Homework 5

This homework is going to be all about advanced plotting and curve fitting! All of these are extremely useful in any STEM based course, so the goal of this homework is to get y'all more comfortable with these methods.

In [4]:
```
#Code Preamble#
import numpy as np
import scipy as sp
import matplotlib.pyplot as plt
import scipy.optimize as fitter
%matplotlib inline
```

Problem 1: 3D Subplots (25 Points)

Plot the following graphs in one figure. We should see 4 total plots (2 x 2)

Top Left:

$$f(x,y) = \frac{sgn(xy) * sgn(1-(9x)^2+(9y)^2)}{9}$$

Top Right:

$$f(x,y) = (0.4^2 - (0.6-(x^2+y^2)^{1/2})^2)^{1/2}$$

Bottom Left:

$$f(x,y) = \frac{1}{15(x^2+y^2)}$$

Bottom Right:

$$f(x,y) = \frac{(1-sgn(-x-.51+abs(2y)))(sgn(\frac{1}{2}-x)+1)}{3}$$

Use the same colormap for the whole figure (all 4 plots), add a colorbar for the whole figure, a title for each plot, and x and y labels for each plot. Pick the same domain for x and y and it should be the same for all plots. I will let you figure out what domain works best. Each of these is an interesting shape or structure. Create a markdown cell below your plotting cell describing what you see in each plot.

Hint: `numpy` has a sgn(x) function

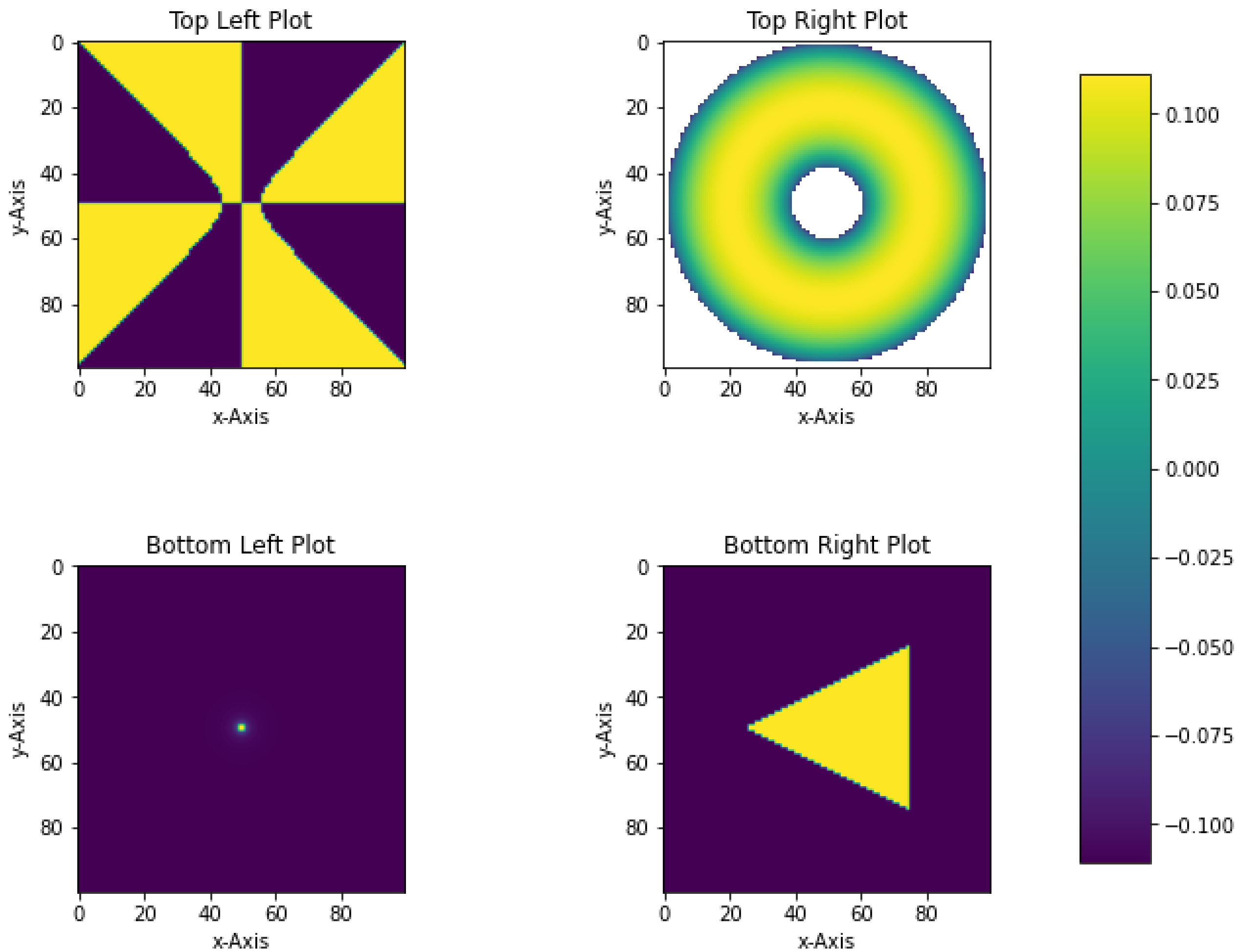
In [5]:
```
x,y = np.meshgrid(np.linspace(-1, 1, 100),
                   np.linspace(-1, 1, 100))

top_left = (np.sign(x*y)*np.sign(1-(9*x)**2+(9*y)**2))/9
top_right = np.sqrt(0.4**2-(0.6-np.sqrt(x**2+y**2))**2)
bottom_left = 1/(15*(x**2+y**2))
bottom_right = ((1-np.sign(-x-0.51+np.abs(2*y)))*(np.sign(1/2-x)+1))/3

fig, ((tleft,tright),(bleft,bright)) = plt.subplots(2,2,figsize = (10,8)) # 2 row 2 columns, to make left and right plo
plt.subplots_adjust(wspace = 0.6, hspace = 0.6) #space between plots

#Top Left Plot
t_left = tleft.imshow(top_left)
#plot1.plot(x,g_plot1, label= '$g(x) = e^{-3x}$')
#Title
tleft.set_title("Top Left Plot")
#Axis Label
tleft.set_xlabel("x-Axis")
tleft.set_ylabel("y-Axis")



#Top Right Plot
tright.imshow(top_right)
#plot1.plot(x,g_plot1, label= '$g(x) = e^{-3x}$')
#Title
tright.set_title("Top Right Plot")
#Axis Label
tright.set_xlabel("x-Axis")
tright.set_ylabel("y-Axis")


#Bottom Left Plot
bleft.imshow(bottom_left)
#plot1.plot(x,g_plot1, label= '$g(x) = e^{-3x}$')
#Title
```

```python
bleft.set_title("Bottom Left Plot")
#Axis Label
bleft.set_xlabel("x-Axis")
bleft.set_ylabel("y-Axis")


#Bottom Right Plot
bright.imshow(bottom_right)
#plot1.plot(x,g_plot1, label= '$g(x) = e^{-3x}$')
#Title
bright.set_title("Bottom Right Plot")
#Axis Label
bright.set_xlabel("x-Axis")
bright.set_ylabel("y-Axis")


# subplot for colorbar
fig.subplots_adjust(right=0.8)
cbar_ax = fig.add_axes([0.85, 0.15, 0.05, 0.7])
fig.colorbar(t_left, cax=cbar_ax)
```

```
C:\Users\josel\AppData\Local\Temp\ipykernel_5088\1477371814.py:5: RuntimeWarning: invalid value encountered in sqrt
  top_right = np.sqrt(0.4**2-(0.6-np.sqrt(x**2+y**2))**2)
```

