EE2703 week3

G.Sai Krishna Reddy <ee21b049>

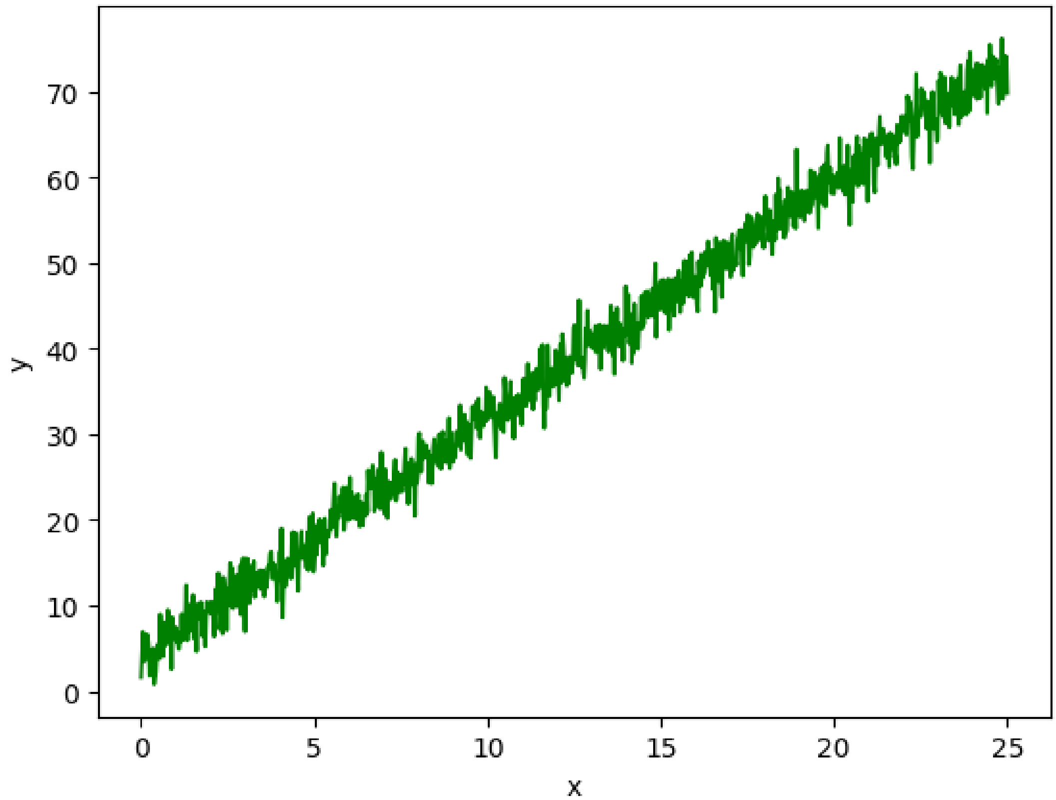
February 18, 2023

1 Libraries:

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import math
%matplotlib inline
from scipy.optimize import curve_fit
```

2 Dataset 1:

[2]:
```
k1=open('dataset1w3.txt',"r")
n1=k1.readlines()
x1=[]
y1=[]
for line in n1:
    p=line.split()
    x1.append(float(p[0]))
    y1.append(float(p[1]))
x1=np.array(x1)
y1=np.array(y1)
plt.xlabel('x')
plt.ylabel('y')
