Facultad de Matemática, Astronomía, Física y Computación, U.N.C.

Computación

Aula Virtual: https://famaf.aulavirtual.unc.edu.ar/course/view.php?id=747

Resguardo tutoriales: https://www.famaf.unc.edu.ar/∼moreschi/docencia/Computacion/

Tutorial Problemas 5 de la Guía N° 6

Problema 5: Determinación de la constante de un resorte de datos observacionales: Un bloque de masa $m = 1$kg, está sujeto a un resorte y puede deslizarse sobre una superficie horizontal libre de roce. El otro extremo del resorte está sujeto a una pared. Se realizan observaciones de la posición del centro de masa del bloque como función del tiempo. De estas mediciones se debe calcular la constante k del resorte.

En este problema nos enfrentamos a que para funciones que no son polinomios, el ajuste de funciones es más complicado. Intentaremos varias técnicas hasta conseguir un resultado aceptable.

Existe más de una forma de hacer el ajuste de curvas con funciones de PYTHON. En lo que sigue veremos el empleo de dos de ellas con variantes.

a) Se simulará la situación experimental generando aleatoriamente datos alrededor de un movimiento ideal del bloque. Usaremos dos maneras alternativas de describir el movimiento teórico. Para ello use:

```
def f(t,a,om,fi):
    return a*np.cos(om*t+fi)

"""
Recordar que:
A cos(om*t+fi) = A cos(om*t) cos(fi) - A sin(om*t) sin(fi)
= cc cos(om*t) + cs sin(om*t)
tan(fi) = - cs/cc
A = sqrt( cc**2 + cs**2)
"""

def f2(t,cc,cs,om):
    return cc*np.cos(om*t) + cs*np.sin(om*t)
```

b) El intervalo de tiempo, las constantes que determinan el movimiento y la generación de ruido observacional, se calculan de:

```
tmin = 0.
tmax = 14.

# elección de la semilla ----------------------------------------
np.random.seed(12)

# elección de Nrep ----------------------------------------------------
Nt = 1001
t = np.linspace(tmin, tmax, Nt)
A0 = 5. # generamos datos con este valor ideal de A_0, om0 y fi0
om0 = 3.

cc = 4.
```

```
cs = 3.
fi0 = np.arctan(-cs/cc)

ancho = 2.

ya = f(t,A0,om0,fi0)
y = ya + np.random.normal(size=t.size,loc=0.,scale=ancho)

np.random.seed(12)
yb = f2(t,cc,cs,om0)
y2 = yb + np.random.normal(size=t.size,loc=0.,scale=ancho)
```

c) Genere un gráfico que muestre los datos observacionales.

d) Realizaremos varios ajustes con la función curve_fit de la librería scipy.optimize. En este caso usaremos:

```
popt1, pcov1 = curve_fit(f, t, y)
print('popt1 =',popt1)
print('pcov1 =',pcov1)

popt2, pcov2 = curve_fit(f2, t, yb)
print('popt2 =',popt2)
print('pcov2 =',pcov2)
```

e) Genere un gráfico mostrando estos ajustes y además fijando el rango de los posibles valores como se muestra en estos comandos:

```
plt.plot(t, f(t, *popt1), 'm-', linewidth=5.0,
         label='fit: $A$=%6.4f, om=%6.4f, fi=%6.4f, (ajuste f)'
         % tuple(popt1))

plt.plot(t, f2(t, *popt2), 'c--', linewidth=5.0,
         label='fit: $Ac$=%6.4f, $As$=%6.4f, om=%6.4f, (ajuste f2)'
         % tuple(popt2))

# Se puede forzar la búsqueda entre límites con "bounds":
popt, pcov = curve_fit(f, t, y, bounds=([3.5, 1,-2.3],
                                        [6., 5., 2.14159]))
print('popt=',popt)
plt.plot(t, f(t, *popt), 'r-', linewidth=5.0,
         label='fit: $A$=%6.4f, om=%6.4f, fi=%6.4f, (segundo ajuste f)'
         % tuple(popt))

poptb, pcovb = curve_fit(f2, t, y2, bounds=([2.8,2.7,2.1],
                                            [5.6, 5.6, 4.14159]))
print('poptb=',poptb)
plt.plot(t, f2(t, *poptb), 'g--', linewidth=5.0,
         label='fit: $Ac$=%6.4f, $As$=%6.4f, om=%6.4f, (segundo ajuste f2)'
         % tuple(poptb))
```

