## Exercise 1: Curve Fit using Polynomial Regression 3rd degree

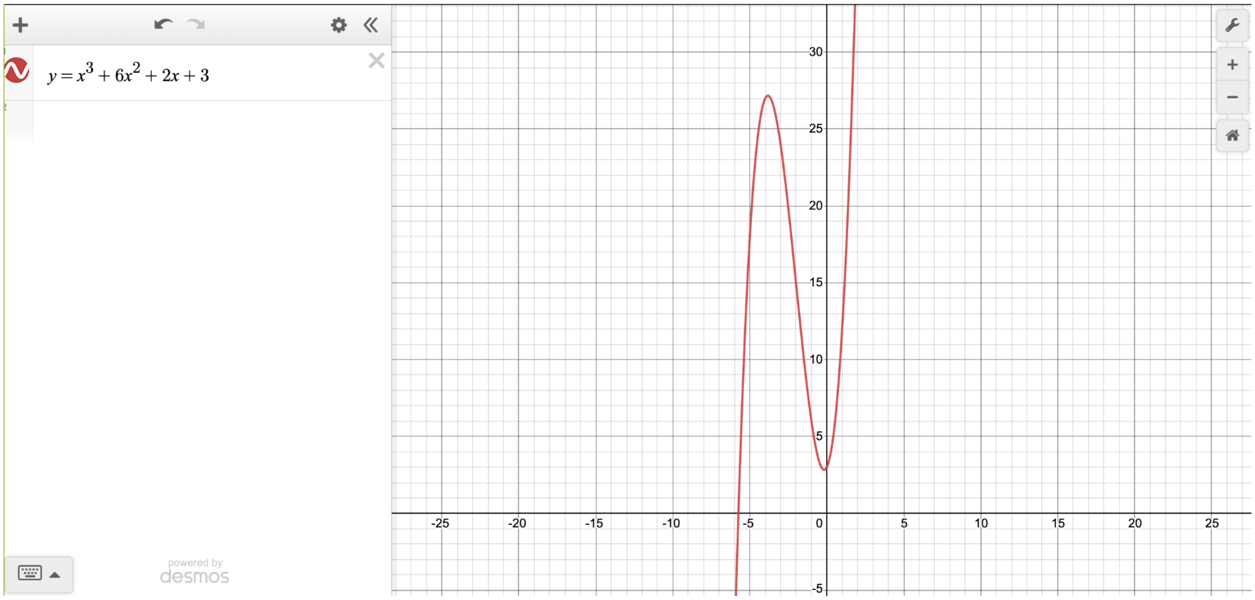



*Figure: Desmos Calculator of the third-degree polynomial function*

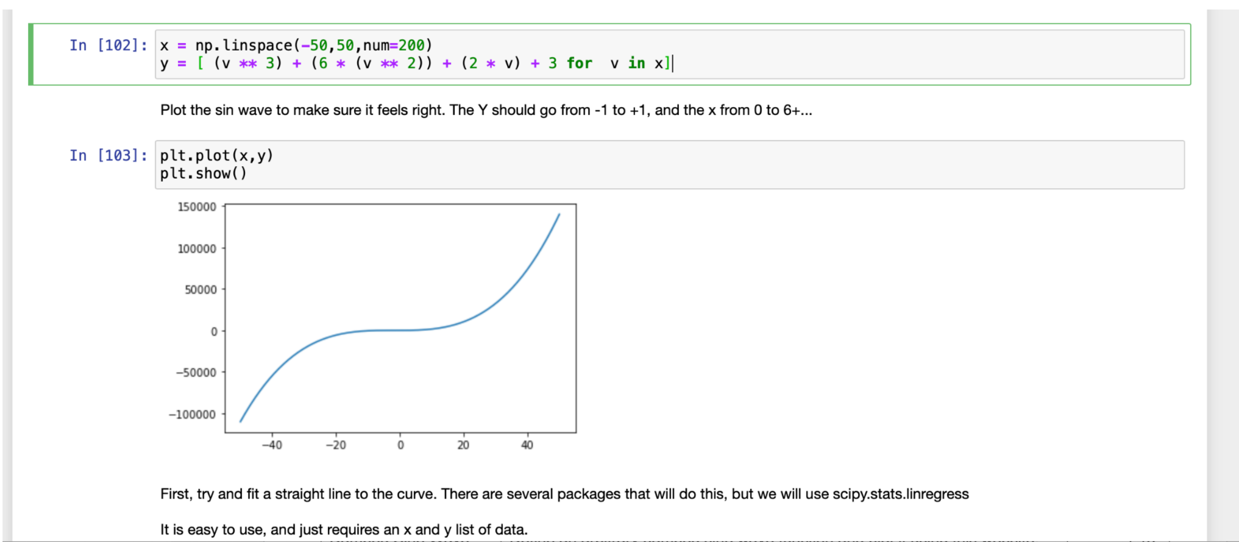



*Figure: Plot of polynomial function in Python*

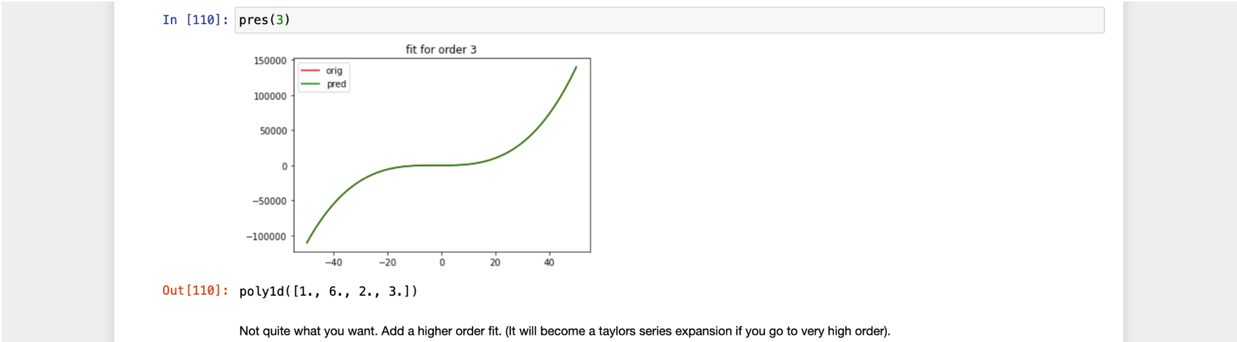



*Figure: Third-order polynomial regression*

# Curve fit using Ridge or Linear Regression Method

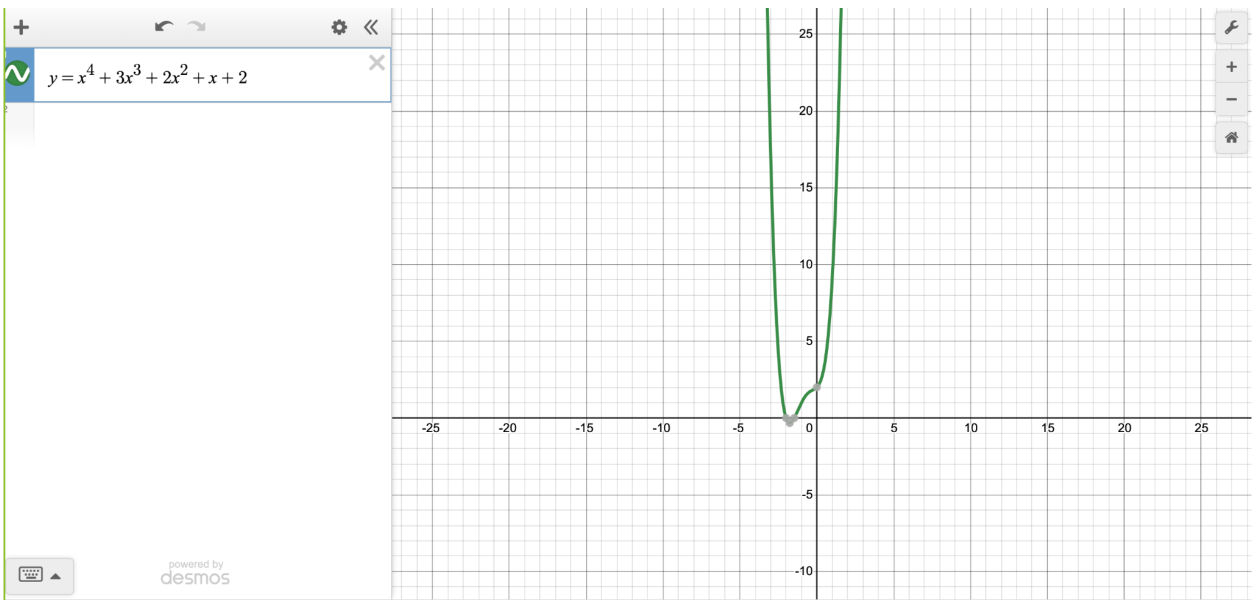



*Figure: Polynomial from Desmos*

```
In [2]: import numpy as np
        import scipy.optimize as opt