plt.plot(x1,y1,c='g')
plt.show()
```

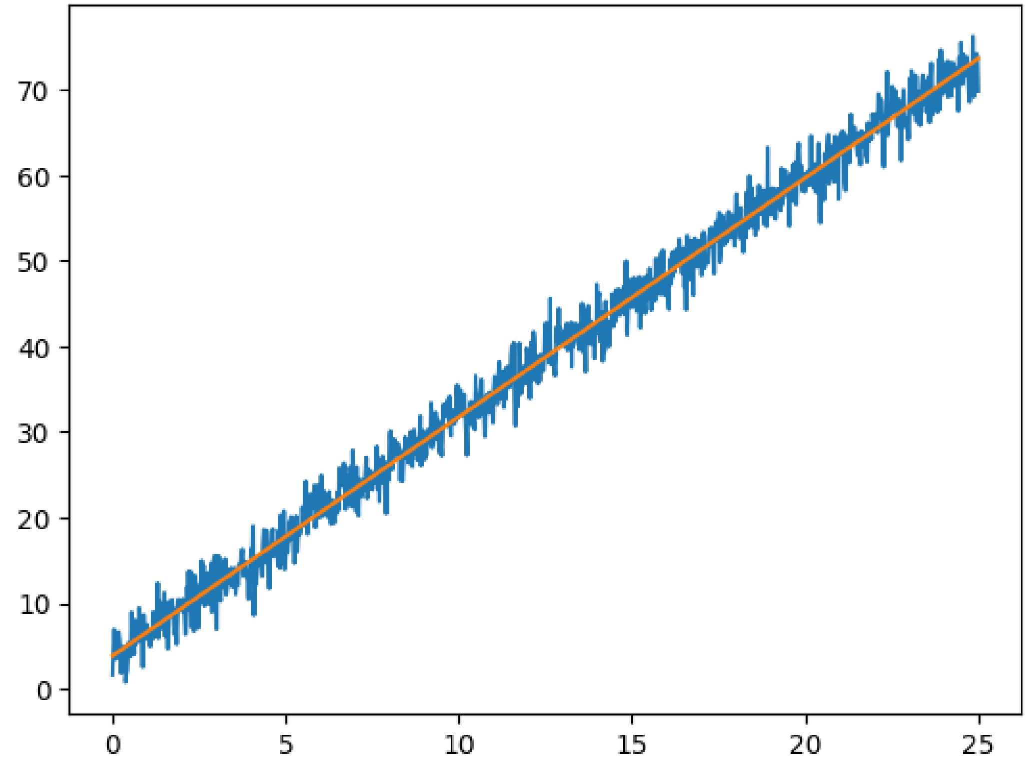


[3]:
```
M = np.column_stack([x1, np.ones(len(x1))])
(p1, p2), _, _, _ = np.linalg.lstsq(M, y1, rcond=None)
print(f"The estimated equation is {p1} t + {p2}")
```

```
The estimated equation is 2.791124245414918 t + 3.848800101430742
```

[4]:
```
def stline(x, m, c):
    return m * x + c
yn=stline(x1,p1,p2)
plt.plot(x1,y1,x1,yn)
```

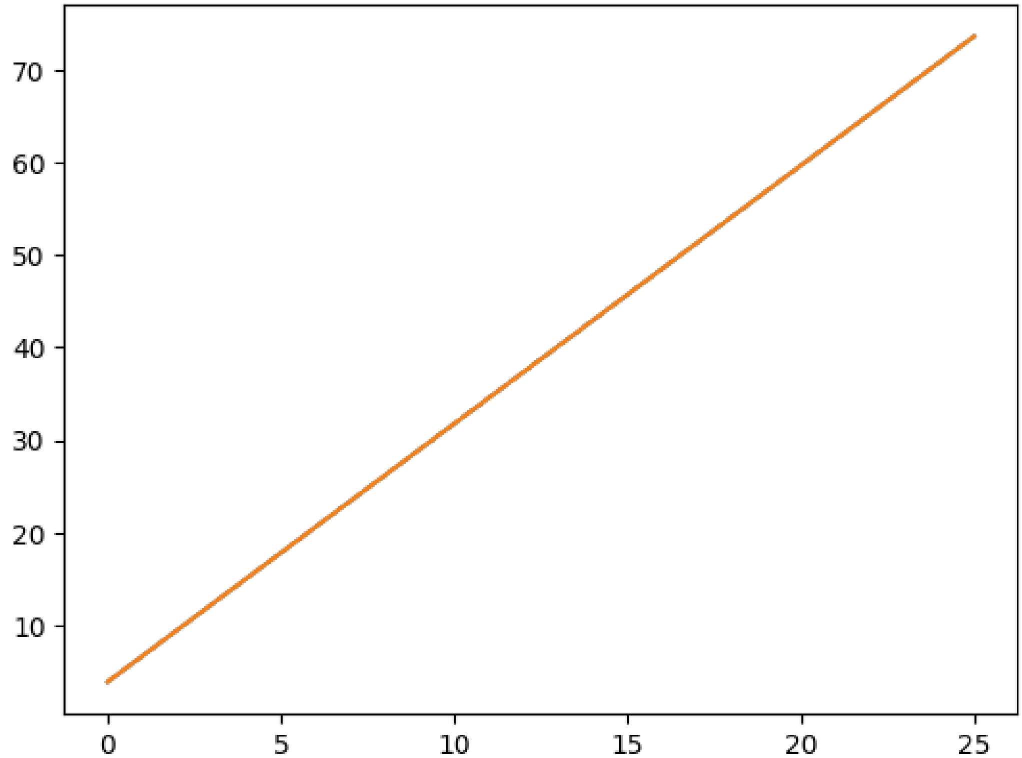
[4]:
```
[<matplotlib.lines.Line2D at 0x7fd4e875f520>,