f) Realizaremos ahora varios ajustes con la función least_squares de la librería scipy.optimize. En este caso usaremos:

```
t_train = t
y_train = y

# notar que en lo que sigue x denota el arreglo de
# los coeficientes que se desean ajustar
def fun(x, t, y):
    return x[0]*np.cos(x[1]*t+x[2]) - y

# propuesta de valores iniciales
x0 = np.array([3.5, 2.1, 0.])
print('   x0=',x0)

print()
print('       ejecuto: res_lsq = least_squares(fun, \
x0, args=(t_train, y_train), method="lm")')
res_lsq = least_squares(fun, x0, args=(t_train, y_train), method='lm')

print(' valores calculados:')
print('          res_lsq.x=',res_lsq.x)

print()
print(' otras dos formas usando valores para "loss":')

#res_soft_l1 = least_squares(fun, x0, loss='soft_l1', f_scale=0.1,
res_soft_l1 = least_squares(fun, x0, loss='huber', f_scale=0.1,
args=(t_train, y_train))
print('      res_soft_l1.x=',res_soft_l1.x)
print()
#res_log = least_squares(fun, x0, loss='cauchy', f_scale=0.1,
res_log = least_squares(fun, x0, loss='arctan', f_scale=0.1,
args=(t_train, y_train))
print('          res_log.x=',res_log.x)
print()
```

g) Genere un gráfico mostrando estos ajustes y además fijando el rango de los posibles valores como se muestra en estos comandos:

```
y_lsq = f(t, *res_lsq.x)
y_soft_l1 = f(t, *res_soft_l1.x)
y_log = f(t, *res_log.x)

plt.figure( figsize=(10, 7.5) )
plt.title('Observaciones + ajuste (f, least_squares) para el oscilador, '
          +str(Nt)+' puntos')
plt.plot(t_train, y_train, 'o', label='observaciones')
plt.plot(t, y_lsq, linewidth=5.0,
         label='fit: $A$=%6.4f, om=%6.4f, fi=%6.4f, linear loss, \
         con: method="lm"'
         % tuple(res_lsq.x))
plt.plot(t, y_soft_l1, linewidth=5.0,
         label='fit: $A$=%6.4f, om=%6.4f, fi=%6.4f, huber loss'
         % tuple(res_soft_l1.x))
plt.plot(t, y_log, linewidth=5.0,
         label='fit: $A$=%6.4f, om=%6.4f, fi=%6.4f, arctan loss'
         % tuple(res_log.x))
```

```
plt.plot(t, ya, 'k', linewidth=1.5, label='exacta')
plt.xlabel("t")
plt.ylabel("y")
plt.grid()
plt.legend(loc="best")
plt.savefig('graficos/p5-least_squares-observaciones-oscilador-'
            +str(Nt)+'.png', dpi=100)
plt.show()
```

h) Elija los valores calculados con alguno de los métodos y calcule la constante k del resorte.

i) Corra el programa nuevamente con `Nt = 1000` y realice los ajustes nuevamente.

Tutorial:

- Guarde en el archivo p5.py las siguientes instrucciones:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit, least_squares, leastsq


def f(t,a,om,fi):
    return a*np.cos(om*t+fi)

"""
Recordar que:
A cos(om*t+fi) = A cos(om*t) cos(fi) - A sin(om*t) sin(fi)
               = cc cos(om*t) + cs sin(om*t)
tan(fi) = - cs/cc
A = sqrt( cc**2 + cs**2)
"""

def f2(t,cc,cs,om):
    return cc*np.cos(om*t) + cs*np.sin(om*t)


# se asume que la unidad de tiempo es el segundo y la de posición el metro
tmin = 0.
tmax = 14.

# elección de la semilla ----------------------------------------
np.random.seed(12)

# elección de Nrep ----------------------------------------------------
Nt = 1001
t = np.linspace(tmin, tmax, Nt)
A0 = 5. # generamos datos con este valor ideal de A_0, om0 y fi0
om0 = 3.