        import scipy.stats as st
        import math
        import matplotlib.pyplot as plt
```

Create an X and Y vector for 100 points of a sin wave.

This uses elaborations, and embedded for loops.

```
In [3]: x = np.linspace(-50,50,num=200)
        y = [ (v ** 4) + (3 * (v ** 3)) + (2 * (v ** 2))+ (v) + 2 for  v in x]
```

Plot the sin wave to make sure it feels right. The Y should go from -1 to +1, and the x from 0 to 6+...

```
In [4]: plt.plot(x,y)
        plt.show()
```

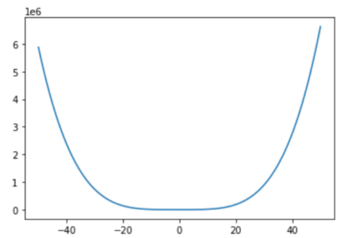
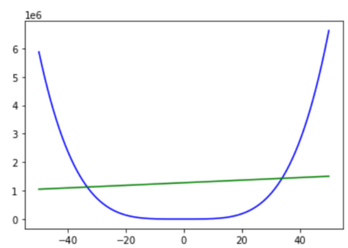



*Figure: Plot of polynomial function in Python*

```
In [6]: ypred = [ linfit.slope*xv+linfit.intercept for xv in x]
        plt.plot(x,y,'b-')
        plt.plot(x,ypred,'g-')
        plt.show()
```




Just because we got an answer and the software doesn't give an error doesn't mean the fit is a 'good' one. The value R^2 indicates how well the data fits the line. An R^2 of 1 is a perfect fit, and 0 is no relationship. Take a look at R^2:

```
In [7]: linfit.rvalue **2
Out[7]: 0.0059655842029013905
```

It kinda fits, but isn't very useful for engineering work.

