CM2104: Computational Mathematics Assessed Individual Project Work: MATLAB Simple 2D Geometry Lab

William Cooter – C1535277

My program consists of 9 MATLAB files. The file to be run is main.m. This contains the GUI for the program (see figure 1). The interface consists of 4 buttons. The first is the 'New Line' button. When clicked this opens a survey (see figure 2) for the X and Y co-ordinates of the end points of the line to be entered. When 'OK' is pressed, the data is added to the lines matrix. The second button is 'New Circle' functions similarly (see figure 3) except the inputs are the x and y co-ordinates of the centre of the circle and the radius, to be stored in the circles matrix. Below these two buttons is the plotting grid which is initially empty. Beneath the plot are two more buttons. The 'Reset' button restores the plot to its initial state using the same code that is executed in the main_OpeningFcn() function. The final button is 'Plot' which executes the plotAll() function, and draws all inputed circles and lines onto the grid.

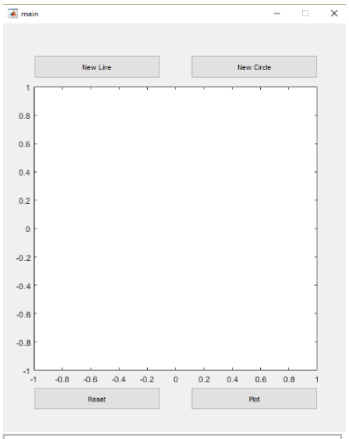

Figure 1

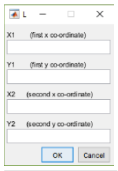

Figure 2

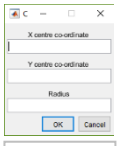

Figure 3

The plotAll() function takes inputs of the lines and circles matrices, and returns a plot of all items entered and their intersection points. This is done by looping through both matrices and using the plot() function to plot each input. After this the intersects and the axis limits are calculated using the intersects() and axeslimits() functions.

The intersects() function (task 1) contains 3 for loops. The first loops through all items is the lines array and calculates the intersects between them using the StraightAndStraightIntercept() function. The second loops through each item in lines and finds the intersect of it with each item in circles using the StraightAndCircleIntercept() function. The third functions like the first except it finds the intersects of all items in circles using the CircleAndCircleIntercept() function.

The StraightAndStraightIntercept() function calculates the intercection points of all entries in the lines array. First the gradients and y-intercepts are calculated for each line. After this is a 5 part if statement. The first condition is if both graidents are infinite (i.e.: both lines are verticle). In this case there are infinite intersection points if the lines overlap, so an empty array is returned. The second and third conditions are if only one of the gradients is infinite. In this case the x co-ordinate of the intersection is simply the x co-ordinates of the end points of the verticle line, and the y co-ordinate is simply $y = mx + c$. This gives the intersection point as if the lines did not have limits, however as they have end points, the lineIn() function is used to determine whether this point is between the limits. If it is, the intersection point is returned, and if not an empty array is returned. The fourth condition comes about when the gradient of both lines is 0. This is similar to the first condition where if the lines do intersect it is at inifinite points, and so an empty array is returned. The final condition is when both gradients are finite and at least one is not 0. In this case the x co-ordinate of the intersect is calculated by the difference in the y intercepts diveded by the difference in the gradients, and the y co-ordinate is again $y = mx + c$. The lineIn() function is used again to determine whether the intersection is between the endpoints, and if it is then the intersection is returned.

The second intersection to be calculated is between lines and circles. This is achieved using the StraightAndCircleIntercept() function. Firstly the gradient and y-intercept of the line are calculated. After this is a 2 part if statement. The first part is for occasions when the gradient of the line is

infinite. When this is the case the intersection points are calculated by $y = b \pm \sqrt{(r^2 - (x-a)^2)}$, as the x value is simply the x co-ordinate of the verticle line. If the intersection points are the same (i.e.: the line is a tangent), one of them is deleted. A while loop is then used to iterate through the intersections and uses the circleIn() function to check whether they are between the end points of the line. The second part on the function is if the gradient of the line is finite, in which case the pre-built linecirc() function is used, which returns all x co-ordinates of the intersects. Again if it is a tangent then 2 results are returned, so one of them is deleted. If there are no intersects, an array of NaN's is given, and so an empty array is returned. The function then goes on to use circleIn() again to determine whether the intersects are between the endpoints.

The final intersections are calculated between circles and circles using the CircleAndCircleIntercept() function. This is done using the pre-built circcirc() function returning the x values of the intersection points. Again if there are none, a NaN array is given, and so an empty array is returned. If the circles do not have the same x co-ordinates (i.e.: they are not on top of eachother – see figure 4), the y co-ordinates of the intersects using $y = b \pm \sqrt{(r^2 - (x-a)^2)}$. However, if the circles are identical in all but their y co-ordinates (figure 4), then all points on the circles with those x co-ordinates are calculated and then compared so that the ones the circles have in comman are the intersection points.

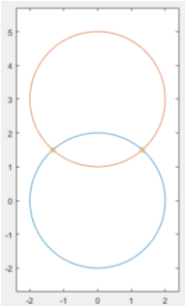

Figure 4