Computing for Physicists (PHYS2962) Test 1

Fill in this jupyter-notebook and upload <u>both</u> the jupyter-notebook and a pdf of it to the LMS at the end of time. (3:30)

1) The command line

For this problem, retrieve a file named "datatest.txt" from the remote computer with ip address 74.69.18.77. On your local machine, preprocess the file at the command line and plot the data.

(Note: the username is "class" and the password is "phys2962")

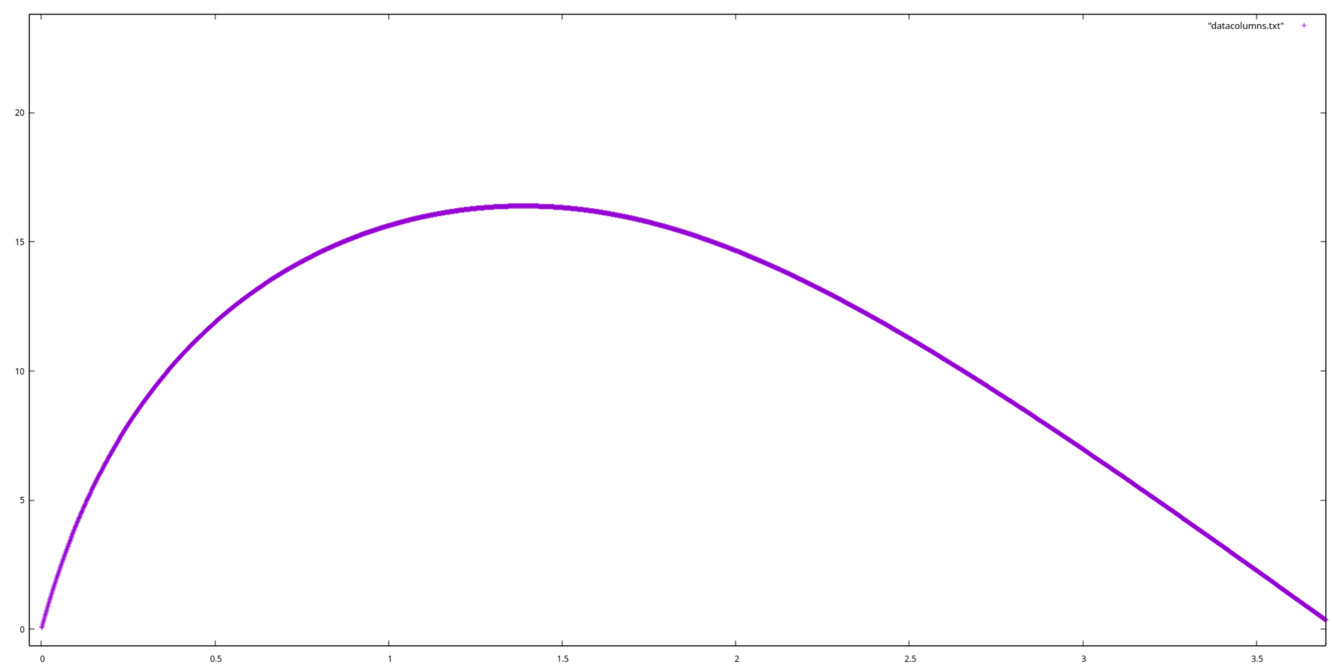
a) What command did you issue from the command prompt in order to copy the file to your local machine?

``` scp class@74.69.18.77:./a54/sdkfjhsdf/mary/a/little/lamb/arewethereyet/yes/datatest.txt data.txt

### b) At the command line create a new file which contains two columns of numerical data (the y and t values). What command/s did you use?

