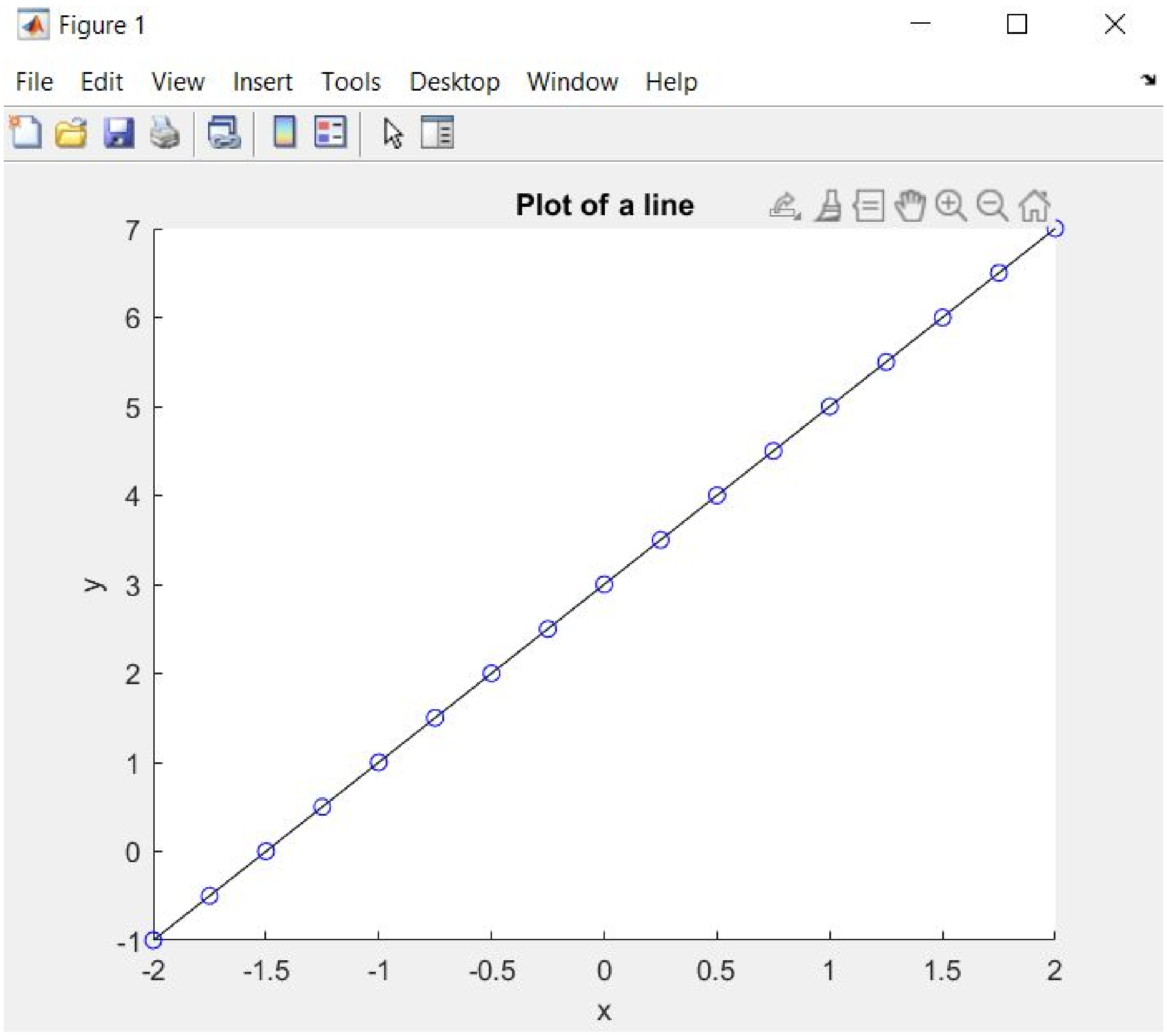
Andrew Garwood
Question 1:

Figure 1 from given code:



(a) Which line of code creates the black line? Which line of code creates the blue circles?
The line of code that creates the black line is: plot(x, y, 'k')
The line of code that creates the blue circles is: plot(x, y, 'bo')

(b) Explain in your own words what the blue circles in the plot represent?
The blue circles on the plot represent points on the graph. Specifically, the x values, are all numbers in the set {x modulo 0.25 = 0 | -2 <= x <= 2}; sequential x values of the blue circles are separated by a magnitude of 0.25. The y values result from plugging x into the function y = 2x + 3; it is noteworthy to point out that sequential y values have an absolute difference of 0.5

