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

#Include path that tells python what folder to check for the module
import sys
sys.path.append('/Users/Luke/Desktop/Advanced Laboratories I & II/Adv Labs 2/My Lab Module')
#Import my module
import MyModule as mod
```

In [2]:
```
#the help() function is very useful for checking single functions:
help(mod.linear_best_fit)
```

```
Help on function linear_best_fit in module MyModule:

linear_best_fit(xarray, yarray, yerrs, init_estimate_m, init_estimate_c, title='Linear Fit', xlabel='xlabe
l', ylabel='ylabel', precision=3)
    Uses scipy.optimize.curvefit() to fit a straight line to the data.
    Must use np arrays. "precision' is the number of decimal places.
```

In [3]:
```
#or use help() for checking the entire module!
#it includes a list of all the functions, as well as their arguments
help(mod)
```

```
Help on module MyModule:

NAME
    MyModule

FUNCTIONS
    exponential(xarray, yarray, xerrs, yerrs, init_estimate, title, xlabel, ylabel, precision)

    lin_chi_sqr(x, y, parameter_array, y_errs)
        #Linear Chi Square

    lin_reduced_chi_sqr(x, y, parameter_array, y_errs)
        #Linear Reduced Chi Square

    linear(xarray, yarray, xerrs, yerrs, init_estimate, title, xlabel, ylabel, precision)
        #WODR section (Weighted Orthogonal Distance Regression)
        #wodr is the general call, and then executes either linear (default), quadratic or
        #exponential depending on given argument

    linear_best_fit(xarray, yarray, yerrs, init_estimate_m, init_estimate_c, title='Linear Fit', xlabel='xla
bel', ylabel='ylabel', precision=3)
        Uses scipy.optimize.curvefit() to fit a straight line to the data.
        Must use np arrays. "precision' is the number of decimal places.

    longtable_2f(array_of_arrays)
        #Used to change python arrays into a latex table format

    quad_best_fit(xarray, yarray, yerrs, init_estimate_a2, init_estimate_m, init_estimate_c, title="Quadrati
c Fit", xlabel="xlabel", ylabel="ylabel", precision=3)
        Uses scipy.optimize.curvefit() to fit a quadratic curve to the data.
        Must use np arrays. "precision' is the number of decimal places.

    quad_chi_sqr(x, y, parameter_array, y_errs)
        #Quadratic Chi Square

    quad_reduced_chi_sqr(x, y, parameter_array, y_errs)
        #Quadratic Reduced Chi Square

    quadratic(xarray, yarray, xerrs, yerrs, init_estimate, title, xlabel, ylabel, precision)

    wodr(xarray, yarray, xerrs, yerrs, init_estimate, function='linear', title="Title", xlabel='xlabel', yla
bel="ylabel", precision=3)
        Using the Matrix Approach with the Weighted Orthogonal Distance Regression method of finding lines o
f best fit.

FILE
    c:\users\luke\desktop\advanced laboratories i & ii\adv labs 2\my lab module\mymodule.py
```

Notes when using this module

(Please read me, the most important points are in the first 2 bullet points and I swear it's useful)

This module was created by Luke Kilmartin to make data analysis faster and easier. :D
If this helped you, bonus points if you credit me in your lab report! Plagiarism isn't cool but referencing is. :)

- Make sure that `sys.path.append('...')` in the preamble actually points to your folder containing *MyModule.py* (Bracket direction is important here! It should look like this `->`.)
- Most of the functions here have useful arguments such as `title='Plot Title'` or `xlabel='X axis label'`. Please use `help(mod.function_name)` to see a list of all the arguments for a function. If an argument looks like `title="Plot Title"`, then that is the default argument, and you can call the function without specifying that argument if you want.

- *MyModule.py* can be easily modified by opening it with Notepad++ or any similar editor.
- I am considering putting this onto GitHub and making it a collaborative effort between our year group if anyone wants to add anything to it.
- If you're making changes to the module, saving the *MyModule.py* file, and then trying to run the (changed) module in your current session of Jupyter Notebook, you have to either do:

  - "Kernel -> Restart" or
  - "Kernel -> Restart and Clear Output" or
  - "Kernel -> Restart and Run All"

  between your changes, and remember to re-run the preamble (the imports cell) before running anything else.

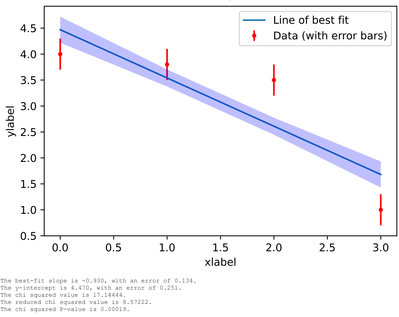
Linear curve_fit()

This method acts as if you only have errors in the y-axis.
(using matrix methods to propagate errors - shown by John in 3rd year)

Prints the best-fit parameters with errors (number of decimal points can be changed using the "precision=" argument).
Prints chi-square values for a linear best fit.

Optional arguments: title='Linear Fit', xlabel='xlabel', ylabel='ylabel', precision=3

In [4]:
```
x=np.array([0,1,2,3])
y=np.array([4,3.8,3.5,1])
xerr=np.ones(4)*.5
yerr=np.ones(4)*.3