 <matplotlib.lines.Line2D at 0x7fd4e875f580>]
```



[6]:
```
(m,c),cov=curve_fit(stline,x1,y1)
print(m,c)
ym=stline(x1,m,c)
plt.plot(x1,yn,x1,ym)
```

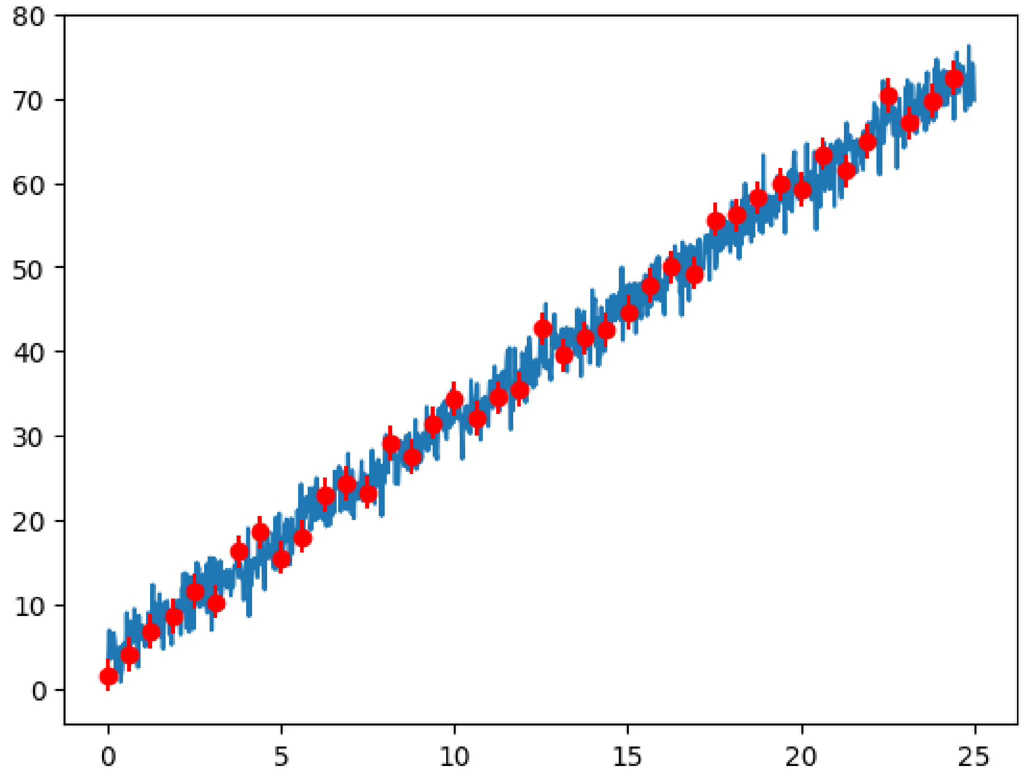
```
2.7911242448201588 3.848800111263445
```

[6]:
```
[<matplotlib.lines.Line2D at 0x7fd4e66ed6a0>,
 <matplotlib.lines.Line2D at 0x7fd4e66ed520>]
```



As we can see from the above plot lstsq and curve_fit alsomt gave same answer(upto 10 decimals), so we can use either of them.