``` cat data.txt | grep y= | gawk '{print $2,4$}' >> datacolumns.txt

### c) read in the data from the new file and plot y vs. t



```
        faq, bugs, etc: type "help FAQ"
        immediate help: type "help" (plot window: hit 'h')

Terminal type is now 'qt'
gnuplot> plot columndata.txt
         undefined variable: columndata

gnuplot> plot "columndata.txt"
         warning: Cannot find or open file "columndata.txt"
         No data in plot

gnuplot> plot "datacolumns.txt"
gnuplot> plot "datacolumns.txt"
gnuplot> q
[plea@x-ray-174 Exam1]$
```

2) numPy arrays

a) write code below to generate the following 3-dimensional array:

```
array([[[  0,   4,   8],
        [ 12,  16,  20],
        [ 24,  28,  32],
        [ 36,  40,  44],
        [ 48,  52,  56],
        [ 60,  64,  68],
        [ 72,  76,  80],
        [ 84,  88,  92],
        [ 96, 100, 104]],

       [[108, 112, 116],
        [120, 124, 128],
        [132, 136, 140],
        [144, 148, 152],
        [156, 160, 164],
        [168, 172, 176],
        [180, 184, 188],
        [192, 196, 200],
        [204, 208, 212]],

       [[216, 220, 224],
        [228, 232, 236],
        [240, 244, 248],
        [252, 256, 260],
        [264, 268, 272],
        [276, 280, 284],
        [288, 292, 296],
        [300, 304, 308],
        [312, 316, 320]]])
```

In [106…

```python
import numpy as np

arr = np.arange(0,321,4)
arr.reshape(3,9,3)
```

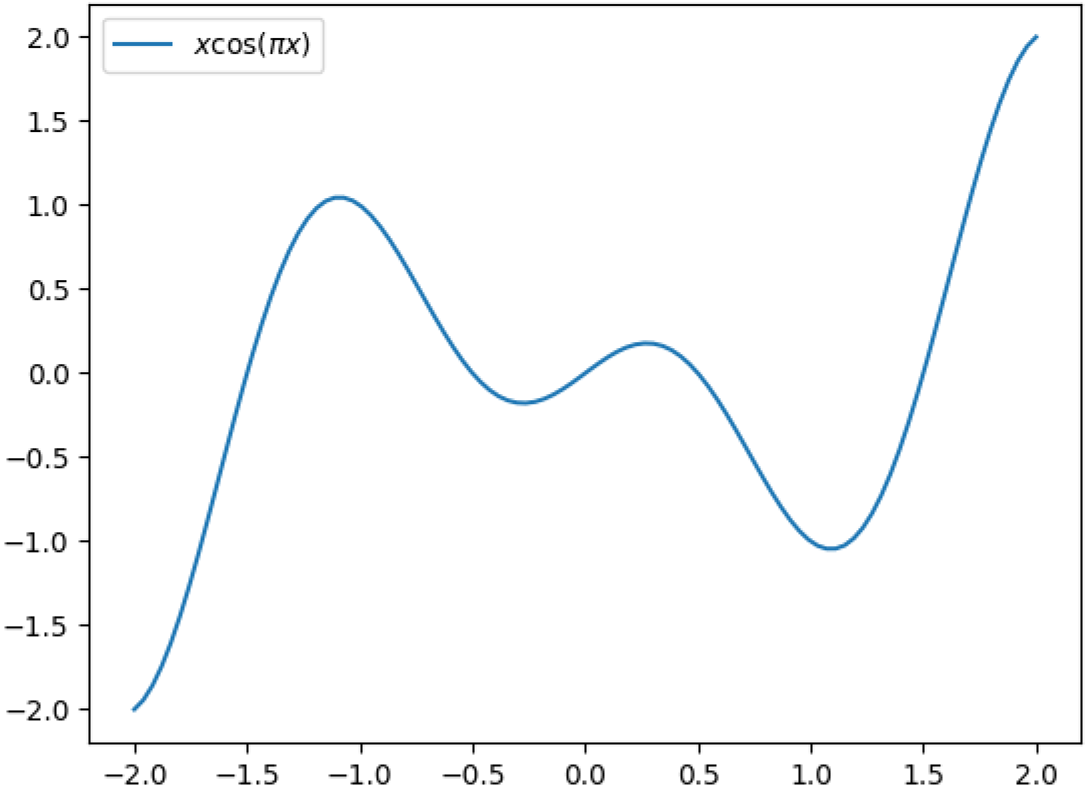
Out[106]:
```
array([[[  0,   4,   8],
        [ 12,  16,  20],
        [ 24,  28,  32],
        [ 36,  40,  44],
        [ 48,  52,  56],
        [ 60,  64,  68],
        [ 72,  76,  80],
        [ 84,  88,  92],
        [ 96, 100, 104]],

       [[108, 112, 116],
        [120, 124, 128],
        [132, 136, 140],
        [144, 148, 152],
        [156, 160, 164],
        [168, 172, 176],
        [180, 184, 188],
        [192, 196, 200],
        [204, 208, 212]],

       [[216, 220, 224],
        [228, 232, 236],
        [240, 244, 248],
        [252, 256, 260],
        [264, 268, 272],
        [276, 280, 284],
        [288, 292, 296],
        [300, 304, 308],
        [312, 316, 320]]])
```

