Ajuste de datos

September 4, 2020

1 Tutorial ajuste de datos

1.1 Profesores: Dr. Alejandro Velez Zea, Fabian Castaño

En este Notebook de Jupyter les hare una demostración de como ajustar una función definida a datos experimentales, y luego testear la bondad del ajuste resultante. Para este propósito, realizaremos un ajuste de mínimos cuadrados para una función gaussiana.

1.1.1 1. Generar función de ajuste

Primero definimos la función modelo que trataremos de ajustar a los datos experimentales. En este caso la función será una Gaussiana. Para definir esta función, importaremos de numpy las funciones exp, de exponencial, linspace para generar el dominio de la funcion gaussiana y random para generar números aleatorios. Los argumentos de la gaussiana serán x, que es el dominio, amp, la amplitud, cen, donde está centrada, y finalmente anc, que es el ancho. Estos últimos valores serán los que buscaremos por medio del ajuste.

[2]:
```
from numpy import exp, linspace, random, std, mean

def gaussiana(x, amp, cen,anc):
    return amp*exp(-(x-cen)**2/anc)
```

1.1.2 2. Cargar (o generar) los datos a ajustar.

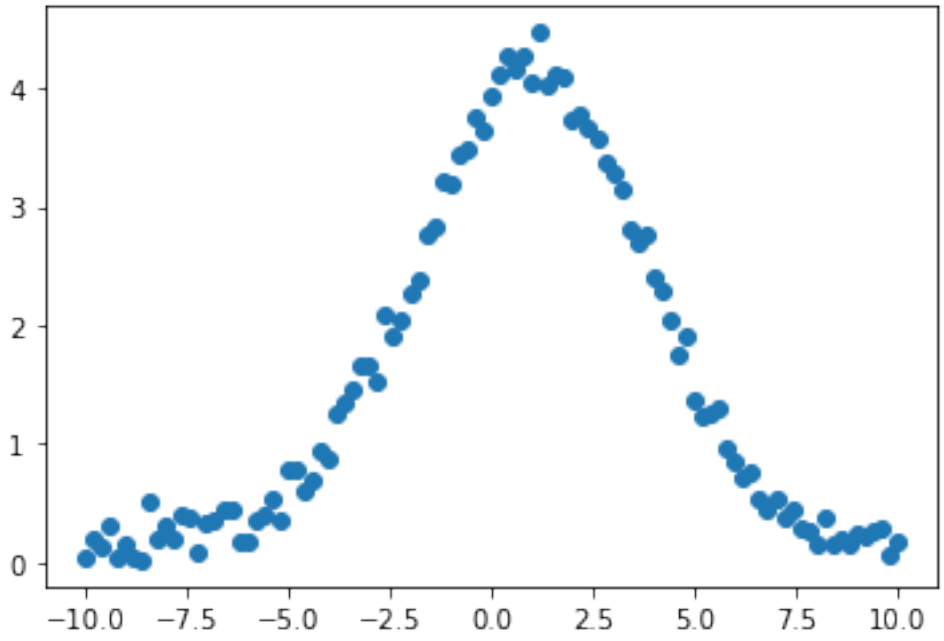
En este caso, no usaremos datos experimentales si no que los generaremos datos por medio de la funcion gaussiana, añadiendo ruido aleatorio por medio de la funcion random.

[3]:
```
x=linspace(-10,10,101)
y=gaussiana(x,4,1,15)+abs(random.normal(0,0.2,x.size))
```

Grafiquemos los datos generados

[4]:
```
import matplotlib.pyplot  as plt
plt.plot(x,y,'-o',linestyle='none',label='Datos Experimentales')
```

[4]:
```
[<matplotlib.lines.Line2D at 0x22ac8fc9940>]
```



1.1.3 3. Hallar parámetros ajustados

Ahora inicializamos la funcion que usaremos para el ajuste, que es llamada scipy.optimize.curve_fit. Para ello creamos un array con valores aproximados para los argumentos de la gausiana que queremos ajustar, en este caso [amp,cen,anc]. Observe los datos experimentales y trate de ofrecer unos valores iniciales plausibles, de lo contrario puede que el ajuste no converge a una solución razonable.