[59]:
```
plt.plot(x1,y1)
plt.errorbar(x1[::25], y1[::25], np.std(yn-y1), fmt='ro')
plt.show()
```




```
1.9958487870009778
```

The size of the errorbars is the third argument in plt.errorbar() and in the above code its mentioned as np.std(), which gives standard deviation.It is calculated as shown below.

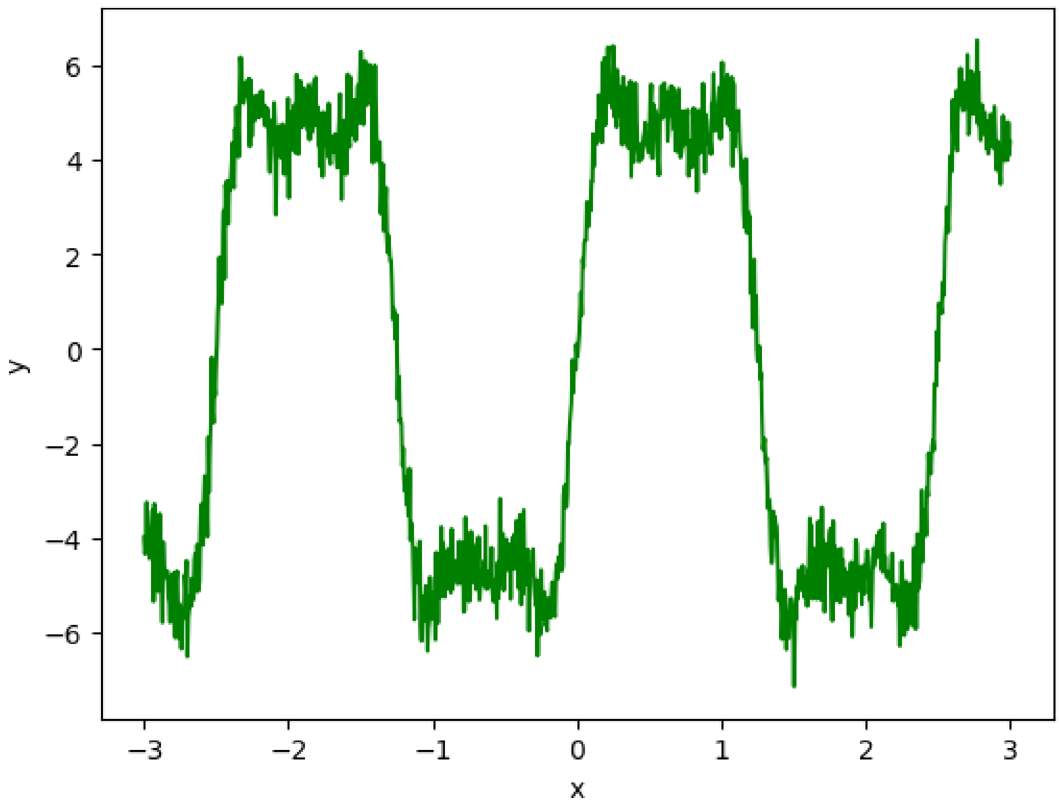
[58]:
```
def stddev(x1):
    j=len(x1)
    s=sum(x1)/j
    sd=[(X-s)**2 for X in x1]
    dev=math.sqrt(sum(sd)/j)
    return dev
print(stddev(yn-y1))
```

```
1.9958487870009785
```

3 Dataset 2:

[7]:
```
k1=open('dataset2w3.txt',"r")
n1=k1.readlines()
x1=[]
y1=[]
for line in n1:
```

```
    p=line.split()
    x1.append(float(p[0]))
    y1.append(float(p[1]))
x1=np.array(x1)
y1=np.array(y1)
plt.xlabel('x')
plt.ylabel('y')
plt.plot(x1,y1,c='g')
plt.show()
```



For using curve_fit, we need to know the function and it estimates any variables in the function.So by observing graohs of fourier transforms of sum of sin waves, we can conclude the given plot is sum of 3 sine harmonics.And to determine which harmonics, I tried plotting different harmonics and 1,3,5 matches the given plot.Then I used curve_fit to estimate frequency and amplitudes.And non linear fitting is better since the plot is non linear.