b) Plot $xcos(\pi x)$ in the range of $x = [-2, 2]$

In [107…
```python
import matplotlib.pyplot as plt
#Numpy imported from previous section
x = np.linspace(-2,2,100)
plt.plot(x, x*np.cos(np.pi*x), label = "$x  \\cos(\\pi x)$")
plt.legend()
plt.show()
```



c) Generate an array of uniformly distributed random numbers in the range of [0,2) (generate enough so that you can do decent statistics ~ 10,000). Calculate the mean and standard deviation of the dataset.

In [108…

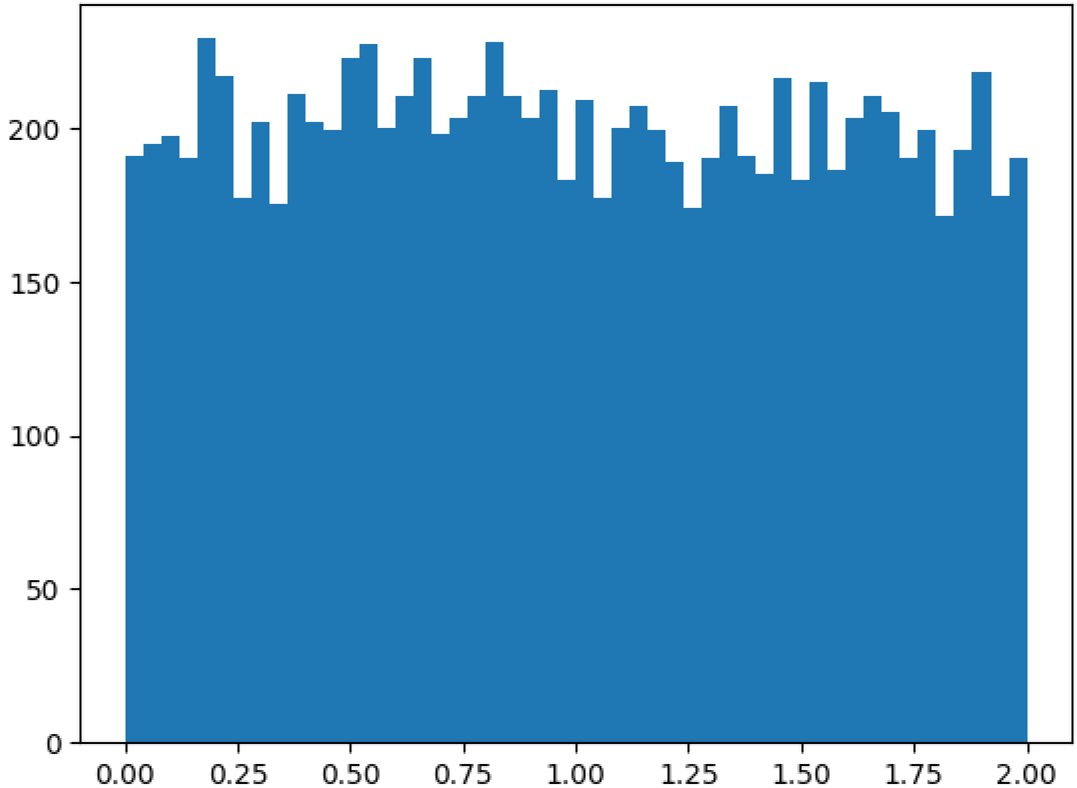
```
import random as rand
statarr = np.random.rand(10000) * 2
mean = np.mean(statarr)
print("Mean is: ", mean)
stddev = np.std(statarr)
print("Standard Deviation is:, ", stddev )
```

```
Mean is:  0.9906038963948719
Standard Deviation is:,  0.5738970435849345
```

d) Display a histogram of the numbers generated in part (c)