cc = 4.
cs = 3.
fi0 = np.arctan(-cs/cc)

```

```
38 ancho = 2.
39
40 ya = f(t,A0,om0,fi0)
41 y = ya + np.random.normal(size=t.size,loc=0.,scale=ancho)
42
43 np.random.seed(12)
44 yb = f2(t,cc,cs,om0)
45 y2 = yb + np.random.normal(size=t.size,loc=0.,scale=ancho)
46
47 print('t =',t)
48 print('y =',y)
49 print('len(t) =',len(t))
50 print('len(y) =',len(y))
51 print('min(y) =',min(y))
52 print('max(y) =',max(y))
53 print('    A0 =',A0)
54 print('   om0 =',om0)
55 print('   fi0 =',fi0)
56 print(' ancho =',ancho)
57 print('    cc =',cc)
58 print('    cs =',cs)
59
60
61 plt.figure( figsize=(10, 7.5) )
62 plt.title('Observaciones para el oscilador, '+str(Nt)+' puntos')
63 plt.xlabel('t')
64 plt.ylabel('y')
65 plt.grid()
66 plt.plot(t , y,'o-', label='observaciones')
67 plt.legend(loc="best")
68 plt.savefig('graficos/p5-observaciones-oscilador-'+str(Nt)+'.png', dpi=100)
69 plt.show()
70
71 print('--------- Usando: curve_fit --------------------')
72
73 popt1, pcov1 = curve_fit(f, t, y)
74 print('popt1 =',popt1)
75 print('pcov1 =',pcov1)
76
77 popt2, pcov2 = curve_fit(f2, t, yb)
78 print('popt2 =',popt2)
79 print('pcov2 =',pcov2)
80
81
82 plt.figure( figsize=(10, 7.5) )
83 plt.title('Observaciones + ajuste (f y f2, curve_fit) para el oscilador, '
84     +str(Nt)+' puntos')
85 plt.xlabel('t')
86 plt.ylabel('y')
87 plt.grid()
88 plt.plot(t , y,'o', label='observaciones')
89 plt.plot(t, f(t, *popt1), 'm-', linewidth=5.0,
90          label='fit: $A$=%6.4f, om=%6.4f, fi=%6.4f, (ajuste f)'
91          % tuple(popt1))
92
93 plt.plot(t, f2(t, *popt2), 'c--', linewidth=5.0,
```

```
         label='fit: $Ac$=%6.4f, $As$=%6.4f, om=%6.4f, (ajuste f2)'
         % tuple(popt2))

# Se puede forzar la búsqueda entre límites con "bounds":
popt, pcov = curve_fit(f, t, y, bounds=([3.5, 1,-2.3],
                                        [6., 5., 2.14159]))
print('popt=',popt)
plt.plot(t, f(t, *popt), 'r-', linewidth=5.0,
         label='fit: $A$=%6.4f, om=%6.4f, fi=%6.4f, (segundo ajuste f)'
         % tuple(popt))

poptb, pcovb = curve_fit(f2, t, y2, bounds=([2.8,2.7,2.1],
                                            [5.6, 5.6, 4.14159]))
print('poptb=',poptb)
plt.plot(t, f2(t, *poptb), 'g--', linewidth=5.0,
        label='fit: $Ac$=%6.4f, $As$=%6.4f, om=%6.4f, (segundo ajuste f2)'
        % tuple(poptb))

plt.plot(t, ya, 'k', linewidth=1.5, label='exacta')
plt.legend(loc="best")
plt.savefig('graficos/p5-observaciones+ajuste-oscilador-'+str(Nt)+
            '.png', dpi=100)
plt.show()


print('--------- Usando: least_squares --------------------')

t_train = t
y_train = y

# notar que en lo que sigue x denota el arreglo de
# los coeficientes que se desean ajustar
def fun(x, t, y):
     return x[0]*np.cos(x[1]*t+x[2]) - y

# propuesta de valores iniciales
x0 = np.array([3.5, 2.1, 0.])
print('   x0=',x0)

print()
print('       ejecuto: res_lsq = least_squares(fun, \
x0, args=(t_train, y_train), method="lm")')
res_lsq = least_squares(fun, x0, args=(t_train, y_train), method='lm')

print(' valores calculados:')
print('          res_lsq.x=',res_lsq.x)
#print()
print('       res_lsq.cost=',res_lsq.cost)
#print()
print(' res_lsq.optimality=',res_lsq.optimality)

print()
print(' otras dos formas usando valores para "loss":')