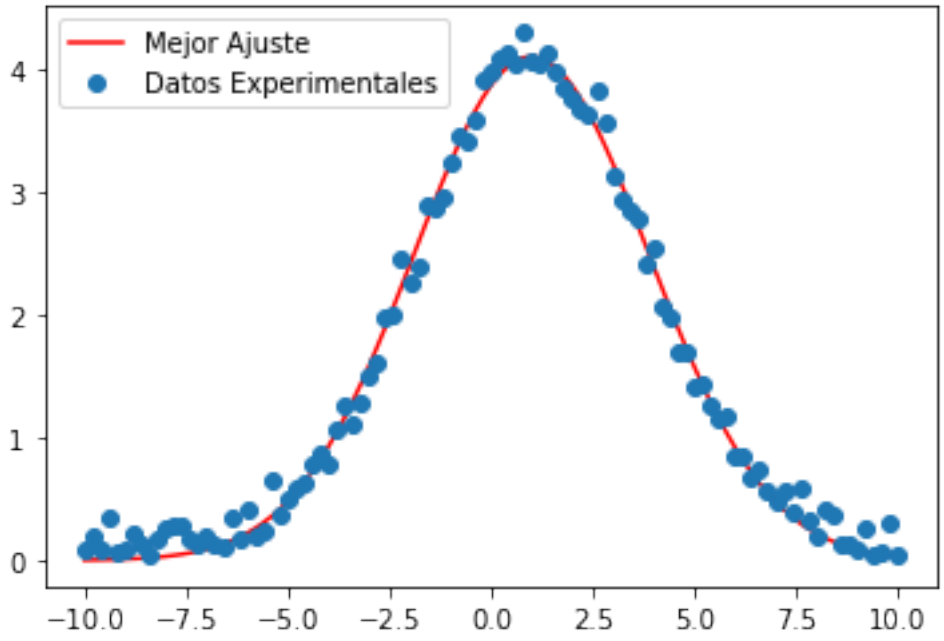
[6]:
```
from scipy.optimize import curve_fit

val_ini = [1,0,1]
mej_val, cov = curve_fit(gaussiana,x,y,val_ini)
print('mej_val: {}'.format(mej_val))
```

```
mej_val: [ 4.13981052  0.97021978 16.9935744 ]
```

Los valores obtenidos de mej_val, representan los argumentos que mejor ajustan la funcion gaussiana a los datos que usamos. Grafiquemos el ajuste contra los datos experimentales.

[7]:
```
yaj=gaussiana(x,mej_val[0],mej_val[1],mej_val[2])
plt.plot(x,yaj,'r',label='Mejor Ajuste')
plt.plot(x,y,'-o',linestyle='none',label='Datos Experimentales')
plt.legend(loc='best')
```

[7]:
```
<matplotlib.legend.Legend at 0x2bffbd32790>
```



1.1.4 4. Determinar la bondad del ajuste.

Para esto usamos el coeficiente de determinación R^2.

[9]:
```
def r_sqr(datexp,dataj):
    return 1-sum((datexp-dataj)**2)/sum((datexp-mean(datexp))**2)

print('R^2 = ', r_sqr(y,yaj))
```

```
R^2 =  0.9915630103875068
```

El valor aqui obtenido es muy cercano a 1, lo que tambien verifica la bondad del ajuste obtenido.

Para cargar un archivo CVS podemos usar pandas

[ ]:
```
import pandas
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit

data=pandas.read_csv(r'Diodo 1N4148-1.csv')

df=pandas.DataFrame(data,columns=['Voltaje (V)','Corriente (mA)'])
a=df.to_numpy()
x=a[:,0]
y=a[:,1]
```