*Figure: Linear regression*

# Damped Sine wave

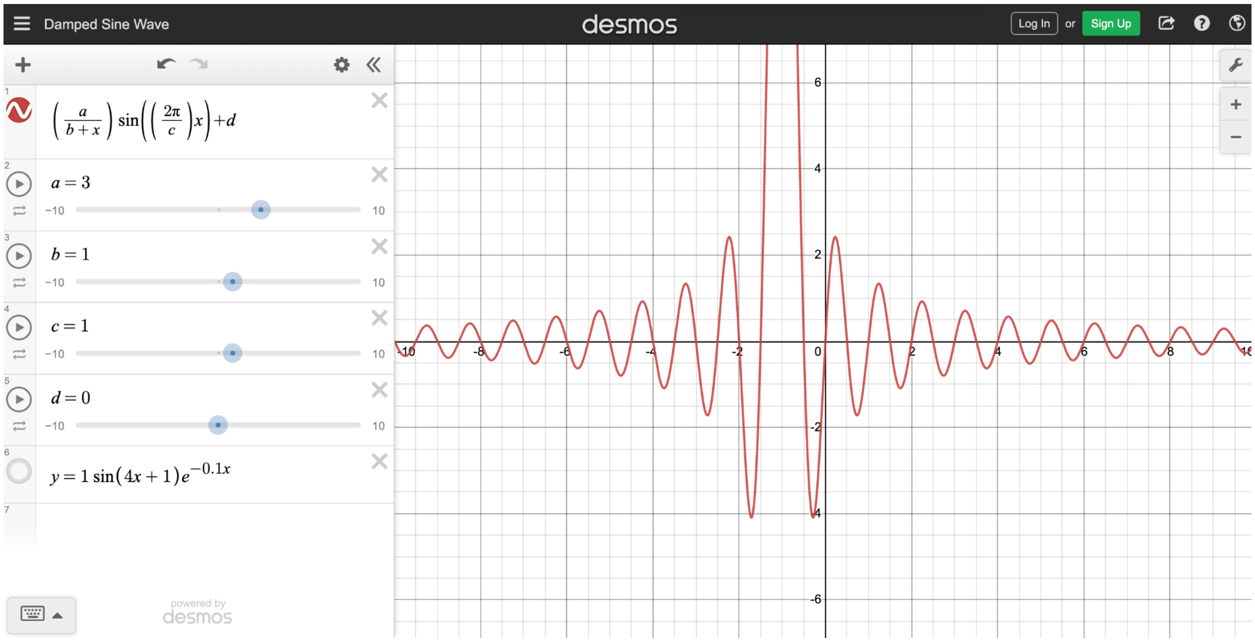



Figure: Damped Sine Wave from Desmos

```
In [17]: # first p is the sin multiplier
         # second p is the sin phase
         # third p is the exponent weight
         # fourth is an overall gain factor
         #def fds(xin,sm,sp,ew,gain):
             #return [ gain*math.sin(sm*x+sp)*math.exp(-ew*x) for x in xin]
```

```
In [18]: def frs(xin, a, b, c, d):
             return [((a/(b + x))*math.sin(((2*math.pi)/c)*x)+d) for x in xin]
```

Make a quick plot to test things out...

```
In [19]: xv = [x/10 for x in range(400)] # create x values
         #yv = fds(xv,4,1,0.1,1) # just dummy values for now
         yv = frs(xv, 3, 1, 1, 0) # just dummy values for now
         plt.plot(xv,yv)
         plt.show()
```

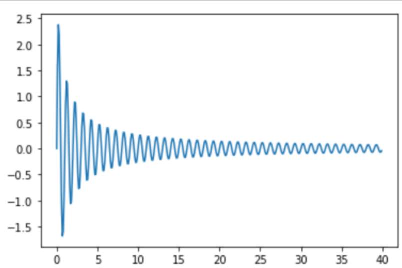


Figure: Plot of damped sine wave in Python

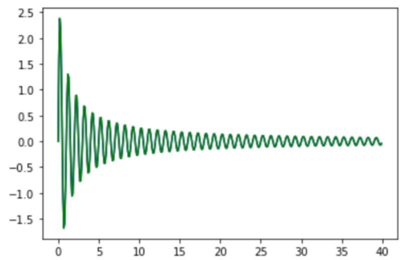
Now, try a curve fit to this data... See what parameters the software finds.