In [109…

```
plt.hist(statarr, bins=50)
plt.show()
```

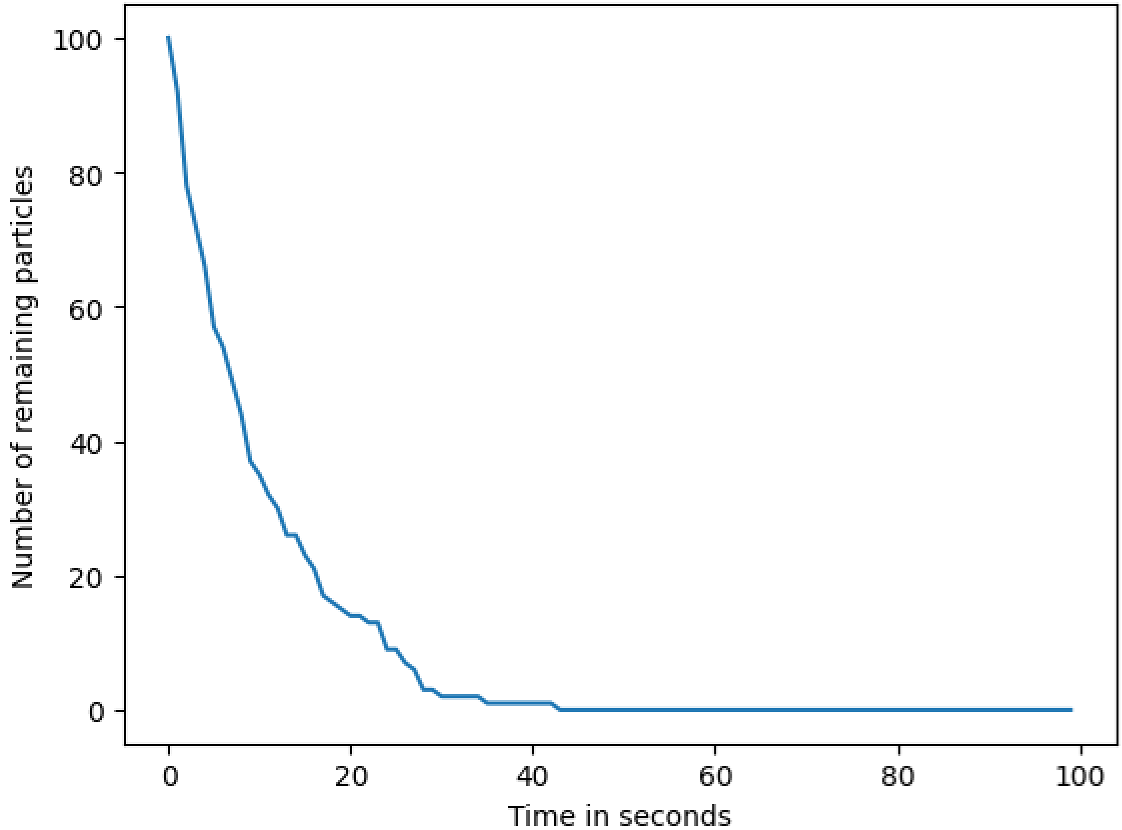


3) A Radioactive isotope has a 10% chance of decaying every second. Given a sample containing 100 atoms of such an isotope --

a) Perform a Monte Carlo simulation to determine the Number of atoms (N) remaining as a function of time and plot it:

In [118…

```
N = 100
Ns=[]
for i in range(100):
    Ns.append(N)
    for j in range(N):
        if(np.random.random() < 0.1 ): #This would be a very radioactive sample
            N = N-1
plt.ylabel("Number of remaining particles")
plt.xlabel("Time in seconds")
plt.plot(Ns)
plt.show()
print(len(Ns))
```