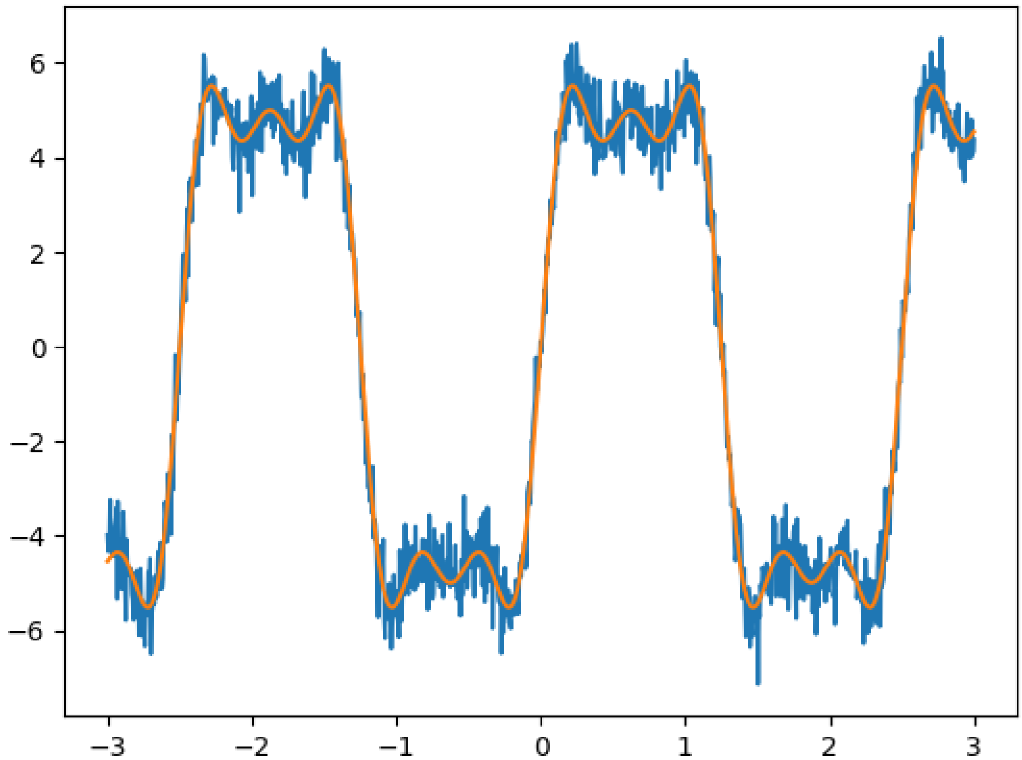
[8]:
```
def f(t,a1,a2,a3,p):
    return a1 * np.sin(2 * np.pi * p * t)+a2 * np.sin(2 * np.pi * p*3 * t)+a3 *␣
 ↪np.sin(2 * np.pi * p*5 * t)

(a1,a2,a3,p),cov = curve_fit(f, x1, y1,p0=(1,1,1,0.35))
```

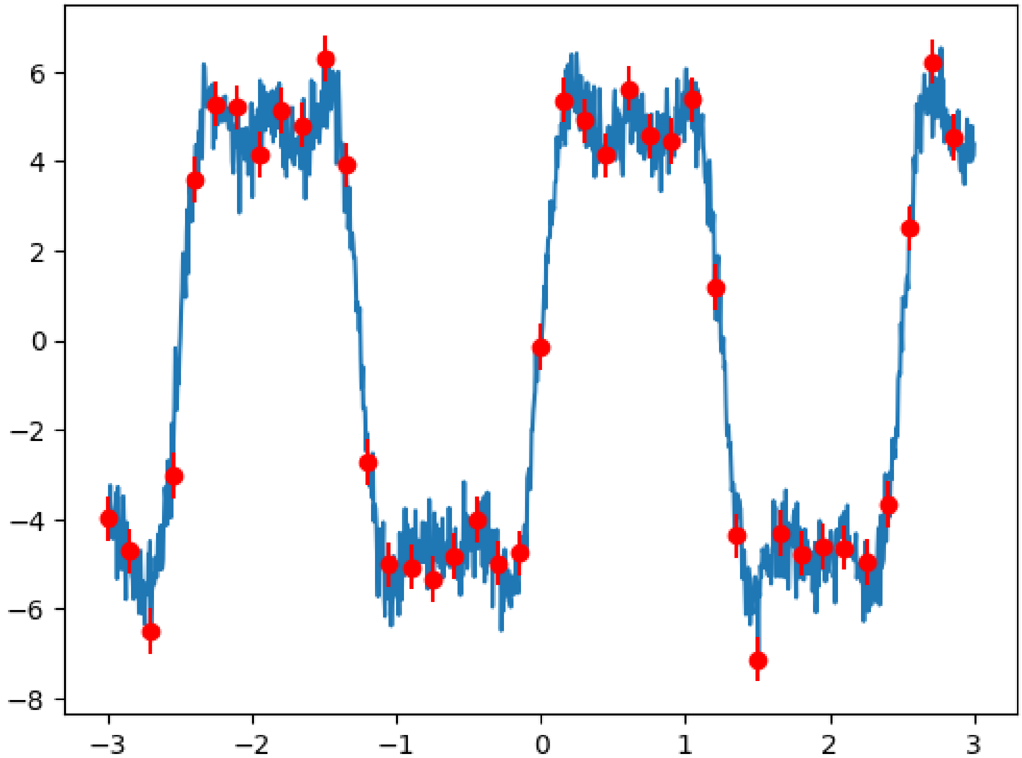
```
print(a1,a2,a3,p)
plt.plot(x1,y1)
y2=f(x1,a1,a2,a3,p)
plt.plot(x1,y2)
```

```
6.011120030226662 2.001458608426161 0.9809070269021316 0.3999141230975358
```

[8]:
```