Out[5]:

```
<matplotlib.colorbar.Colorbar at 0x1d1ae16a520>
```



Plot Description: We can see a total different output from the graphs above. The shape for the top left plot is like a pattern of yellow and purple triangle changing from one color to another. The top right plot on the other hand, has shape like a donut that consist of yellow-green color and hole in the middle and white background. The bottom left plot is just showing a yellow small dot with all purple background no matter how I tried to change the domain for x and y (and it will be worse for the other if I change the domain). Lastly, the bottom right plot has a shape of one yellow triangle pointing to the left with purple background.

Plotting Data

Problem 2: Covid-19 (25 Points)

Last week we learned how to import data into our notebooks in the form of arrays. Well we can actually plot that data! For this problem we will actually make a pretty relevant plot to the state of the world in 2020. As someone who has taken the Python DeCal you will be able to say that you have worked with real COVID-19 Data. So often we simply listen to people tell us about the data, but you should be able to look at it yourself (to reduce bias of course).

In this problem we will plot some COVID-19 United States Data from the https://ourworldindata.org/coronavirus-source-data, You should have a file named `Covid_Data.csv` in the same directory as this notebook. The data starts on January 1st, 2020 and each new row below that is the next day until the end of the data set which goes to September 23rd, 2020.

Part a) Your task for this problem is to make a scatter plot of the Total Cases (unsmoothed) in the United States for each day since the data started being taken. Make sure to import the data using the `numpy.loadtxt` function.

Hint 1: Set `Unpack = True` and import all the required data sets as their own 1D arrays

Hint 2: Skip the first row in the data set.

Hint 3: You dont need the first column because that is the date which would be a string, while the rest of the data is floats. This may cause some issues. Think about how you can get around this.

