ECOR 2606 Lab #2

To get marks for completing the lab you must get each part checked by a TA (after you complete it) during the lab session. Of course, you may ask questions if you get stuck.

If you are not able to complete the entire lab during your lab section, you may complete the lab during another lab section, for section scheduled for Lab 2. To see the lab schedule, check the course outline. Even if you do not complete the entire lab, **submit "*overlap.m*", "*script.m*", and "*centredist.m*" at the end of the lab, to prove your attendance. Be sure that you successfully submit all the files together.** (Submit empty files if you didn't complete all parts of the lab.) **Use "View" to check that all the files are there.** Many students lose marks on quizzes by submitting two copies of one file.

Part 1

In this part you are going to develop a function m-file (to be called "*overlap.m*") that calculates the overlap area of two circles. This function is given the radii of the two circles (*R1* and *R2*) as well as the distance (*D*) between the centres of the circles (all in mm). In other words, *R1*, *R2*, and *D* are the input arguments. It calculates and returns the overlap area, called *area* (in mm^2). In other words, *area* is the output argument.

Start by creating the function m-file (be sure to select File -> New -> Function (not Script)). Complete the top line of the function (specifying the output argument, function name, and input arguments). Now complete the function documentation (comments – each line must be preceded by "%"), as per the following instructions: On the next line put the function name in all capitals, followed by a one sentence description of the job of this function. On the following line write "Inputs:", and on the three following lines, give the name of each input argument and a one sentence description. On the next line write "Output:", and on the line after that, give the name of the output argument and a one sentence description. Then leave a blank line and finally add a line with your name and student number. **In this course, your name and student number are required for full marks on all code.**

Now click the green "run" button. You will be prompted to save the file. Ensure that you call it overlap.m (which will be the name that Matlab suggests). Next you will get a dialog box asking about the Matlab directory path. Select "add to path". Now go to the Matlab command window and type "help overlap". You will see the comments that you wrote. (Note that the all capitals "OVERLAP" comes out as bold but lower case.) Check that the comments are correct. If not, fix them up, save or run the file again, and type "help overlap" in the command window until they look good. **In this course, a complete and detailed comment block is required for full marks on any functions that you write on tests (written tests or lab quizzes).**

Now we are going to write the code for function *overlap*. We have four cases to consider.

Case 1: The first case is that one or more of the inputs don't make sense. In this case, the inputs are invalid if *R1* or *R2* is negative or zero, or if *D* is negative. We will deal with situations like this by generating an error message. This will cause our message to be output in red in the command window. Unless you are a perfect Matlab programmer, you have likely seen these messages generated by built in Matlab functions! To generate an error message we use the *error* function. The parameter to the error function is the message (enclosed in single quotes) that we wish displayed. After outputting the error message, the *error* function causes our function (*overlap*) to end. Thus we don't need to put a "return" statement after a call to the *error* function.

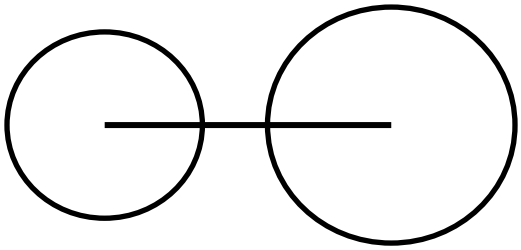
Write an if statement with the error statement inside the if statement, i.e.:

```
% invalid inputs
if ……
   error('……')
end
```

Note that in Matlab, brackets are not required around the "if" condition. The symbols for less than, less than or equal to, is equal to, and, or, etc. are exactly the same as in C / C++ (only not equal to and not differ – use ~ not !). Be sure that your error message indicates what values are valid inputs. Save the file once you have completed your if statement. (Note that you may have one "if" statement for each parameter, or one "if" for all three parameters. One "if" makes the code a little shorter.)

Test your first case by typing in the command window: overlap(-1,1,1). You should see your error message displayed in red.
Now test that nothing happens (as the code isn't there yet) if the inputs are valid, for example: overlap(1,1,1)

Case 2: The inputs are valid, but the circles do not overlap (see diagram below). In other words, *D* is larger than or equal to the sum of *R1* plus *R2*. In this case, the overlap area is 0.



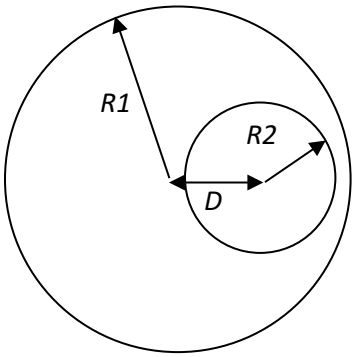
Write the appropriate if statement for this case (after case 1). Note that, unlike C/C++, we don't return the answer; we store it in the output argument (*area*). To avoid the need for an "else" after this if, add a return statement after assigning 0 to area.

```
% the circles do not overlap
if ……..
   area = 0;
   return
end
```

Now we want to test case 2. Try, for example: overlap(1,2,4). Be sure that you get the expected output of 0.

Next make sure that case 1 is still working, e.g.: overlap(-1,1,0). And, finally, check that you don't get an answer for a case that we haven't dealt with yet, e.g.: overlap(1,1,1).

Case 3: The inputs are valid and one circle is completely inside the other.

