Student Names and IDs:

- Newton Kwan, nk150

Problem 1.1

Spell out the system

$$A\mathbf{c} = \mathbf{a}$$

for this special case.

Solution 1.1

$$c_0 1 + c_1 x_0 = y_0$$

Problem 1.2

Give expressions for two different possible solutions $\mathbf{c}$ to this equation in terms of x_0 and y_0.

Solution 1.2

$$c' = [y_0, 0]^T$$

$$c'' = [y_0(1 - x_0), y_0]^T$$

Problem 1.3

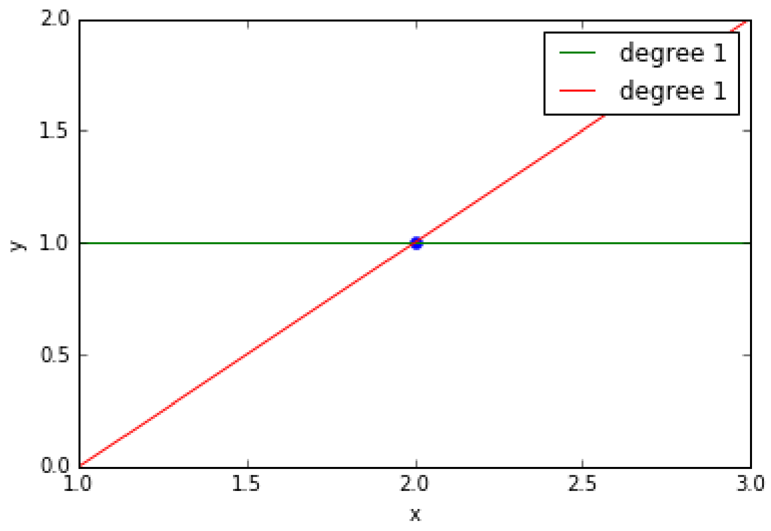
Write and run Python code to draw, in a single plot, the two lines corresponding to the two solutions you gave when

$$(x_0, y_0) = (2, 1).$$

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt

        def show(x, y, cList = []):
            plt.ion()
            plt.plot(x, y, marker='.', markersize=12, ls='')
            npt = 100
            xrange = [x - 1, x + 1] if x.size == 1 else [np.amin(x), np.amax(x)]
            xFine = np.linspace(xrange[0], xrange[1], npt)
            for c in cList:
                nc = c.size
                ycFine = np.zeros(xFine.shape)
                xPow = np.ones(xFine.shape)
                for i in range(nc):
                    ycFine += c.item(i) * xPow
                    xPow *= xFine
                plt.plot(xFine, ycFine, label = 'degree ' + str(nc-1))
            plt.xlabel('x')
            plt.ylabel('y')
            plt.legend()
            plt.show()