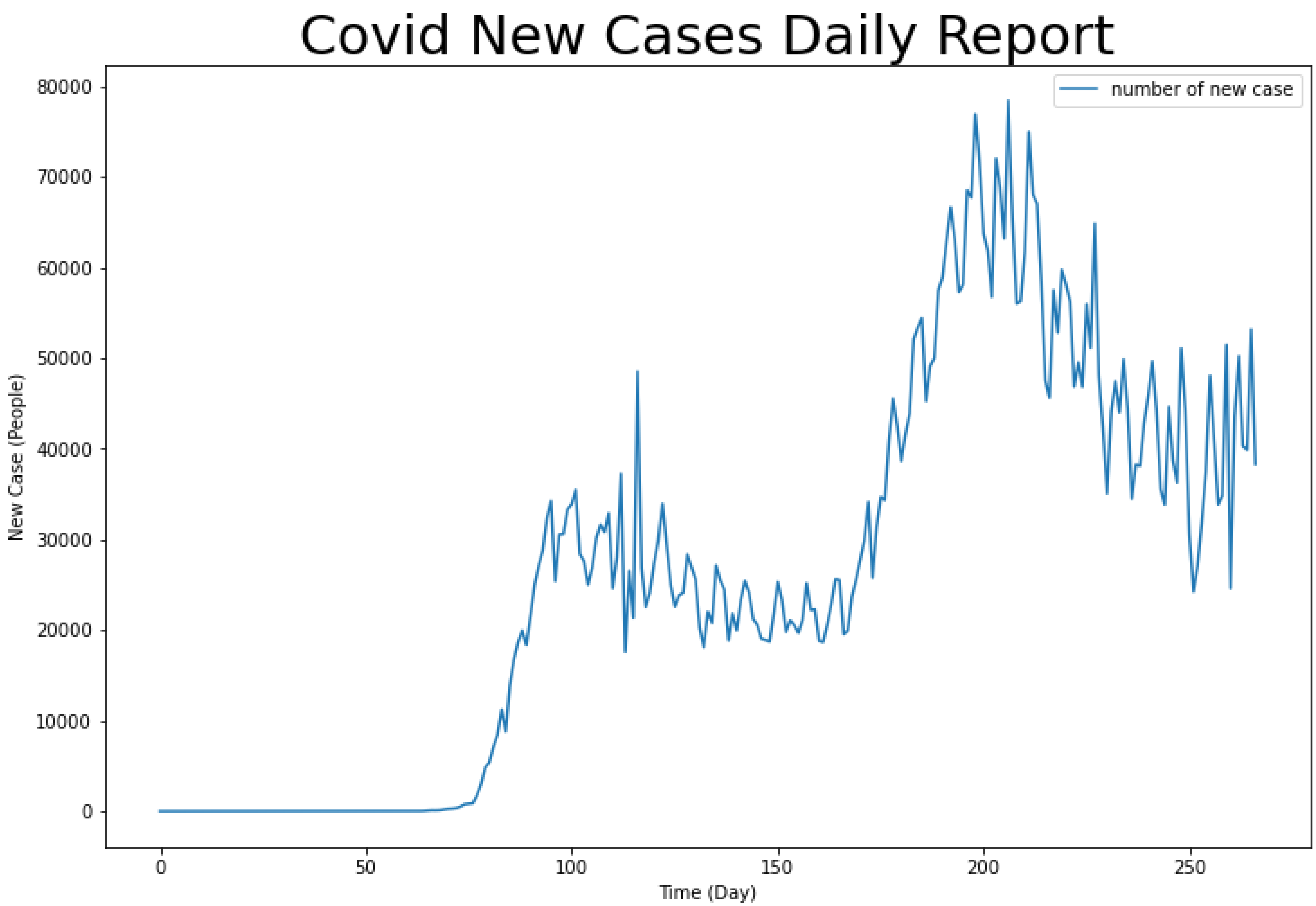
Extra: If you feel ambitious, try and figure out how you can make the x-axis in units of months instead of just number of days. You `plt.xticks()` could be useful for this because you can put strings for your x-axis tick marks.

In [6]:
```
with open("Covid_Data.csv") as f:
    ncols = len(f.readline().split(','))
data = np.loadtxt('Covid_Data.csv', delimiter = ",", unpack = True, skiprows = 1,usecols= range(1,ncols))
```

Part b) Now make a similar plot of New Cases (unsmoothed) vs. time (days). Dont forget to include your plot's title, axis labels (with units), and a legend.

In [7]:
```
#Initializing Figure
plt.figure(figsize = (12,8))
indices = list(range(len(data[1])))
plt.plot(indices,data[1],label = 'number of new case')

#Making it Nice
plt.title('Covid New Cases Daily Report', fontsize=30)
plt.ylabel('New Case (People)')
plt.xlabel('Time (Day)')
plt.legend()
```