mod.linear_best_fit(x,y,yerr,-.9,4.1,precision=3)
```




```
The best-fit slope is -0.930, with an error of 0.134.
The y-intercept is 4.470, with an error of 0.251.
The chi squared value is 17.14444.
The reduced chi squared value is 8.57222.
The chi squared P-value is 0.00019.
```

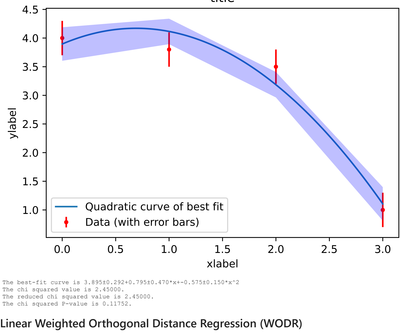
Quadratic curve_fit()

This method acts as if you only have errors in the y-axis. (using matrix methods to propagate errors - shown by John in 3rd year)

Prints the best-fit parameters with errors (number of decimal points can be changed using the "precision=" argument).
Prints chi-square values for a quadratic best fit.

Optional arguments: title='Quadratic Fit', xlabel='xlabel', ylabel='ylabel', precision=3

In [15]:
```
mod.quad_best_fit(x,y,yerr,0,-.9,4.1)
```




```
The best-fit curve is 3.895±0.292*0.795±0.470*x+0.575±0.150*x^2
The chi squared value is 2.45000.
The reduced chi squared value is 2.45000.
The chi squared P-value is 0.11752.
```

Linear Weighted Orthogonal Distance Regression (WODR)

This method acts as if you have errors in the x-axis AND the y-axis.
I'm fairly sure this method doesn't work if you have errors of zero in one axis, but I haven't tested it.
(uses matrix methods to propagate errors - shown by John in 3rd year)

Optional arguments:
function='linear', title='Title', xlabel='xlabel', ylabel='ylabel', precision=3

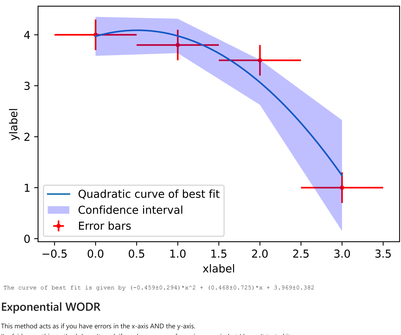
In [ ]:
```
x=np.array([0,1,2,3])
y=np.array([4,3.8,3.5,1])
xerr=np.ones(4)*.5
yerr=np.ones(4)*.3

mod.wodr(x,y,xerr,yerr,np.array([-.9,4.1]),function='linear')
```

Quadratic WODR

This method acts as if you have errors in the x-axis AND the y-axis.
I'm fairly sure this method doesn't work if you have errors of zero in one axis, but I haven't tested it.
(uses matrix methods to propagate errors - shown by John in 3rd year)

Optional arguments:
function='linear', title='Title', xlabel='xlabel', ylabel='ylabel', precision=3

In [13]:
```
mod.wodr(x,y,xerr,yerr,np.array([0,-.9,4.1]),function='quadratic')
```




```
The curve of best fit is given by (-0.459±0.294)*x^2 + (0.468±0.725)*x + 3.969±0.382
```

Exponential WODR

This method acts as if you have errors in the x-axis AND the y-axis.
I'm fairly sure this method doesn't work if you have errors of zero in one axis, but I haven't tested it.
(uses matrix methods to propagate errors - shown by John in 3rd year)

Optional arguments:
function='linear', title='Title', xlabel='xlabel', ylabel='ylabel', precision=3

In [ ]:
```
mod.wodr(x,y,xerr,yerr,np.array([-1,1,4]),function='exponential')
```

Using a self-defined function to get arrays for a LaTeX table:

Note that it is fairly easy to generalise this to precisions other than .2f, but I'm not bothered at the moment.

In [ ]:
```
#Initialising 3 example numpy arrays
array1=np.array([1,2,3,4,5])
array2=np.array([2,4,6,8,10])
array3=np.array([1,4,9,16,25])
print(array1,array2,array3)
```

In [ ]:
```
#This results in First array - First column

array_of_arrays_1_2_and_3=np.array([array1,array2,array3]) #unnecessarily long variable name for teaching pu

#calling the function
mod.longtable_2f(array_of_arrays_1_2_and_3)

#Then just copy and paste this into your LaTeX table and you're golden
```