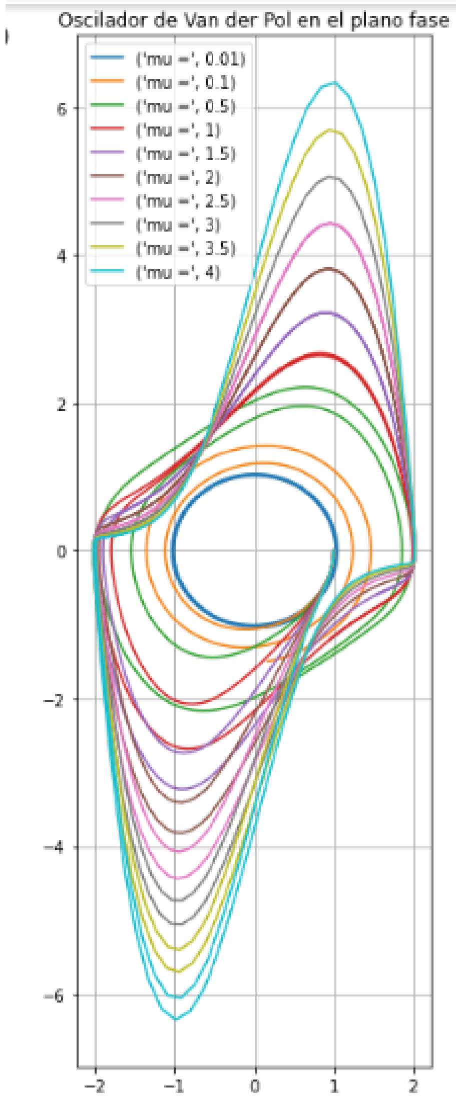
```
[ ]
t = np.linspace( -7, 7, 501 )
#Mu = 3.0
mu = 3.0
#COndiciones iniciales
t_0 = 0.0
y_0 = [1.0, 0.0]
Solve = odeint( OVDP, y_0, t, args=(mu,) )
y1_3 = Solve [:,0]
y2_3 = Solve [:,1]
```

```
t = np.linspace( -7, 7, 501 )
#Mu = 3.5
mu = 3.5
#COndiciones iniciales
t_0 = 0.0
y_0 = [1.0, 0.0]
Solve = odeint( OVDP, y_0, t, args=(mu,) )
y1_3_5 = Solve [:,0]
y2_3_5 = Solve [:,1]
```

```
[ ]
t = np.linspace( -7, 7, 501 )
#Mu = 4.0
mu = 4.0
#COndiciones iniciales
t_0 = 0.0
y_0 = [1.0, 0.0]
Solve = odeint( OVDP, y_0, t, args=(mu,) )
y1_4 = Solve [:,0]
y2_4 = Solve [:,1]
```

Se grafica el plano fase del oscilador:

```
plt.figure(figsize= (4,12))
plt.plot(y1_0_01, y2_0_01, label = ("mu =", 0.01))
plt.plot(y1_0_1, y2_0_1, label = ("mu =",0.1))
plt.plot(y1_0_5, y2_0_5, label = ("mu =",0.5))
plt.plot(y1_1, y2_1, label = ("mu =",1))
plt.plot(y1_1_5, y2_1_5, label = ("mu =",1.5))
plt.plot(y1_2, y2_2, label = ("mu =",2))
plt.plot(y1_2_5, y2_2_5, label = ("mu =",2.5))
plt.plot(y1_3, y2_3, label = ("mu =",3))
plt.plot(y1_3_5, y2_3_5, label = ("mu =",3.5))
plt.plot(y1_4, y2_4, label = ("mu =",4))

plt.legend(loc="best")
plt.title("Oscilador de Van der Pol en el plano fase")
plt.xlabel("t")
plt.grid()
plt.show()
```



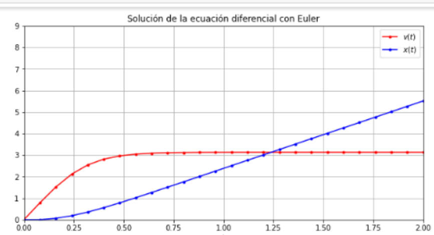
2.3 Ejercicio 3

Encuentre las soluciones de las siguientes Ecuaciones Diferenciales Ordinarias, utilizando los siguientes métodos:
Método de Euler, Método de Runge-Jutta RK4 función scipy.integrate.odeint ó scipy.integrate-solve_ivp.