        # Usage example
        p = [2, 1]
        show(np.array(p[0]), np.array(p[1]), [np.array([1, 0]), np.array([-1,1
        ])])
        #show(np.array(p[0]), np.array(p[1]), [np.array([1, 1])])
```



Problem 2.1

Derive this last equation from the system

$$A\mathbf{c} = \mathbf{a}$$

spelled out for this special case.

Solution 2.1

Given the initial equations

$$c_0 + c_1 x = y$$
$$A\mathbf{c} = \begin{bmatrix} 1 & x_0 \\ 1 & x_1 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 1 \end{bmatrix} = \mathbf{a}$$

Using Gaussian elimination,

$$\left[\begin{array}{cc|c} 1 & x_0 & y_0 \\ 1 & x_1 & y_1 \end{array}\right] \rightarrow \left[\begin{array}{cc|c} 1 & x_0 & y_0 \\ 0 & x_1 - x_0 & y_1 - y_0 \end{array}\right]$$

This gives us the following two equations:

$$c_0 + c_1 x_0 = y_0$$
$$c_0(x_1 - x_0) = y_1 - y_0$$

Rearranging the equations for c_0 and c_1:

$$c_0 = y_0 - c_1 x_0$$
$$c_1 = \frac{y_1 - y_0}{x_1 - x_0}$$

Substituting the second equation into the first equation:

$$c_0 = y_0 - \frac{y_1 - y_0}{x_1 - x_0} x_0$$

Substituting c_0 and c_1 into the first equation:

$$y_0 - \frac{y_1 - y_0}{x_1 - x_0} x_0 + \frac{y_1 - y_0}{x_1 - x_0} x = y$$

Rearranging,

$$y - y_0 = \frac{y_1 - y_0}{x_1 - x_0}(x - x_0)$$

Problem 3.1

> Write the matrix $A^T A$ and vector $A^T \mathbf{a}$ in terms of N, X, Y, S, P for the special case $k = 1$.

Solution 3.1

$$A^T A = \begin{bmatrix} 1 & 1 & 1 \\ x_0 & x_1 & x_2 \end{bmatrix} \begin{bmatrix} 1 & x_0 \\ 1 & x_1 \\ 1 & x_2 \end{bmatrix} = \begin{bmatrix} N & X \\ X & S \end{bmatrix}$$

$$A^T \mathbf{a} = \begin{bmatrix} 1 & 1 & 1 \\ x_0 & x_1 & x_2 \end{bmatrix} \begin{bmatrix} y_0 \\ y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} Y \\ P \end{bmatrix}$$

Problem 3.2

Find expressions for c_0 and c_1 in terms of N, X, Y, S, P by solving the normal equations. Recall that

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}^{-1} = \frac{1}{ad - bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

Solution 3.2

First, we solve for $\mathbf{a}$,

$$A^T A\mathbf{c} = A^T\mathbf{a}$$

Multiplying both sides by the inverse of $A^T A$

$$(A^T A)^{-1}(A^T A)\mathbf{c} = (A^T A)^{-1}A^T\mathbf{a}$$
$$\mathbf{c} = (A^T A)^{-1}A^T\mathbf{a}$$

Using the definition of the the inverse of a 2x2 matrix,

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}^{-1} = \frac{1}{ad-bc}\begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

$$(A^T A)^{-1} = \begin{bmatrix} N & X \\ X & S \end{bmatrix}^{-1} = \frac{1}{NS-X^2}\begin{bmatrix} S & -X \\ -X & N \end{bmatrix}$$

Substituting into the equation above,

$$\mathbf{c} = \begin{bmatrix} c_0 \\ c_1 \end{bmatrix} = \frac{1}{NS-X^2}\begin{bmatrix} S & -X \\ -X & N \end{bmatrix}\begin{bmatrix} Y \\ P \end{bmatrix} = \frac{1}{NS-X^2}\begin{bmatrix} SY-XP \\ -XY+NP \end{bmatrix}$$

Problem 3.3

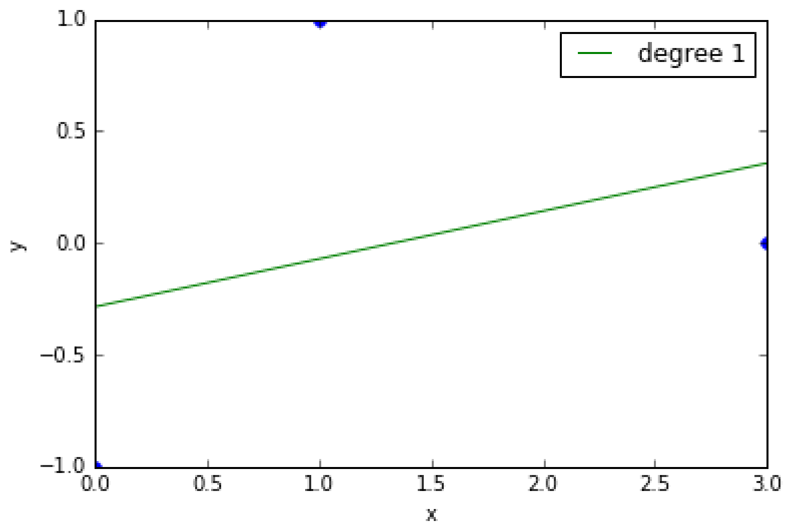
Use the function `show` given earlier to display the three points $(0,-1),(1,1),(3,0)$ and the line fit to them with the formulas you just found for c_0 and c_1.