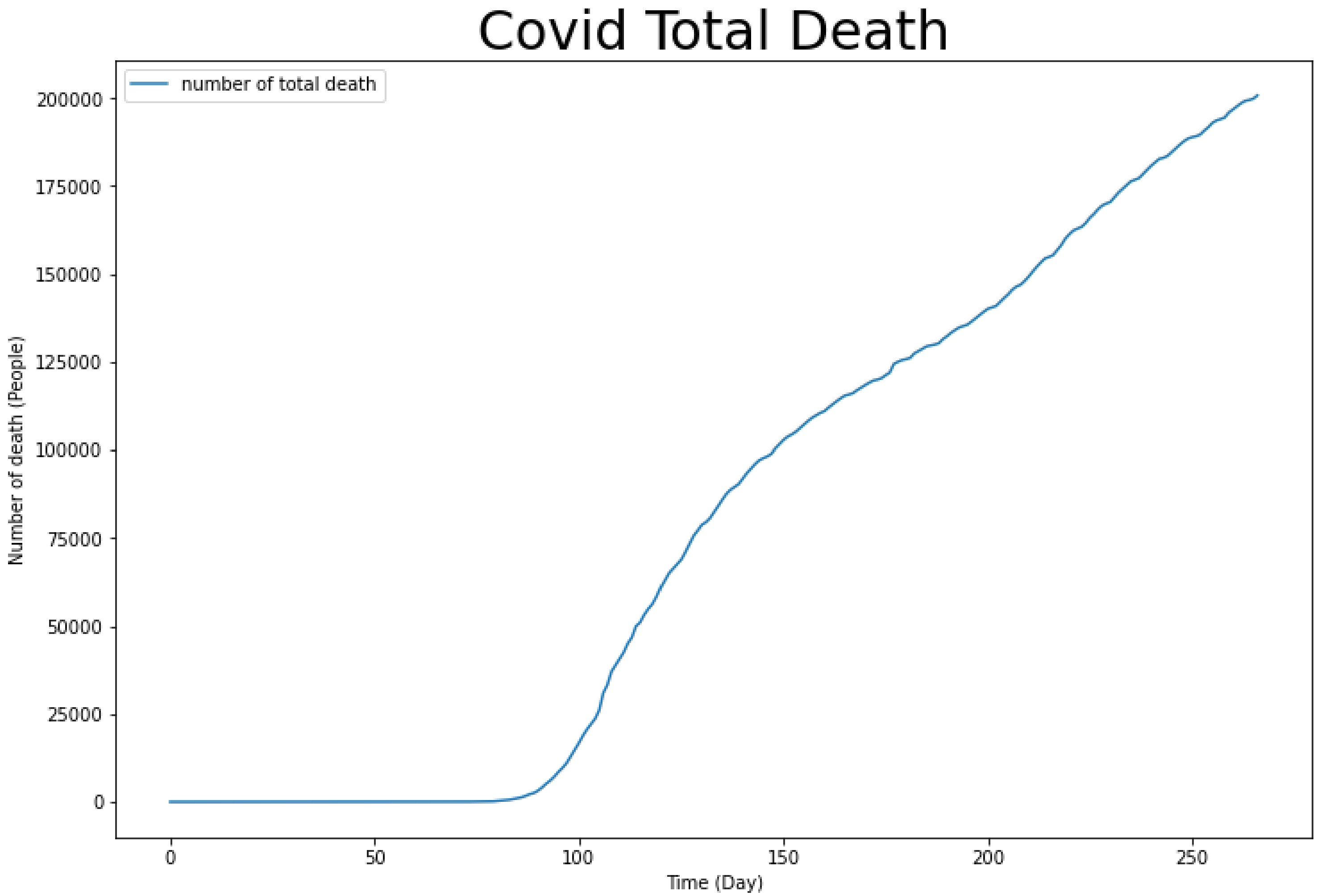
```python
#Display Plot
plt.show()
```



Part c) A crucial part of being a scientist is being able to look at a data set and find something interesting that they want to look at within a data set. Go ahead and open up the data set in some spreadsheet editor (I Use Microsoft Excel for example) and look at the titles for each column. Pick one of the columns that you find really interesting or that is relevant to show someone. Plot that vs time (days). As usual dont forget to add a title, labels, and a legend.

In [8]:
```python
'''I want to show how lethal Covid is by showing how fast the total death increased and in around which day since covid started that the total death start to grow rapidly'''
```

```
#making plot
plt.figure(figsize = (12,8))
indices = list(range(len(data[3]))) #using total_death column
plt.plot(indices,data[3],label = 'number of total death')

#Making it Nice
plt.title('Covid Total Death', fontsize=30)
plt.ylabel('Number of death (People)')
plt.xlabel('Time (Day)')
plt.legend()

#Display Plot
plt.show()
```



Curve Fitting

The following problem will be us exploring how to fit a set of data.

This is inherently very statistics heavy to fully understand. So, do not freak out if the rest of this cell doesn't make any sense to you right now. It freaked me out when I first looked at it but now that I have actually learned stats in my other courses it isn't too scary to read. I am just going to put it here for your reference in the future if you want to understand what is happening more deeply.

Note: The following information was provided and adapted from Physics 77

The simplest technique to describe is **least-squares fitting**. Usually you use the least-squares fit if **you have a set of data** (pairs of data points (x_i, y_i)), and **you want to describe it in terms of a model** $y(x;\{\theta_j\})$, where **you have parameters** $\{\theta_j\}$ **that are unknown**. The purpose of your fit is to determine the values of $\{\theta_j\}$ and (hopefully) their uncertainties. An example of a model is:

$$y = a_0 + a_1 x$$

where the unknown parameters θ_j are a_0 and a_1.