[<matplotlib.lines.Line2D at 0x7fd4e666bf70>]
```

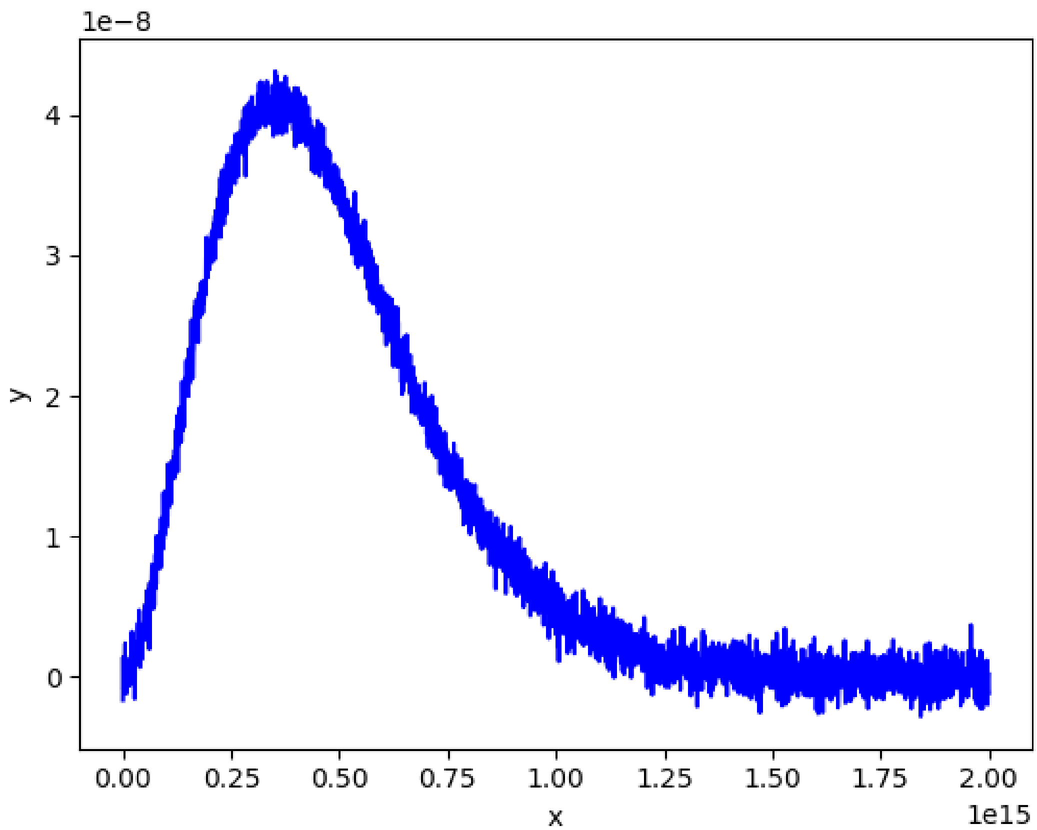


[23]:
```
plt.plot(x1,y1)
plt.errorbar(x1[::25], y1[::25], np.std(y2-y1), fmt='ro')
plt.show()
```



4 Dataset 3:

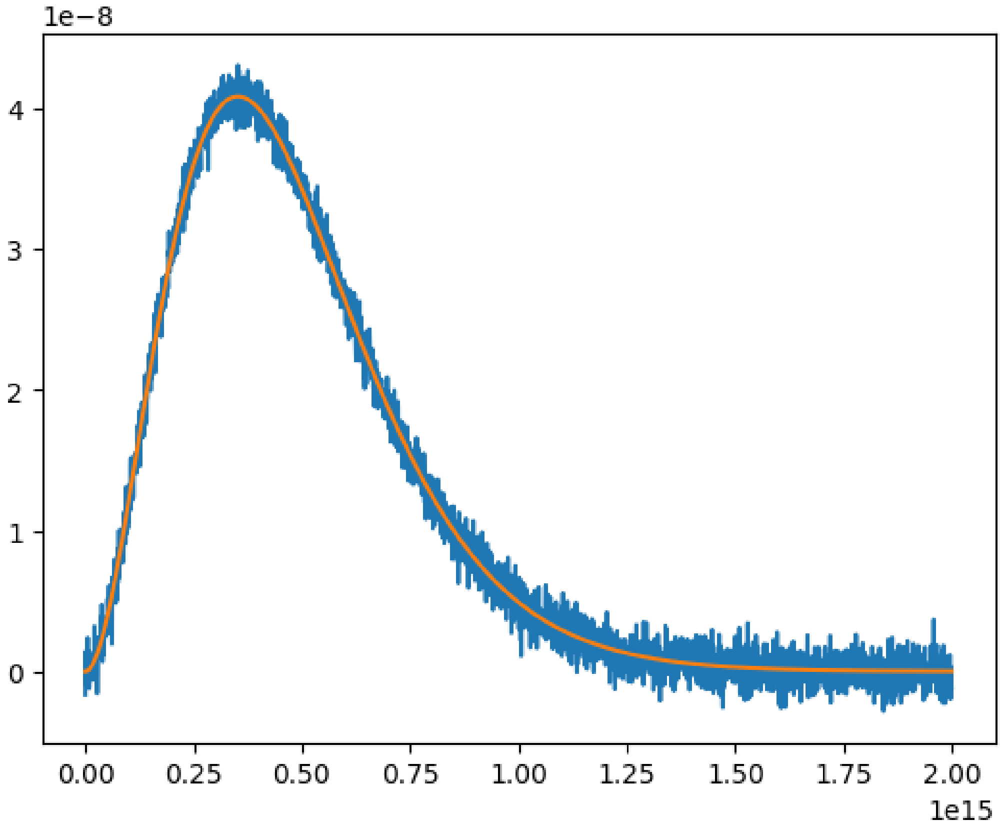
[10]:
```
k1=open('dataset3w3.txt',"r")
n1=k1.readlines()
x1=[]
y1=[]
for line in n1:
    p=line.split()
    x1.append(float(p[0]))
    y1.append(float(p[1]))
x1=np.array(x1)
y1=np.array(y1)
plt.xlabel('x')