2.3.1 Ejercicio 3.1

```
#Definimos la ecuación diferencial:
def EDO1(y,t,k,g):
  x, v = y
  dydt = [v,g - k*v**2]
  return dydt
t = np.linspace(0,4,51)
t0 = 0
x0 = 0
v0 = 0

k = 1
g = 9.81
```

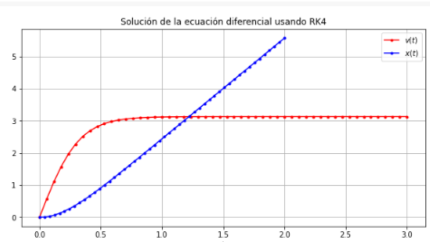
```
#Por el método de Euler:
#Ec1
f = lambda v,t: -k*v**2 + g
v = np.zeros( len( t ) )
v[ 0 ] = v0
for n in range( 0, len( t ) - 1 ):
  v[ n+1 ] = v[ n ] + f (v[ n ], t[ n ] )*( t[ n+1 ] - t[ n ] )
#Ec2
x = np.zeros( len(t) )
x[ 0 ] = x0

for n in range(0,len(t)-1):
  x[ n+1 ] = x[ n ] + v[ n ]*( t[ n+1 ] - t[ n ] )

plt.figure( figsize = ( 10, 5 ) )
plt.plot( t, v, "r.-", t, x, "b.-" )
plt.legend( [ "$v(t)$", "$x(t)$" ] )
plt.title( "Solución de la ecuación diferencial con Euler" )
plt.axis( [ 0, 2, 0, 9 ] )
plt.grid()
plt.xlabel( "t" )
```



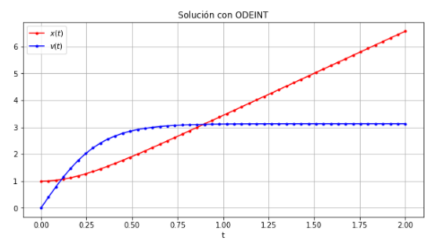

```
#Método Runge-Kutta

def MRK( f, x_0, y_0, h ):

    k_0 = f(x_0, y_0)
    k_1 = f(x_0 + h/2, y_0 + h/2 * k_0)
    k_2 = f(x_0 + h/2, y_0 + h/2 * k_1)
    k_3 = f(x_0 + h, y_0 + h * k_2)

    k = 1/6 * (k_0 + 2.0*k_1 + 2.0*k_2 + k_3)

    x_1 = x_0 + h
    y_1 = y_0 + h * k

    return x_1, y_1
```

```python
a = 0
b = 3
Npts = 51
h = (b-a)/Npts

t0 = 0
x0 = 0
v0 = 0
k = 1
g = 9.81
t=0
v=0

t_values = [t0]
v_values = [v0]

t_values.clear()
v_values.clear()

#Para Ec1:
f = lambda t,v: g - k*v**2

t_values = [t0]
v_values = [v0]

#RK4
for _ in range(Npts):
  t, v = MRK(f, t, v, h)

  t_values.append(t)
  v_values.append(v)
```

```python
#Definimos la función, ya que no hay una ecuacion explicita para la segunda.
def OslVDP( t, y, k, g ):
      dydt = [ y[ 1 ], g - k*( y[ 1 ] )**2 ]
      return dydt


t0 = 0
tmax = 2.0
Npts = 50

#Condiciones iniciales
t_0 = 0.0
y_0 = [ x0, v0 ]

t = np.linspace( t0, tmax, Npts )

abserr = 1.0e-8
relerr = 1.0e-6

sol = solve_ivp( lambda t, y, mu: OslVDP( t, y, k, g ), ( t0, t1 ), y0 = y_0,
                method = "RK45", t_eval = t, args = ( mu, ), rtol = relerr, atol = abserr )

x = sol.y[ 0 ]

#Graficando:
plt.figure( figsize = ( 10, 5 ) )
plt.plot( t_values, v_values, "r.-", label = "$v(t)$" )
plt.plot( t, x, "b.-", label = "$x(t)$" )
plt.legend( loc = "best" )
plt.grid()

plt.title( "Solución de la ecuación diferencial usando RK4" )
plt.xlabel( "t" )
plt.show()
```