There are two standard cases where least-squares method is applicable:

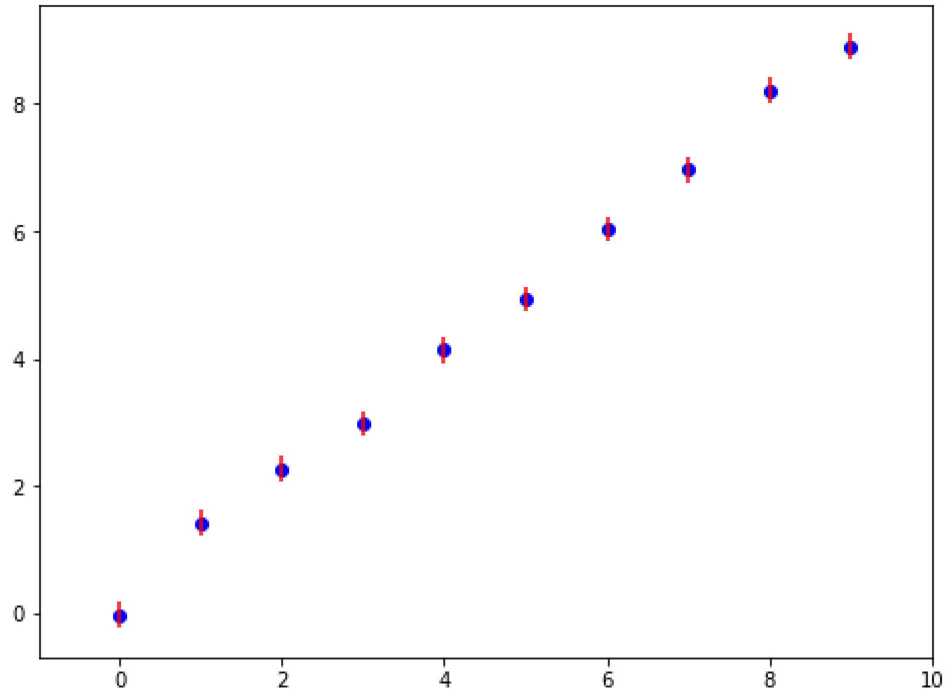
1. You know errors for each data point σ_i and you know that those errors are Gaussian. In this case, you minimize $\chi^2 = \sum\left(\frac{y_i - y(x_i;\theta)}{\sigma_i}\right)^2$ with respect to the parameters $\{\theta_j\}$. The value of the χ^2_{min} can be interpreted as a goodness-of-fit. The parameters $\{\theta_j\}$ that minimize χ^2 have probabilistic interpretation
2. You know that the errors are Gaussian and are the same for each data point, but you do not know their magnitude. In this case, you would minimize the sum of squares: $\mathcal{S} = \sum\left(y_i - y(x_i;\theta)\right)^2$. Then value of $\mathcal{S}$ can be used to *estimate* the errors σ_i for each data point: $\sigma_i = \sqrt{\mathcal{S}/(N_{\text{data}} - N_{\text{parameters}})}$ The errors on θ have a probabilistic definition, but you lose information about the goodness of fit
3. If the errors are not known to be Gaussian, then the least square method is not useful to estimate uncertainties or the goodness of fit. It is also not guaranteed to be unbiased or most efficient.

Let's try it out by fitting a straight line model to some data

First lets generate some fake data and plot it

In [9]:
```
import scipy.optimize as fitter

# Generate artificial data = straight line with a=0 and b=1
# plus some noise.
a0 = 0
b0 = 1
sig = 0.2
Npoints = 10
xdata = np.arange(0,Npoints,1.)
ydata = a0+xdata*b0+sig*np.random.standard_normal(size=Npoints)
sigma = np.ones(Npoints)*sig

plt.figure(figsize=(8,6))
plt.scatter(xdata,ydata,color='b')
```

```
plt.errorbar(xdata,ydata, sigma, color='r',ls='none')
plt.xlim(-1,Npoints)
plt.show()
```