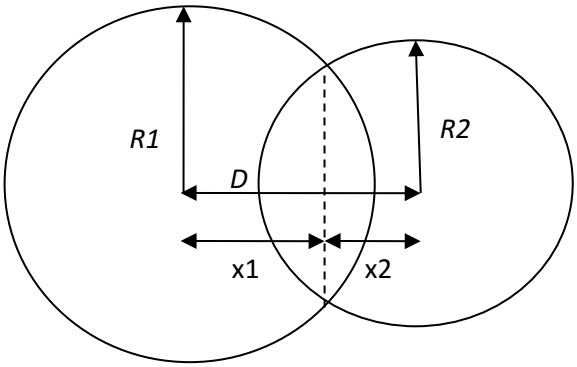


In this case, *D* is less than the absolute value of the difference between *R1* and *R2*. (The absolute value is because it doesn't matter if *R1* or *R2* is larger.) If this is the case, then the answer is just the area of the smaller circle. The area of a circle is πr^2. And the circle we are interested in is the smaller one, thus for *r* we use *min(R1,R2)*.

Write an if statement to take care of this case. Use function *abs* for absolute value, and *pi* for π. Again add a return statement before the end so that we don't need an else statement.

Test case 3, e.g. overlap(4,2,1). Is your answer correct? Re-test cases 1 and 2. And try a test that we haven't dealt with yet, e.g. overlap(1,1,1) to make sure that you don't get an answer.

Case 4: The inputs are valid and the circles partially overlap:



if *D* >= abs (*R1* – *R2*)

As this is the only case left, we don't actually need an "if" statement.

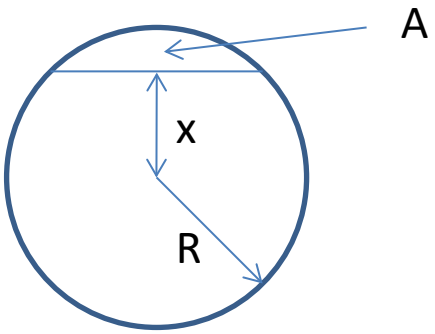
*x*1 and *x*2 can be calculated using:

$$x1 = \frac{D^2 + R1^2 - R2^2}{2D}$$

$$x2 = D - x1$$

Start by adding these two equations to your fourth case. Be careful with your brackets! (Note that one of *x1* or *x2* may be negative. That means that we are calculating the area of a segment that is larger than half the circle – see below.)

The area consists of two circle segments and can be calculated by adding up the areas of the two segments.



The formula for the area of a circle segment is:

$$A = \frac{\pi R^2}{2} - \left[x\sqrt{R^2 - x^2} + R^2 \sin^{-1}\left(\frac{x}{R}\right) \right]$$

This formula is valid for any x between R (A = 0) and -R (A = area of complete circle = πR^2).

As we need to use this formula twice (once for *x1* and once for *x2*), next create an anonymous function of 2 variables (x and R) that calculates A. The Matlab trig functions are at the bottom of the Matlab reference sheet. Hint: you want *asin* here.

Finally your answer for case 4 is the sum of the circle segment for the first circle, i.e. *A(x1,R1)* and the circle segment for the second circle, i.e. *A(x2.R2)*.

Test part 4. Hint: overlap(4,1,3) should give π. overlap(1,1,1) should give 1.2284. Re-test parts 1, 2, and 3. Check that you have adequate comments everywhere.

Note that our function is not capable of handling vectors of distance values. This would be ideal but cannot be achieved just by using "dot" operators. Because of the "if" statements a loop would be required (see function *PRv* in the notes).

Get Part 1 checked by a TA before starting Part 2.

Part 2

Assume that *R1* = 100mm and *R2* = 50mm. Write a script (script.m) that uses function *overlap.m* to:

a) Plot the overlap area for *D* from 50mm to 160mm, using *fplot* (figure 1), and *plot* (figure 2).

Because *overlap* is not a function of one variable it cannot directly be plotted using *fplot*. One way around this little difficulty is the following.

```
R1 = 100
R2 = 50
f = @(D) overlap(R1,R2,D);    % a function of one variable to use for fplot
% or, if you prefer:
f = @(D) overlap(100,50,D);    % a function of one variable to use for fplot
```

To use *plot*, create a vector of distance and overlap values. Because function *overlap* is not vector friendly, this requires a loop. See slide 9 of lecture 4 if you need a reminder of how to do this.

b) Determine and output the value of *D* that gives an overlap area of 2000 mm^2. Hint: This is a root finding problem.

c) Print out a table giving the overlap area for *D* from 50mm to 160mm in steps of 10mm. Note that *fprintf* makes it very easy to produce tables with nicely formatted columns. Output distance with one decimal place and overlap area with five. Don't forget to give your table some headings, e.g. "Distance Overlap Area"

.

Get Part 2 checked by a TA before starting Part 3.

Part 3 (Bonus)

Write a function m-file (*centredist.m*) that, given *R1* and *R2* (both in mm) and an overlap area, *area*, (in mm^2), computes and returns the corresponding centre to centre distance (in mm).

This is a root finding problem. This function will make use of function *overlap*.

centredist should check that the values provided ensure that the two circles do actually overlap (Case 4). In other words, if any of the three input arguments are zero or negative, or if the area is greater than or equal to the area of the smaller circle, then the inputs are invalid and *centredist* will produce an *error*.

As our circles always fall into Case 4 we can calculate the range for *D* that we'll pass to *fzero*: if *D* is too small, we have Case 3; if *D* is too large, we have Case 2. Thus Case 3 gives us the lower bound for *D*, and Case 2 gives us the upper bound for *D*.

Add code to *script* to test your function. Be sure to test valid and invalid inputs, including D at either end of the range. (After testing the invalid inputs, comment out your test case, so that the rest of your script will still run.)

Get Part 3 checked by a TA.