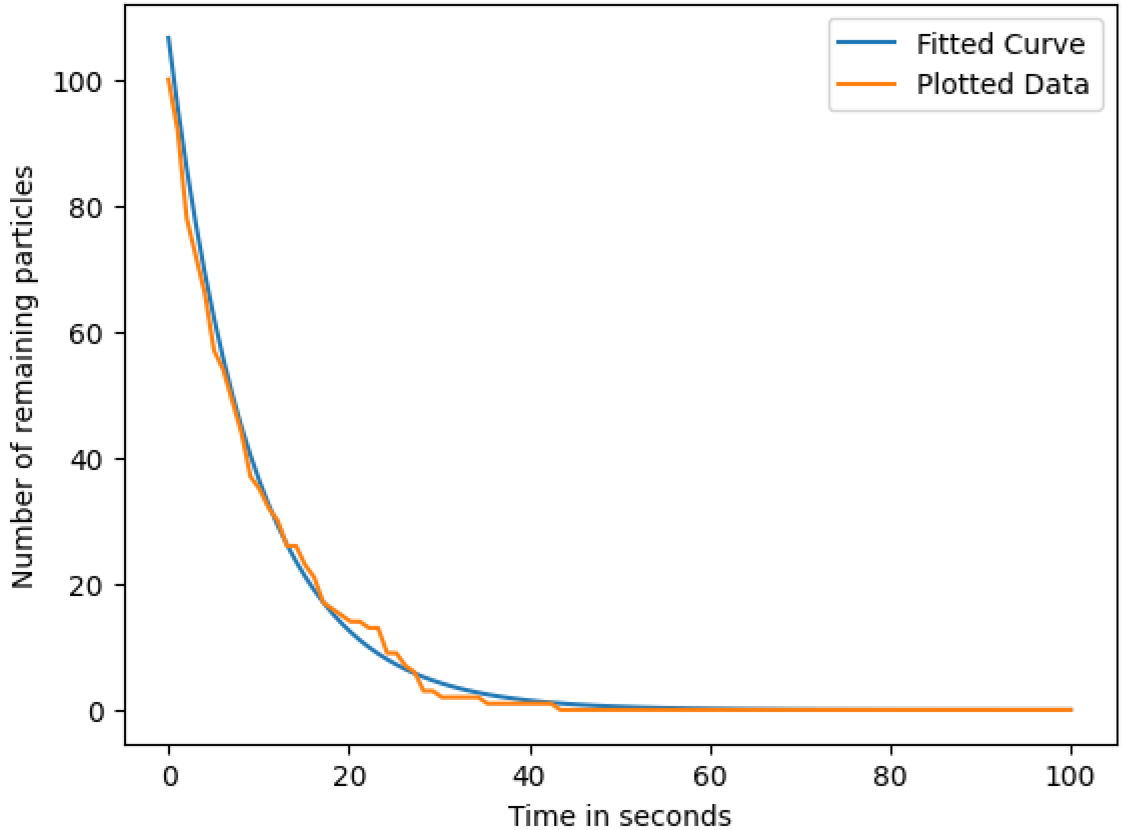

```
100
```

b) Fit your results from part a) to an exponential, $N = N_0 e^{-t/\tau}$. What value of τ do you obtain in your fit? Plot the result from (a) together with your exponential fit.

In [119…

```
from scipy.odr import Model, RealData, ODR
def exponentialdecay(params,t):
    #params[0] is N_0, params[1] is tau
    return params[0] * np.exp(-t/params[1])
def fit_exp_decay_odr(t_data, y_data):
    # Initial guess
    initial_guess = [100, 10]
    model = Model(exponentialdecay)
    data = RealData(t_data, y_data)
    odr = ODR(data, model, beta0=initial_guess)
    output = odr.run()
    params = output.beta
    y_fit = exponentialdecay(params, t_data)
    return params, y_fit
t_data = np.linspace(0,100,100)
Ns = np.array(Ns)
params, yfit = fit_exp_decay_odr(t_data, Ns)
print("Obtained value of tau is: ", params[1])
plt.plot(t_data, exponentialdecay(params, t_data), label = "Fitted Curve")
plt.ylabel("Number of remaining particles")
plt.xlabel("Time in seconds")
plt.plot(t_data,Ns, label="Plotted Data")
plt.legend()
plt.show()
```

```
Obtained value of tau is:  9.361994521114157
```



c) Perform a large number of realizations (100~1000) of the simulation in (a). What is the average time required for all atoms to decay? What is the standard deviation of the time required for all atoms to decay?