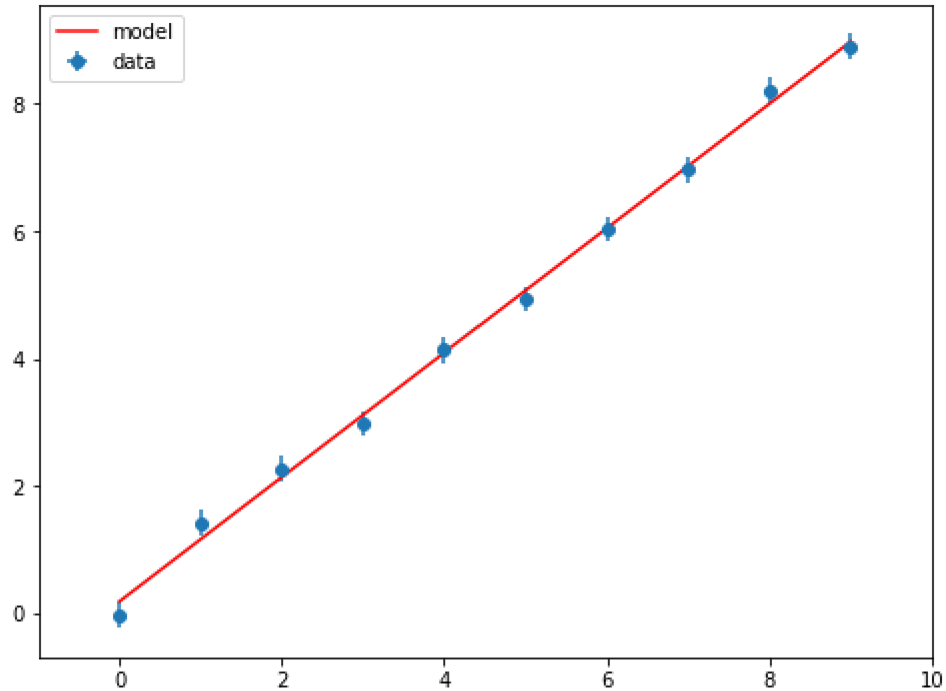


In [10]:

```
# Make sure you run the cell above this first to generate some artificial data
# to which we will fit this curve

#===============================================================================
# Define a fit model. For this part, we will use a linear function
# The function which defines your model HAS TO TAKE ON A SPECIFIC FORM
#===============================================================================

def model(x, a, b):
    return a + b*x

# You have to supply an initial guess of parameters, and they should be "close enough" to the true values, otherwise
# the fit may fall into a false minimum
par0    = np.array([0.5, -0.3]) # initial guess for parameters
par, cov = fitter.curve_fit(model, xdata, ydata, par0, sigma, absolute_sigma=True)

#===============================================================================
# the par arrays contains the values of parameters. cov is the covariance matrix
```

```
# decode it now
#===========================================================================================

print("Your model's parameters and their uncertainties are the following: \n")
a = par[0]
ea = np.sqrt(cov[0,0])
print('a={0:6.3f}+/-{1:5.3f} \n'.format(a,ea))
b = par[1]
eb = np.sqrt(cov[1,1])
print('b={0:6.3f}+/-{1:5.3f} \n\n'.format(b,eb))

#===========================================================================================
# compute reduced chi2
#===========================================================================================

print("Your model's chi^2 value and reduced chi^2 value and their uncertainties are the following: \n")
chi_squared = np.sum(((ydata - model(xdata, *par))/sigma)**2)
reduced_chi_squared = (chi_squared)/(len(xdata)-len(par))
print ('chi^2 = {0:5.2f} \n'.format(chi_squared))
print ('chi^2/d.f.={0:5.2f} \n'.format(reduced_chi_squared))

#===========================================================================================
# overlay plot over data
#===========================================================================================

plt.figure(figsize=(8,6))
plt.errorbar(xdata, ydata, xerr=0, yerr=sigma, fmt='o', label='data') #plotting the data
plt.xlim(-1,Npoints)
xfit = np.linspace(0,Npoints-1.,50)
plt.plot(xfit,model(xfit,par[0],par[1]),'r-', label='model') #plotting the model
plt.legend()
plt.show()
```

```
Your model's parameters and their uncertainties are the following:

a= 0.184+/-0.118

b= 0.977+/-0.022


Your model's chi^2 value and reduced chi^2 value and their uncertainties are the following:

chi^2 =  5.44

chi^2/d.f.= 0.68
```



In summary:

1. for curve fitting you need to write a model function that you think the data should fit based on some unknown parameters
2. you need to make an array full of guesses for each unknown parameter
3. use `scipy.optimize.curve_fit` function to fit the model's paramters to best match the data
4. This function spits out two things, the parameters array which it's length depends on how many parameters your model needs. The second thing is the Covariance 2D array which the square root of the diagonals correspond to the uncertainties of each

parameter
5. We now can plug these found parameters into our model and plot to see how well the model fits the data points

Problem 3 (25 Points)

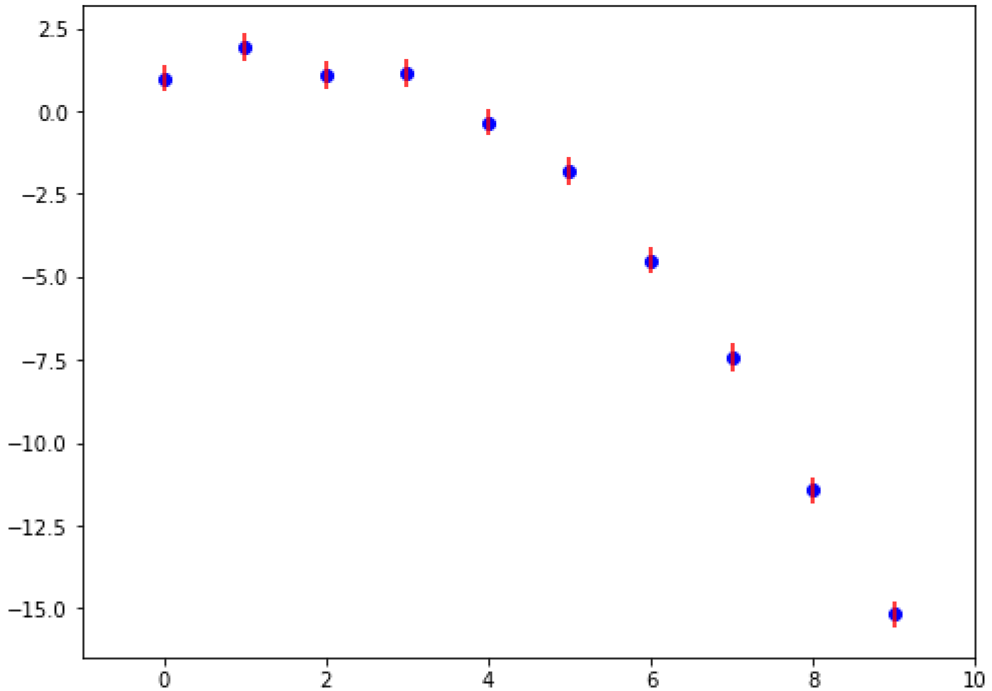
We now introduced a new function: `scipy.optimize.curve_fit()`. The code in the cell immediately below will generate some data where the first column is x values, the second column is the y values, the third column is the uncertainty in each value. Use the techniques above to fit a quadratic model of the form