```
In [2]: import numpy as np
        import matplotlib.pyplot as plt

        def show(x, y, cList = []):
            plt.ion()
            plt.plot(x, y, marker='.', markersize=12, ls='')
            npt = 100
            xrange = [x - 1, x + 1] if x.size == 1 else [np.amin(x), np.amax(x)]
            xFine = np.linspace(xrange[0], xrange[1], npt)
            for c in cList:
                nc = c.size
                ycFine = np.zeros(xFine.shape)
                xPow = np.ones(xFine.shape)
                for i in range(nc):
                    ycFine += c.item(i) * xPow
                    xPow *= xFine
                plt.plot(xFine, ycFine, label = 'degree ' + str(nc-1))
            plt.xlabel('x')
            plt.ylabel('y')
            plt.legend()
            plt.show()

        # Usage example
        p = [0, -1, 1, 1, 3, 0]
        N = 3
        X = 4
        Y = 0
        S = 10
        P = 1
        coeff = 1 / ((N*S) - (X**2))
        c0 = coeff * (S*Y - X*P)
        c1 = coeff * (-X*Y + N*P)
        print("c0 = {:.3f}".format(c0))
        print("c1 = {:.3f}".format(c1))
        show(np.array([p[0], p[2], p[4]]), np.array([p[1], p[3], p[5]]), [np.arr
        ay([c0, c1])])
        #show(np.array(p[0]), np.array(p[1]), [np.array([1, 1])])
```

```
c0 = -0.286
c1 = 0.214
```



Problem 4.1

> In problem 1.3 you found two different solutions to a data fitting problem. Let us call the corresponding coefficient vectors $\mathbf{c}$ and $\mathbf{d}$. Write the values of $\mathbf{c}$ and $\mathbf{d}$, and report their norms. Which is smaller?

Solutions 4.1

```
In [3]: c = np.array([1, 0])
        d = np.array([-1,1])

        c_norm = np.linalg.norm(c)
        d_norm = np.linalg.norm(d)

        print("c = ", c)
        print("d = ", d)
        print("The norm of c is {:.3f}".format(c_norm))
        print("The norm of d is {:.3f}".format(d_norm))
        print("The norm of c is smaller than the norm of d")
```

```
c =  [1 0]
d =  [-1  1]
The norm of c is 1.000
The norm of d is 1.414
The norm of c is smaller than the norm of d
```

Problem 4.2

> Write a function with header
>
> `def fit(x, y, k):`
>
> that fits a polynomial of degree `k` to the data in `x` and `y`.

Solutions 4.1

```
In [4]: def fit(x,y,k):
            '''
            This function takes `numpy` vectors `x` and `y` with $N$ coordinates
         each
            and a degree `k` (a nonnegative integer), and returns a `numpy` vect
        or `c`
            with the coefficients of the polynomial of degree `k` that fits
            the points `(x[n], y[n])` best in the Least-Squares sense.
            '''

            N = len(y)
            arr = np.zeros((N, k+1))
            for i in range(N):
                for j in range(k+1):
                    arr[i, j] = x.item(i) ** j