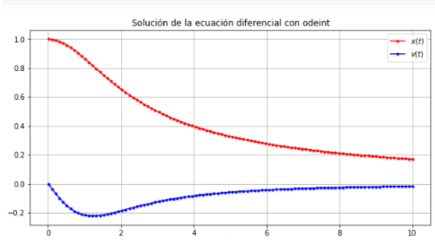


```python
#Usando scipy.integrate.odeint
t0 = 0
tmax = 2.0
Npts = 50
t = np.linspace( t0, tmax, Npts)
#Condiciones iniciales
x_0 = 0.0
a_0 = [1.0, 0.0]

#Error de tolerancia:
abserr = 1.0e-8
relerr = 1.0e-6

Solve = odeint( EDO1, a_0, t, args = ( k, g ), atol = abserr, rtol = relerr )
y1 = Solve[:,0]
y2 = Solve[:,1]

#Graficando:
plt.figure( figsize = ( 10, 5 ) )
plt.plot( t, y1, "r.-", label = "$x(t)$" )
plt.plot( t, y2, "b.-", label = "$v(t)$")
plt.legend( loc = "best" )

plt.title( "Solución con ODEINT" )
plt.xlabel( "t" )
plt.grid()
plt.show()
```



2.3.2 Ejercicio 3.2

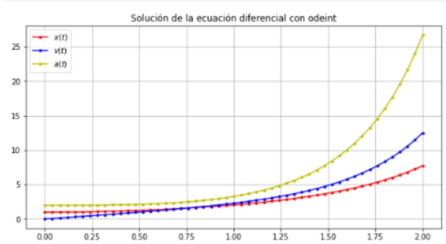
```python
#Vemos que las escuaciones dependen de más de 2 variables,
#por lo tanto usaremos scipy.integrate.odeint
def EDO2( a, x ):
  y, v = a
  dadx = [ v, - ( 2 / x )*v - y**5 ]
  return dadx

x0 = 0.1
y0 = 1.0
v0 = 0
x = np.linspace( 0.01, 10.01, 100 )

#Condinciones iniciales:
x_0 = x0
a_0 = [ y0, v0 ]

#Errores de tolerancia:
abserr = 1.0e-8
relerr = 1.0e-6

Solve = odeint( EDO2, a_0, x, atol = abserr, rtol = relerr )
y1 = Solve[:,0]
y2 = Solve[:,1]

#Graficamos:
plt.figure( figsize = ( 10, 5 ) )
plt.plot( x, y1, "r.-", label = "$x(t)$" )
plt.plot( x, y2, "b.-", label = "$v(t)$")
plt.legend( loc = "best")

plt.title( "Solución de la ecuación diferencial con odeint" )
plt.xlabel( "t" )
plt.grid()
```



2.3.3 Ejercicio 3.3

```python
#Observamos que pasa lo mismo que la anterior, no podemos
#usar el método de Euler ni Runge-Kutta.
def EDO3( b, x ):
  y, v, a = b
  dbdx = [ v, a, ( x - 1 )**2 + y**2 + v - 2]
  return dbdx

x0 = 0.0
y0 = 1.0
v0 = 0.0
a0 = 2.0
x = np.linspace( 0.0, 2.0, 51)

#Condiciones iniciales:
x_0 = x0
a_0 = [ y0, v0, a0 ]

#Errores de tolerancia:
abserr = 1.0e-8
relerr = 1.0e-6

Solve = odeint( EDO3, a_0, x, atol = abserr, rtol = relerr )

y1 = Solve[:,0]
y2 = Solve[:,1]
y3 = Solve[:,2]
```

```python
#Graficamos:
plt.figure( figsize = ( 10, 5 ) )
plt.plot( x, y1, "r.-", label = "$x(t)$" )
plt.plot( x, y2, "b.-", label = "$v(t)$" )
plt.plot( x, y3, "y.-", label = "$a(t)$" )
plt.legend( loc = "best" )

plt.title( "Solución de la ecuación diferencial con odeint" )
plt.xlabel( "t" )
plt.grid()
plt.show()
```



3 Conclusión

Los métodos aprendidos en esta parte del curso son de gran ayuda para la resolución de problemas. Algunos son más aplicables a la hora de eliminar las condiciones por las cuales se rigen los otros métodos, haciendo más sencillo el problema o ejercicio a tratar. La media de complicación de la actividad me pareció un poco difícil debido al cambio que proporciona el aprender un poco más sobre la resolución de ecuaciones. Además que son métodos que jamás había escuchado, por lo que me parecieron bastante interesantes.