In [120…
```
def monteCarloTime():
    N = 100
    Ns=[]
    for i in range(1000):
        Ns.append(N)
        for j in range(N):
            if(np.random.random() < 0.1 ): #This would be a very radioactive sample
                N = N-1
    return Ns.index(0)
fulldecaytime = []
for i in range(1000):
    time = monteCarloTime()
    fulldecaytime.append(time)

avgdecaytime = np.mean(fulldecaytime)
stddecaytime = np.std(fulldecaytime)
print("Average time = ", avgdecaytime, " Standard Devation of time: " , stddecaytime)
```

```
Average time =  49.884  Standard Devation of time:  12.144568497892381
```

4) χ^2 vs. least squares fit

a) Perform a least squares fit of the following data to the function $y(x) = a + bx^2$. Indicate the fitted values of a and b.

In [121…
```
x=np.array([ 0.        ,  0.1010101 ,  0.2020202 ,  0.3030303 ,  0.4040404 ,
```

```
       0.50505051,  0.60606061,  0.70707071,  0.80808081,  0.90909091,
       1.01010101,  1.11111111,  1.21212121,  1.31313131,  1.41414141,
       1.51515152,  1.61616162,  1.71717172,  1.81818182,  1.91919192,
       2.02020202,  2.12121212,  2.22222222,  2.32323232,  2.42424242,
       2.52525253,  2.62626263,  2.72727273,  2.82828283,  2.92929293,
       3.03030303,  3.13131313,  3.23232323,  3.33333333,  3.43434343,
       3.53535354,  3.63636364,  3.73737374,  3.83838384,  3.93939394,
       4.04040404,  4.14141414,  4.24242424,  4.34343434,  4.44444444,
       4.54545455,  4.64646465,  4.74747475,  4.84848485,  4.94949495,
       5.05050505,  5.15151515,  5.25252525,  5.35353535,  5.45454545,
       5.55555556,  5.65656566,  5.75757576,  5.85858586,  5.95959596,
       6.06060606,  6.16161616,  6.26262626,  6.36363636,  6.46464646,
       6.56565657,  6.66666667,  6.76767677,  6.86868687,  6.96969697,
       7.07070707,  7.17171717,  7.27272727,  7.37373737,  7.47474747,
       7.57575758,  7.67676768,  7.77777778,  7.87878788,  7.97979798,
       8.08080808,  8.18181818,  8.28282828,  8.38383838,  8.48484848,
       8.58585859,  8.68686869,  8.78787879,  8.88888889,  8.98989899,
       9.09090909,  9.19191919,  9.29292929,  9.39393939,  9.49494949,
       9.5959596 ,  9.6969697 ,  9.7979798 ,  9.8989899 , 10.        ])
y=np.array([  1.03545807,   1.28195072,   1.00839038,   1.36913201,
         0.62679988,   1.00034059,   1.3169403 ,   0.92239042,
         1.71110414,   1.59441357,   2.97854407,   3.06619471,
         3.16706311,   3.08851556,   2.72175075,   3.66098392,
         5.06361123,   4.19000385,   5.07173457,   6.689527  ,
         6.15908147,   5.62465897,   7.86424009,   5.56925392,
         6.7803182 ,   6.20675219,   8.02694204,  10.48898249,
         7.87006884,   6.83874986,  11.95823263,  10.55901979,
        11.90532882,  14.07415747,  12.76850834,  15.57531864,
        14.94107618,  14.00150556,  15.13171867,  17.00442417,
        19.1028683 ,  14.24767362,  21.51084622,  22.8089485 ,
        22.62978242,  23.56177019,  21.95860918,  23.66265132,
        23.74949617,  32.01727589,  29.39263196,  24.78853825,
        25.96335042,  30.36817615,  29.36061607,  30.03992341,