            A = np.matrix(arr)
            a = np.reshape(y, [y.size, 1])

            c = np.linalg.lstsq(A, a, rcond=None)[0]
            return c
```

Problem 4.3

Show the plot and output obtained with the following code, where `fit` is your function. The file `T.txt` should be in the same directory as your notebook.

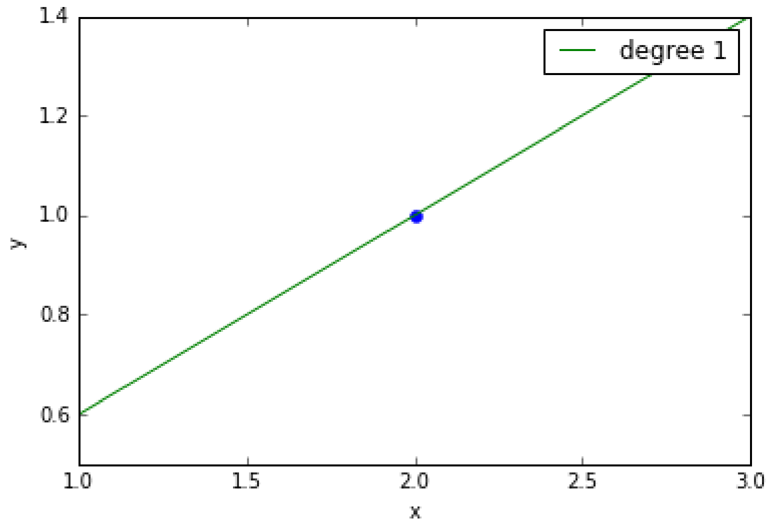
Solutions 4.3

In [5]:
```
try:
    p13 = {'x' : np.array(2), 'y' : np.array([1]), 'k' : [1]}

    np.set_printoptions(formatter={'float': '{: .3f}'.format})

    def run(p):
        x, y = p['x'], p['y']
        cList = [fit(x, y, k) for k in p['k']]
        show(x, y, cList)
        for c in cList:
            print(c.T)

    run(p13)
except NameError:
    pass
```



```
[[ 0.200  0.400]]
```

Problem 4.4

> What is the approximate norm of the solution (to three decimal digits), and is this result consistent with your results in problem 1.3? Why or why not?

Solutions 4.4

In [6]:
```
c = np.array([0.2, 0.4])
c_norm = np.linalg.norm(c)

print("c = ", c)
print("The norm of c is {:.3f}".format(c_norm))
print("The results are not consistent with problem 1.3. The inconsistenc
y arises because we did not use Least Squares in 1.3, "
      + "picking a single solution of many. In 4.3, we picked the soluti
on using Least Squares, giving"
     + " us the answer with the lowest risk. ")
```