#res_soft_l1 = least_squares(fun, x0, loss='soft_l1', f_scale=0.1,
res_soft_l1 = least_squares(fun, x0, loss='huber', f_scale=0.1,
```

```
                                args=(t_train, y_train))
print('      res_soft_l1.x=',res_soft_l1.x)
print()
#res_log = least_squares(fun, x0, loss='cauchy', f_scale=0.1,
res_log = least_squares(fun, x0, loss='arctan', f_scale=0.1,
                        args=(t_train, y_train))
print('          res_log.x=',res_log.x)
print()

"""
   Graficamos las distintas curvas obtenidas.
"""

y_lsq = f(t, *res_lsq.x)
y_soft_l1 = f(t, *res_soft_l1.x)
y_log = f(t, *res_log.x)

plt.figure( figsize=(10, 7.5) )
plt.title('Observaciones + ajuste (f, least_squares) para el oscilador, '
+str(Nt)+' puntos')
plt.plot(t_train, y_train, 'o', label='observaciones')
plt.plot(t, y_lsq, linewidth=5.0,
         label='fit: $A$=%6.4f, om=%6.4f, fi=%6.4f, linear loss, \
con: method="lm"'
         % tuple(res_lsq.x))
plt.plot(t, y_soft_l1, linewidth=5.0,
         label='fit: $A$=%6.4f, om=%6.4f, fi=%6.4f, huber loss'
         % tuple(res_soft_l1.x))
plt.plot(t, y_log, linewidth=5.0,
         label='fit: $A$=%6.4f, om=%6.4f, fi=%6.4f, arctan loss'
         % tuple(res_log.x))
plt.plot(t, ya, 'k', linewidth=1.5, label='exacta')
plt.xlabel("t")
plt.ylabel("y")
plt.grid()
plt.legend(loc="best")
plt.savefig('graficos/p5-least_squares-observaciones-oscilador-'
            +str(Nt)+'.png', dpi=100)
plt.show()