plt.ylabel('y')
plt.plot(x1,y1,c='b')
plt.show()
```



[11]:
```
def planck_law(x,t,h):
    c=3.0e8
    k=1.38e-23
    #x=c/x
    return (2*h*(x**3))/((c**2)*(np.exp(h*x/(k*t))-1))
(t,h),pcov=curve_fit(planck_law,x1,y1,p0=[1000,1e-34])
print(f"The value of planck's constant is {h} and the temperature at which␣
 ↪these observations are taken is {t}k")
y3=planck_law(x1,t,h)
plt.plot(x1,y1)
plt.plot(x1,y3)
```

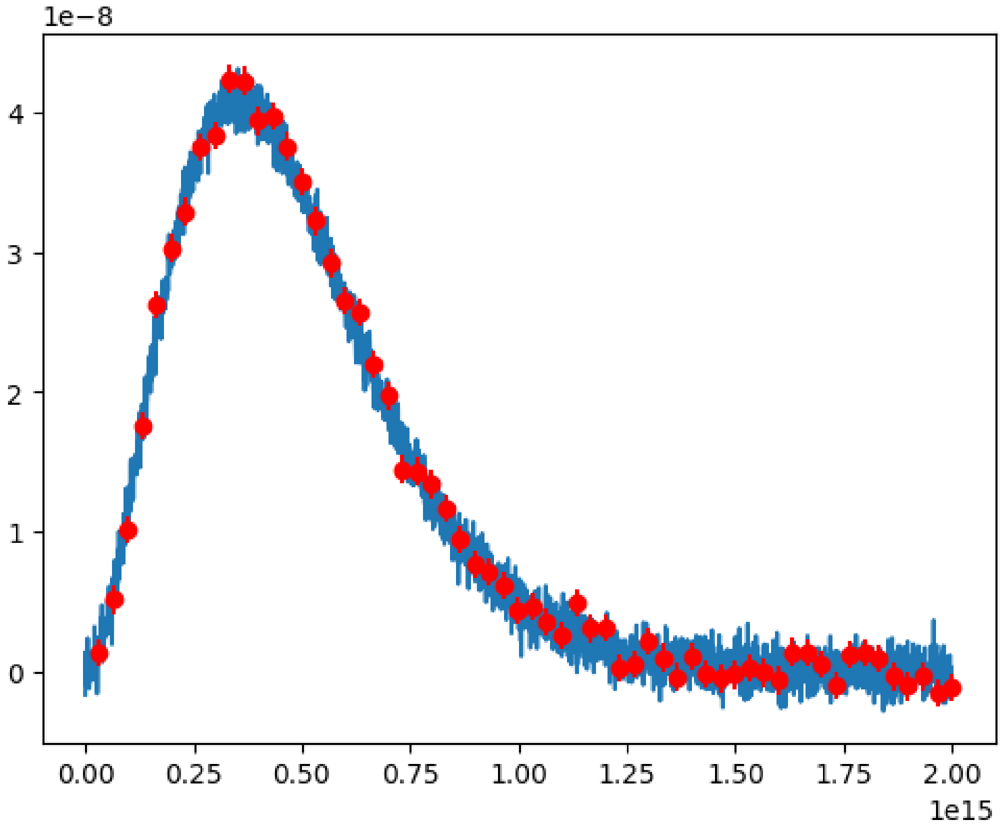
```
The value of planck's constant is 6.643229765102559e-34 and the temperature at
which these observations are taken is 6011.361525876954k
```

[11]: [<matplotlib.lines.Line2D at 0x7fd4e64aca00>]



Here since the function is complex, we have to some initial guess so it will return correct answer.

[29]:
```
plt.plot(x1,y1)
plt.errorbar(x1[::50], y1[::50], np.std(y3-y1), fmt='ro')
plt.show()
```


1e−8
4
3
2
1
0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
1e15