```
c =  [ 0.200  0.400]
The norm of c is 0.447
The results are not consistent with problem 1.3. The inconsistency aris
es because we did not use Least Squares in 1.3, picking a single soluti
on of many. In 4.3, we picked the solution using Least Squares, giving
us the answer with the lowest risk.
```

Problem 4.5

a. Show the plots and outputs obtained with the code below, and answer the following questions:

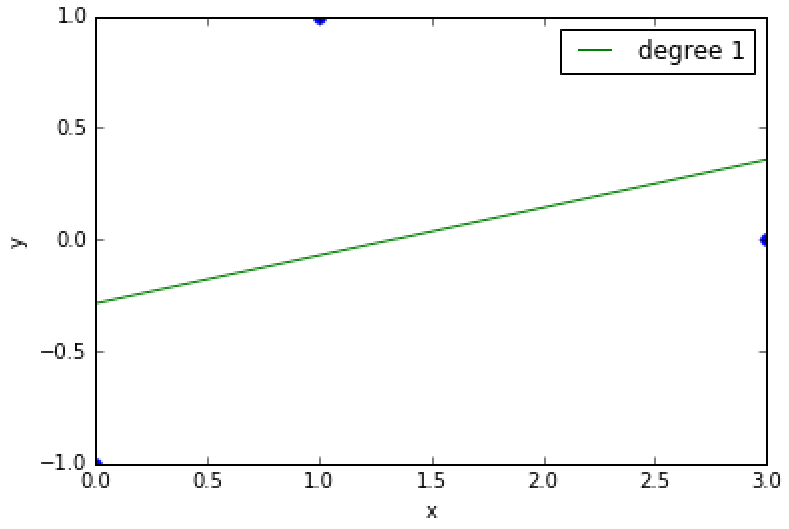
b. Are the plot and value of **c** from the data in `p33` consistent with your answer to problem 3.3?

c. Give a brief qualitative description of the way(s) in which the curves in the plot from the data in `notes` differ from the curves in Figure 1 of the class notes on data fitting (https://www2.cs.duke.edu/courses/fall18/compsci371d/notes/01_DataFitting.pdf). The plots in that Figure were computed from the same ten points as in `T.txt` but with a different numerical package.

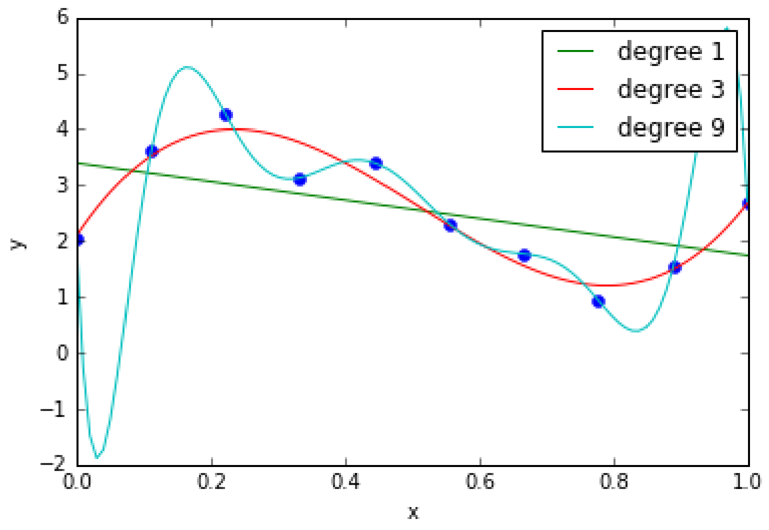
```
In [7]: try:
            p33 = {'x' : np.array([0, 1, 3]), 'y' : np.array([-1, 1, 0]), 'k' :
        [1]}

            x, y = np.loadtxt('T.txt', unpack=True)
            notes = {'x' :x, 'y' : y, 'k' : [1, 3, 9]}

            run(p33)
            run(notes)
        except NameError:
            pass
```



```
[[-0.286  0.214]]
```



```
[[ 3.394 -1.655]]
[[ 2.089  18.049 -49.900  32.440]]
[[ 2.030 -292.804  7046.148 -62434.420  285853.947 -759114.988
   1214741.224 -1155128.463  601089.629 -131759.602]]
```

Solutions 4.5

b) The plot and value of c are consistent with question 3.3
c) The graphs from Figure 1 and the plots shown above look identical.

Problem 4.6 (Extra Credit, 5 Points)

> If you noticed any discrepancies between your plot and the Figure in the class notes, first state why the discrepancies should not occur under ideal circumstances. Then suggest some reasons for them. For your information, the coefficients found for $k = 9$ for the Figure in the class notes are as follows:
>
> $$2.030, -295.642, 7107.144, -62956.114, 288218.499, -765400.622, 1224862.633, -1164835.634$$

Solutions 4.6

I did not find any visual discrepancies between my plots and the Figure in the class notes. But, I can see that the coefficients are differnet. With that said, there are a few reasons I think that the numbers are slightly different:
1) differences in implentation within numerical packages, which can come from the liberty of the programmers and scientists who create them. One example may be that one scientist writing code for numpy might see a particular optimization as more beneficial over another kind, where as another scientist may think the opposite or take liberties elsewhere. I can imagine that these decisions are guided by how the packages are intended to be used by its users;
2) Operating systems varies from computer to computer, and it is possible that a machine running linux vs OSX may produce slightly different outputs;
3) General hardware and limitations of computing: these include how many decimals places python and your computer can store accurately and how much error there will be once you go past something like 13 decimal places. This is why we don't test if a float is equal to another float because they often have trailing digits that don't mean anything after a certain decimal places.