In [22]:
```
popt,pcov = opt.curve_fit(frs,xv,yv)
print(popt)
```

```
[ 3.00000000e+00  9.99999999e-01  1.00000000e+00 -1.93345455e-10]
```

The Y data was generated with 4,1,0.1,1 The program didn't find that. But if you look closely, there are two negatives. Plot both, and see what it looks like

In [23]:
```
plt.plot(xv,yv,'b')
plt.plot(xv,frs(xv,*popt),'g')
plt.show()
```




The two curves are right on top of each other. The negatives cancel in the end, and the phase is 2*pi - 1, and is mathematicly the same phase

*Figure: Damped sine wave curve fit in python*

# Curve Fit with Noise

In [34]:
```
columns = ["X", "Y"]
df = pd.read_csv("wav_10Hz_Output_mono.csv", usecols=columns)
#print("Contents in csv file:\n", df)
plt.plot(df.X, df.Y)
plt.show()
#df = pd.read_csv()
```

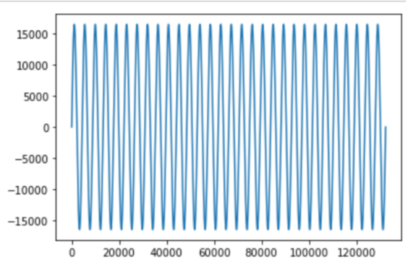
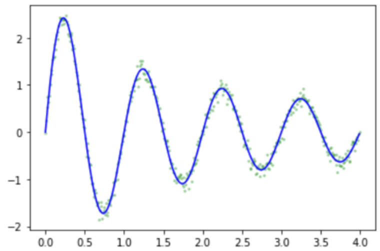



*Figure: Reading the tone csv file and plot*

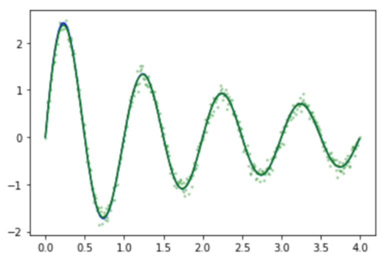
In [25]:
```
xv = [x/100 for x in range(400)] # create x values
#xv = df.X
yp = frs(xv, 3, 1, 1, 0) # perfect y values
#yp = df.Y
yv = [ y+random.gauss(0,0.1) for y in yp] # values with noise
plt.plot(xv,yp,'b')
plt.scatter(xv,yv,alpha=0.3,s=3,c='green')
plt.show()
```




Now, perform the curve fitting and look at the results compared to the perfect curve. (Without any noise)

*Figure: Damped sine function with noise*

In [29]:
```
#print(yp)
popt,pcov = opt.curve_fit(frs,xv,yv)
plt.plot(xv,yp,'b')
plt.plot(xv,frs(xv,*popt),'g')
plt.scatter(xv,yv,alpha=0.3,s=3,c='green')
plt.show()
```




The noise has confused the fit a little, but not too much It helps to plot the error between the perfect curve, and the fit curve.

In [ ]:

*Figure: Damped sine function with noise that fits better on curve*

# Curve Fit with Multi-Variables

```
In [2]: # make two variables
        x0 = [x/50 for x in range(100)]
        x1 = [math.sin(x/10 + math.pi) for x in range(100)]
        xa=(x0,x1)
```

```
In [3]: def rf(X,fx0,fx1):
            x0,x1=X
            rv=np.sin(np.multiply(x0,fx0)+np.sin(np.multiply(x1,fx1)))
            return rv
```

```
In [4]: yv=rf((x0,x1),2,3)
        plt.plot(yv)
        plt.show()
```