$$y = a_0 + a_1 x + a_2 x^2$$

Plot the data and your best fit curve with error and print out the values and their uncertainties as we did above.

In [16]:
```
# Generate artificial data = quadratic function with a0 = .5, a1=1, a2 = -0.3
# plus some noise.
a0 = 0.5
a1 = 1
a2 = -0.3
sig = 0.4
Npoints = 10

xdata = np.arange(0,Npoints,1.)
ydata = a0 + a1 * xdata + a2 * xdata **2 + sig * np.random.standard_normal(size=Npoints)
sigma = np.ones(Npoints)*sig

print("Your x data values are stored as xdata")
print("Your y data values are stored as ydata")
print("Your y data uncertainties are stored as sigma")

plt.figure(figsize=(8,6))
plt.scatter(xdata,ydata,color='b')
plt.errorbar(xdata,ydata, sigma, color='r',ls='none')
plt.xlim(-1,Npoints)
plt.show()
```

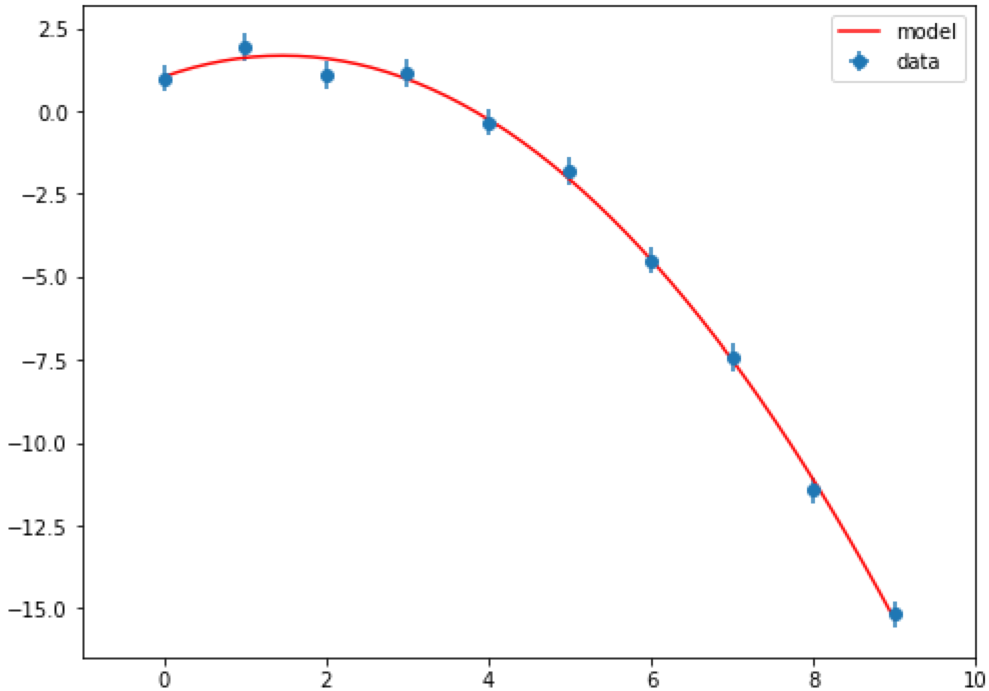
```
Your x data values are stored as xdata
Your y data values are stored as ydata
Your y data uncertainties are stored as sigma
```