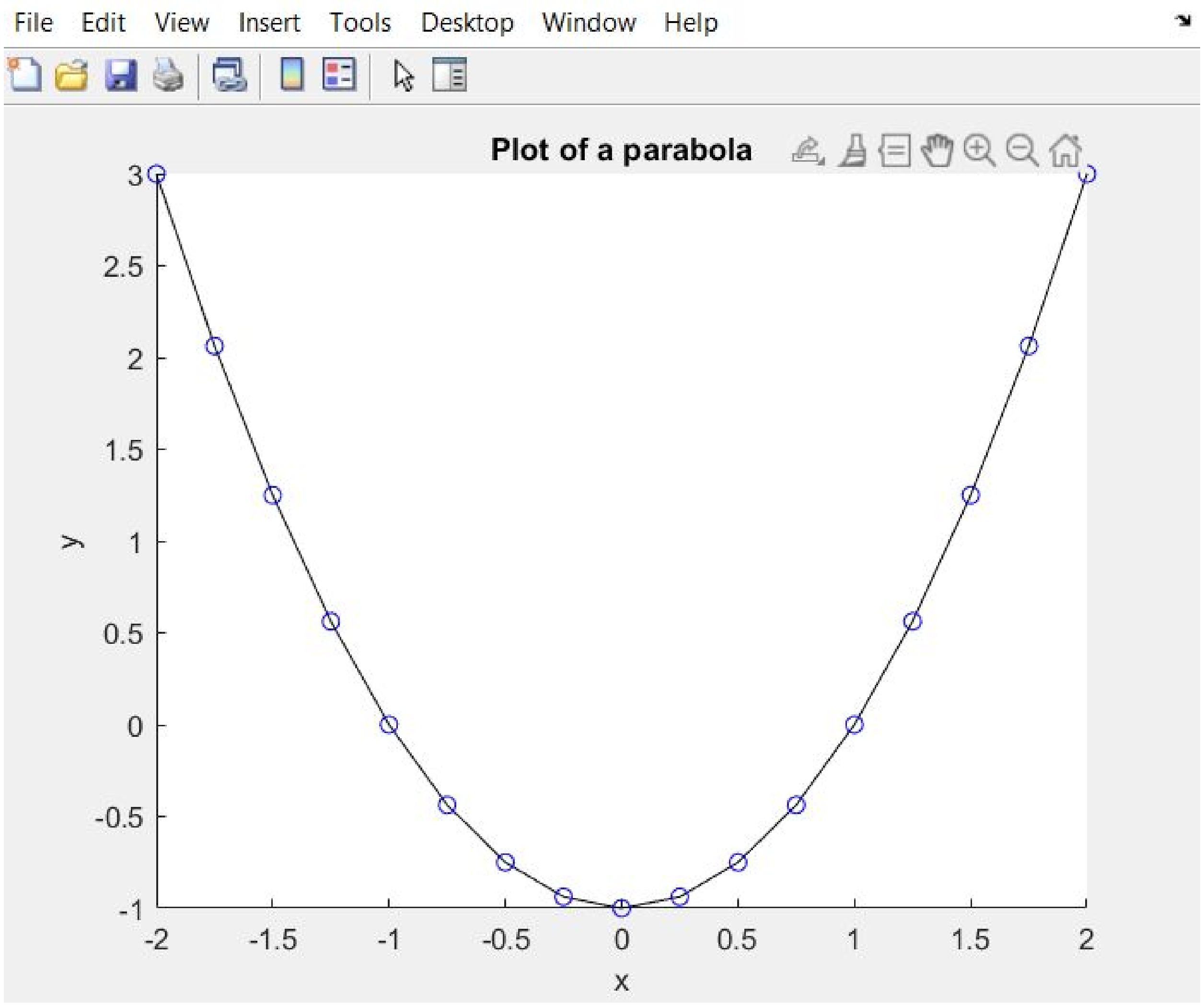
So from the following lines:

```
x = -2:0.25:2;
y = 2*x + 3;
```

x is the domain we want to consider and y is the function we input the x values in to get an output y value.

Question 2:

Figure 2 from question 2's given code:



(a) Explain in your own words what the .^ (in MATLAB) or the ** (in python) operator does.

These functions raise a value to a certain power. For example, 2.^(2) and 2**(2) both raise 2 to the second power.

(b) Explain in your own words what the black curve in the graph represents. Is it exactly the same as the graph of a parabola? If not, what is the difference? (It may help to zoom in on the figure.)

The curve does not seem to be the same as a graph of a parabola upon zooming in. I believe it is an approximation as there is a line segment plotted between each blue circle. By increasing the number of blue circles (and perhaps not plotting them since they'd get in the way) we would get a more accurate parabola. However, there will be line segments and not a perfect graph of a parabola.

Take the line of code:
```
x = -2:0.25:2;
```

My understanding is that if we decrease the value in the middle (i.e. that value that determines the difference between input values) we will get a more precise graph.

Right now the workspace displays x as a 1x17 matrix; which means that there are 17 values we are inputting and 16 line segments that connect sequential points. (the output values from the function y are also stored in a 1x17 matrix)

If we were to change the line to: x = -2:0.10:2, we get a 1x41 matrix for x values (and a 1x41 matrix for y values). So now there are 41 points, 40 lines, and the figure is a better approximation of the parabola y = x^(2) - 1. However this took more computation power and memory I think.