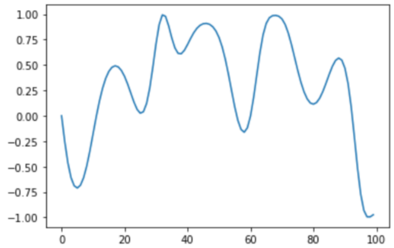


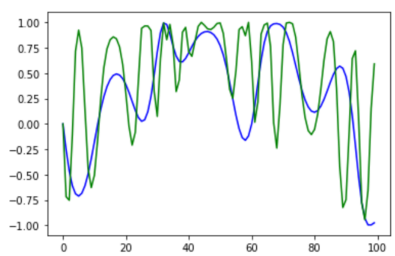

*Figure: Curve of function with multiple variables*

```
In [7]: def with_hint(h0,h1):
            popt,pcov=opt.curve_fit(rf,xa,yv,(h0,h1))
            print(popt)
            plt.plot(rf((x0,x1),2,3),'b')
            plt.plot(rf((x0,x1),*popt),'g')
            plt.show()
```

Now, test it out...

```
In [17]: with_hint(2,10)
```

```
[1.76456234 9.9125184 ]
```

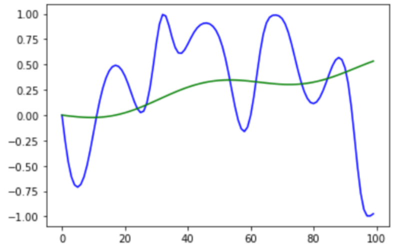



Following trend, but not accurately

*Figure: Curve of function with multiple scenarios trial 1*

In [9]:
```
with_hint(0,0)
```

[0.26317573 0.09050133]

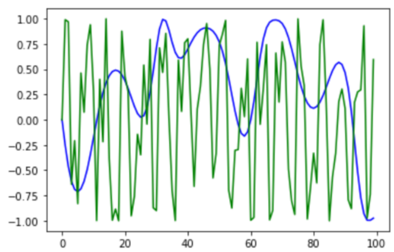


Trend does not hit all the maximum and minimum values

*Figure: Curve of function with multiple scenarios trial 2*

In [27]:
```
with_hint(50,100)
```

[50.62075144 98.8003673 ]

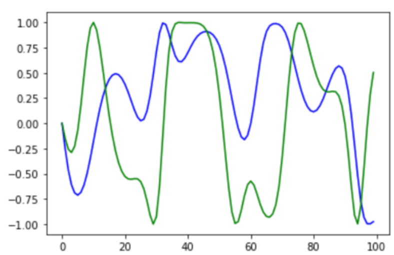


Too noisy and overshoots

*Figure: Curve of function with multiple scenarios trial 3*

In [23]:
```
with_hint(10,3)
```

[9.28158081 3.43654987]

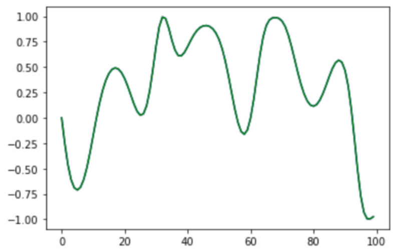


Follows the rise and falls of the curve, but stil overshoots

*Figure: Curve of function with multiple scenarios trial 4*

```
In [25]: with_hint(4,5)
```

```
[2. 3.]
```

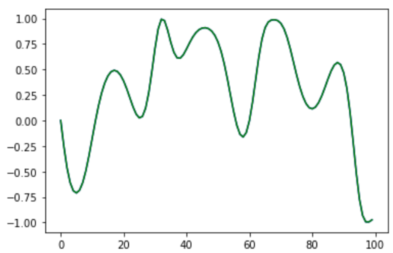



Trend matches curve

*Figure: Curve of function with multiple scenarios trial 5*

```
In [26]: with_hint(2.2,3.4)
```

```
[2. 3.]
```




Trend matches curve. Parameters between (2-2.9, 3-3.9) seem to work

Moral of this is multivariable with repeating variables have many local minimums in the error function. You may not find the 'best' answer. It seems ironic, but this works best if you already know the answer before you start.

*Figure: Curve of function with multiple scenarios trial 6*

Coin Collector


*Figure: Coin collector game with different actor, dimensions, and background*