In [28]:
```
def model3(x, a, b,c):
    return a + b*x + c*(x**2)

# You have to supply an initial guess of parameters, and they should be "close enough" to the true values, otherwise
# the fit may fall into a false minimum
par0    = np.array([1, 1,1]) # initial guess for parameters
par, cov = fitter.curve_fit(model3, xdata, ydata, par0, sigma, absolute_sigma=True)

#==========================================================================================
# the par arrays contains the values of parameters. cov is the covariance matrix
# decode it now
#==========================================================================================

print("Your model's parameters and their uncertainties are the following: \n")
a = par[0]
ea = np.sqrt(cov[0,0])
print('a={0:6.3f}+/-{1:5.3f} \n'.format(a,ea))
b = par[1]
eb = np.sqrt(cov[1,1])
print('b={0:6.3f}+/-{1:5.3f} \n\n'.format(b,eb))
c = par[2]
eb = np.sqrt(cov[2,2])
```

```python
print('c={0:6.3f}+/-{1:5.3f} \n\n'.format(c,eb))

#==========================================================================================
# compute reduced chi2
#==========================================================================================

print("Your model's chi^2 value and reduced chi^2 value and their uncertainties are the following: \n")
chi_squared = np.sum(((ydata - model3(xdata, *par))/sigma)**2)
reduced_chi_squared = (chi_squared)/(len(xdata)-len(par))
print ('chi^2 = {0:5.2f} \n'.format(chi_squared))
print ('chi^2/d.f.={0:5.2f} \n'.format(reduced_chi_squared))

#==========================================================================================
# overlay plot over data
#==========================================================================================

plt.figure(figsize=(8,6))
plt.errorbar(xdata, ydata, xerr=0, yerr=sigma, fmt='o', label='data') #plotting the data
plt.xlim(-1,Npoints)
xfit = np.linspace(0,Npoints-1.,50)
plt.plot(xfit,model3(xfit,par[0],par[1],par[2]),'r-', label='model') #plotting the model
plt.legend()
plt.show()
```

```
Your model's parameters and their uncertainties are the following:

a= 1.037+/-0.314

b= 0.877+/-0.163


c=-0.300+/-0.017


Your model's chi^2 value and reduced chi^2 value and their uncertainties are the following:

chi^2 =  3.70

chi^2/d.f.= 0.53
```



Root Finding

The following problem will be us exploring how to find the roots of a function numerically.

Problem 4 (25 Points)

[Adapted from Newman 6.15]

Consider a sixth-order polynomial

$$P(x) = 924x^6 - 2772x^5 + 3150x^4 - 1680x^3 + 420x^2 - 42x + 1$$

There is no general formula for the roots of a polynomial of degree 6, but you can compute the roots numerically.

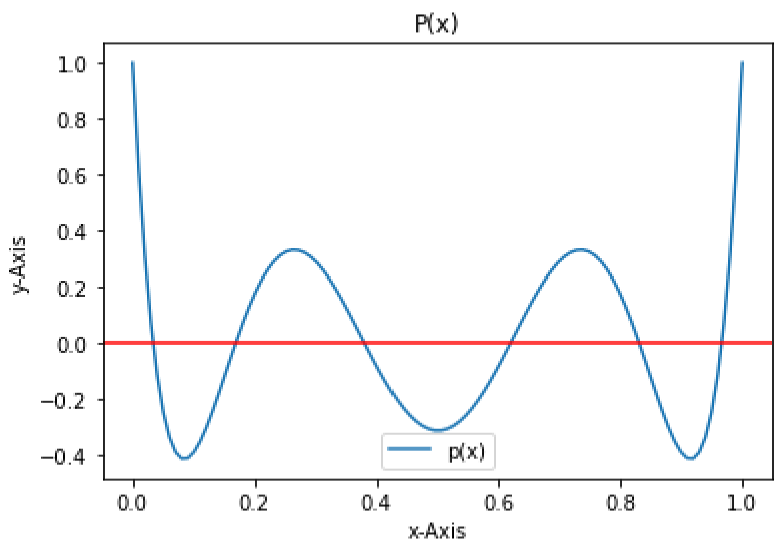
1. Make a plot of $P(x)$ from $x = 0$ to $x = 1$ and by inspecting it find rough values for the six roots of the polynomial.

2. Write the code to solve for the positions of all six roots to at least ten decimal places using at least one of the methods dsicussed in class. (you can/should use the built-in functions).

Hint: I would recommend using `fsolve` in the scipy library

In [12]:
```
from scipy.optimize import fsolve
x = np.linspace(0,1,100)
p = 924*x**6 - 2772*x**5 + 3150*x**4 - 1680 *x**3 + 420*x**2 -42*x +1

#p plot
plt.plot(x,p, label= 'p(x)') #need to put label here for legend later
plt.axhline(y=0, color = 'red') #make it easier to see y=0

#Title
plt.title("P(x)")

#Axis Label
plt.xlabel("x-Axis")
plt.ylabel("y-Axis")

#Legend
plt.legend()
```

Out[12]:
```
<matplotlib.legend.Legend at 0x1d1af671340>
```



From the graph, we can see the rough values for the six roots of the polynomial is at around x = 0.05, 0.2, 0.4, 0.6, 0.8, 1.0

In [13]:

```python
# Using fsolve
x = np.linspace(0,1,100)
def p(x):
    p = 924*x**6 - 2772*x**5 + 3150*x**4 - 1680 *x**3 + 420*x**2 -42*x +1
    return p
p = fsolve(p, x0 = [0.05,0.2,0.4,0.6,0.8,1.0])
list(map('{:2.12f}'.format,p))
```

Out[13]:

```
['0.033765242892',
 '0.169395306782',
 '0.380690406963',
 '0.619309593037',
 '0.830604693217',
 '0.966234757103']
```