        34.8370933 ,  34.57508206,  40.12515065,  36.80525687,
        40.54987719,  39.32043583,  42.95524361,  45.56408584,
        40.6492202 ,  41.96250039,  51.33449992,  41.71904604,
        49.11857925,  41.52333409,  52.78654852,  46.61854518,
        53.16282846,  56.56414651,  58.16278177,  53.98777614,
        61.49677912,  65.5494806 ,  57.2735989 ,  55.9342531 ,
        68.70554848,  65.17879191,  74.70959111,  72.26875598,
        82.45815245,  80.07970789,  69.38613556,  77.99281414,
        76.15075653,  83.1599349 ,  85.55032887,  88.00218236,
        90.41702902, 102.15919646,  85.09058058,  96.16198339,
       103.92010228,  94.48636688, 103.10764057, 106.06465354])
errorbars = np.array([0.1       , 0.15050505, 0.2010101 , 0.25151515, 0.3020202 ,
       0.35252525, 0.4030303 , 0.45353535, 0.5040404 , 0.55454545,
       0.60505051, 0.65555556, 0.70606061, 0.75656566, 0.80707071,
       0.85757576, 0.90808081, 0.95858586, 1.00909091, 1.05959596,
       1.11010101, 1.16060606, 1.21111111, 1.26161616, 1.31212121,
       1.36262626, 1.41313131, 1.46363636, 1.51414141, 1.56464646,
       1.61515152, 1.66565657, 1.71616162, 1.76666667, 1.81717172,
       1.86767677, 1.91818182, 1.96868687, 2.01919192, 2.06969697,
       2.12020202, 2.17070707, 2.22121212, 2.27171717, 2.32222222,
       2.37272727, 2.42323232, 2.47373737, 2.52424242, 2.57474747,
       2.62525253, 2.67575758, 2.72626263, 2.77676768, 2.82727273,
       2.87777778, 2.92828283, 2.97878788, 3.02929293, 3.07979798,
       3.13030303, 3.18080808, 3.23131313, 3.28181818, 3.33232323,
       3.38282828, 3.43333333, 3.48383838, 3.53434343, 3.58484848,
       3.63535354, 3.68585859, 3.73636364, 3.78686869, 3.83737374,
       3.88787879, 3.93838384, 3.98888889, 4.03939394, 4.08989899,
       4.14040404, 4.19090909, 4.24141414, 4.29191919, 4.34242424,
       4.39292929, 4.44343434, 4.49393939, 4.54444444, 4.59494949,
       4.64545455, 4.6959596 , 4.74646465, 4.7969697 , 4.84747475,
       4.8979798 , 4.94848485, 4.9989899 , 5.04949495, 5.1       ])
```

In [122…

```python
import scipy as sp
def drwestsfunction(params,x):
    return params[0] + params[1] * x ** 2

paramslsq = sp.optimize.curve_fit(drwestsfunction, x,y)
print("A = ", paramslsq[0], " B = " , paramslsq[1])
```

```
A =  [18.6243544]  B =  [[0.67452297]]
```

b) Perform a χ^2 fit of the same data. What values of a and b do you obtain?

In [123…

```python
paramschisq = sp.optimize.curve_fit(drwestsfunction, x,y, sigma=errorbars)
print("A = ", paramschisq[0], " B = " , paramschisq[1])
```

```
A =  [4.93403075]  B =  [[0.50848438]]
```

c) Plot the original data (with error bars) in black, your least-squares fit function in red, and your χ^2 fit function in green all in the same figure.

In [124…

```python
plt.errorbar(x,y,errorbars, color= "black", ecolor="grey" )
plt.plot(x,drwestsfunction(paramslsq,x)[0], label= "Least Squares Regression", color = "
plt.plot(x,drwestsfunction(paramschisq,x)[0], label = "Chi Squared Regression", color =
plt.xlabel("X")
plt.ylabel("Y")
plt.show()
```