#quit()
```

- Desde la terminal ejecute:

  python3 p5.py

  e interprete el resultado.

  Alternativamente ejecute:

  python3

  y vaya agregando uno a uno los bloques del programa.

- Estudie cada paso del programa y agrege comentarios explicativos.

- Corra el programa nuevamente con `Nt = 1000` y realice los ajustes nuevamente.

Se deberían generar los siguientes gráficos:

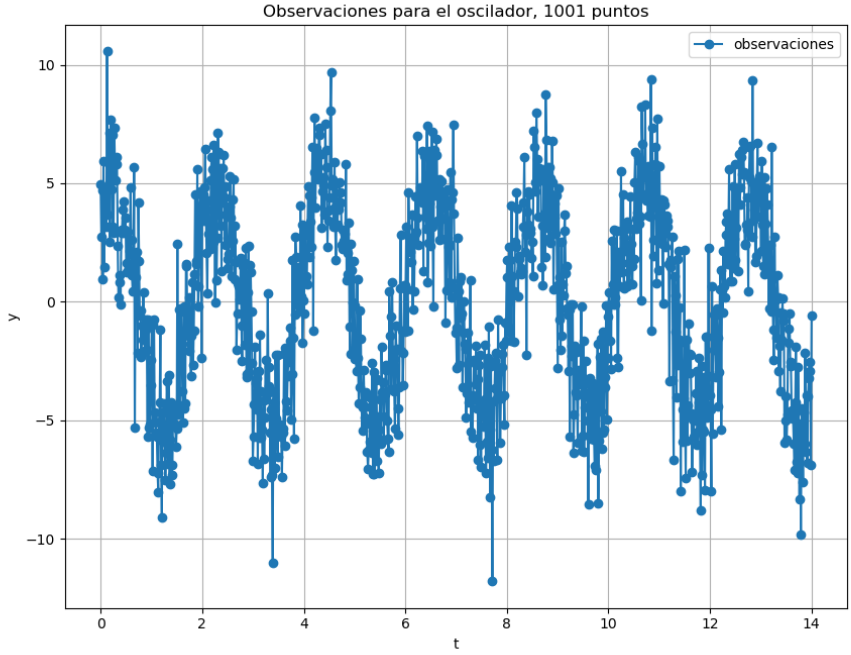

Figura 1: Observaciones para determinar la constante k del oscilador.

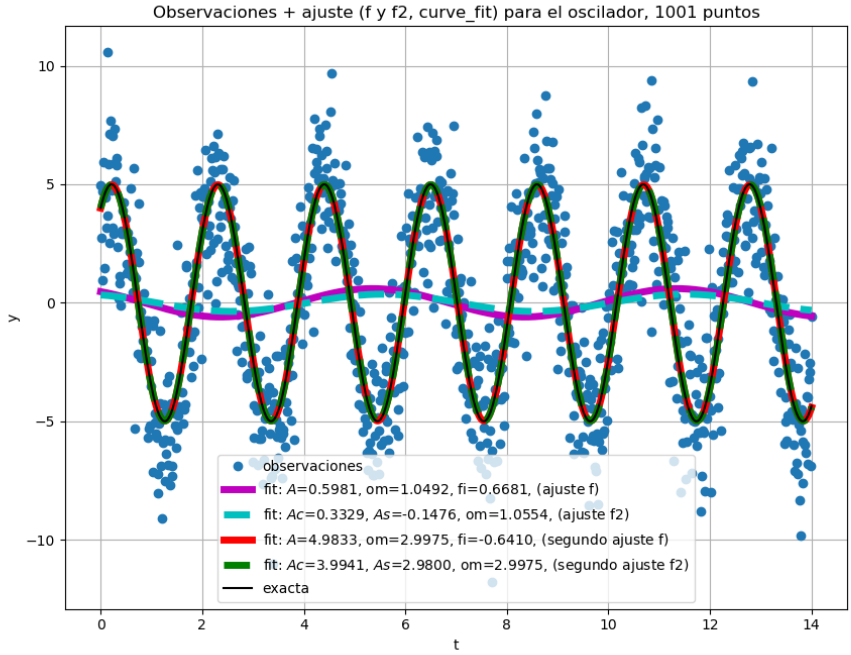

Figura 2: Observaciones y ajustes, usando 'curve_fit', para determinar la constante k.

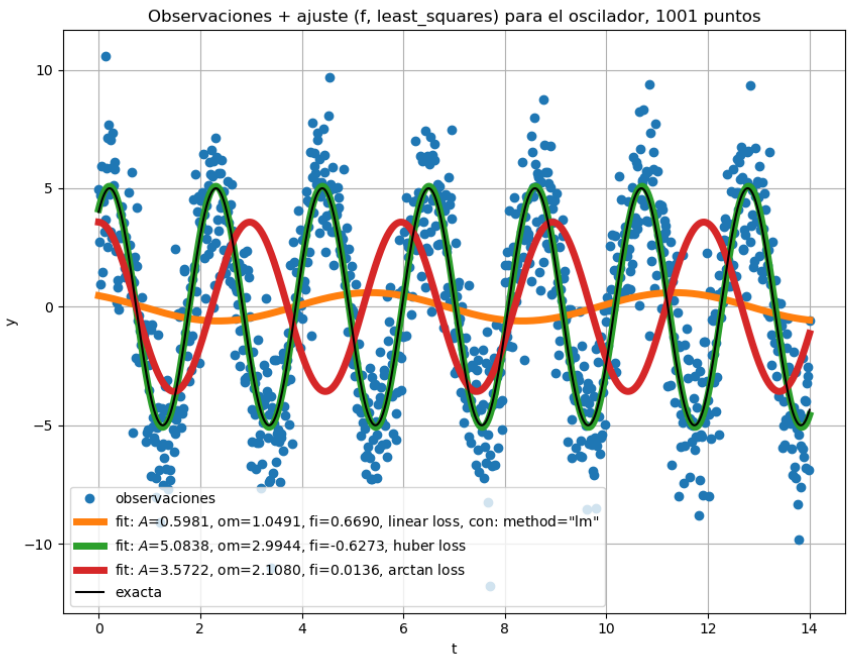

Figura 3: Observaciones y ajustes, usando 'least_squares' para determinar la constante k.