Q.1.

1. I defined multiple for loops to filter out the conditions in which ‘pi’ or ‘e’ is not input, or if the number is less than 0 and if number is more than 200
2. Inside the if loops, now we have only filtered contents.
3. There, the sage kernel is started by using the ‘sage -c’ command.
4. Inside that pi.n(digits = 201) was applied, so that we can get the value of pi upto 201 digits, and there will be no rounding off errors, etc.
5. Similar is done for e. Also, the n() function is used for numerical approximation.
6. After that, I just used the cut command to take out the digit which has been requested for. You need to add 2, as [‘3’, ‘.’] are the first 2 elements of the array of digits of pi.

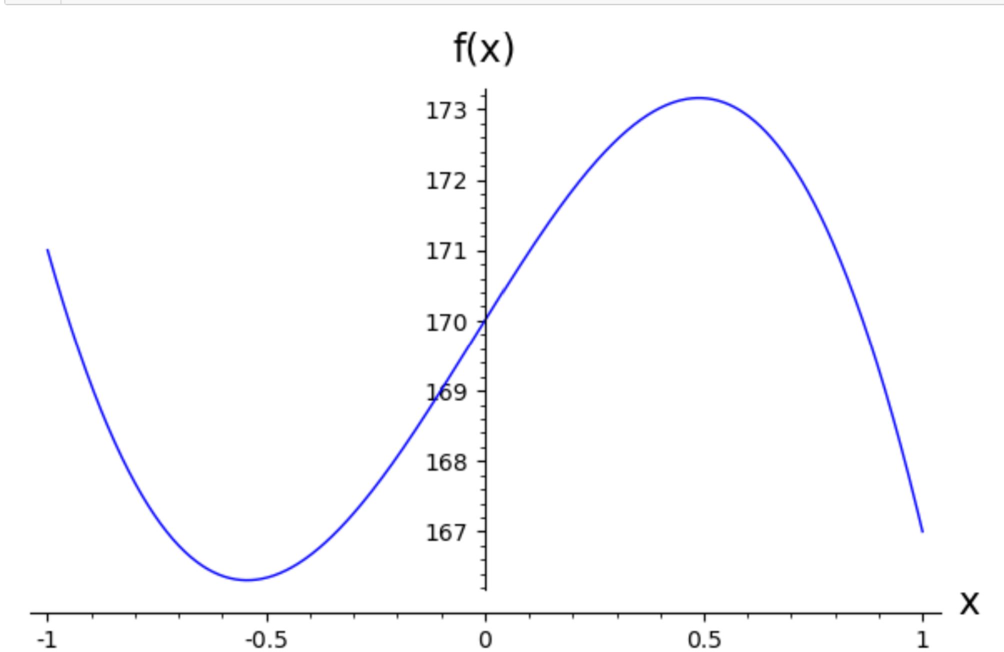
Q.2.

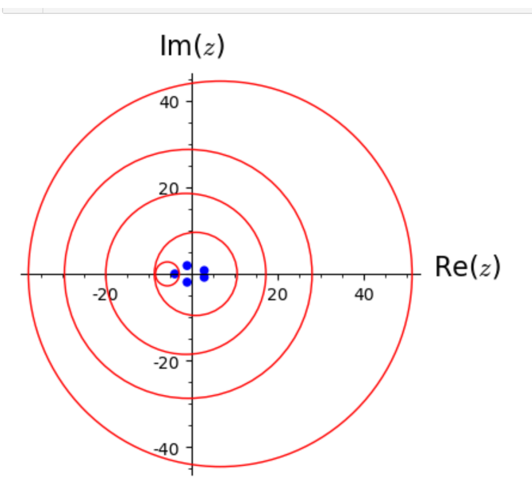
1. numpy, Point, math are imported for running this program.
2. Firstly, xarray[ ] and yarray[ ] are defined, and 20 random elements belonging to the real field in between 0 to 100 are imputed to them.
3. After that, they are converted to np arrays, and then zipped so that (x,y) coordinate based tuples will be created and their coordinates are allotted to the variable p.
4. A function calculate_distance() for 2 points is defined, which takes 4 arguments and returns the distance between them.
5. After that, a 20*20 array, named arr[ ] is defined, and this is the crucial moment.
6. If only the distance is stored, then we won’t have the required points to solve only.
7. So I put 2 tuples, i.e. the point from which we are calculating(constant across a row of the array) and the other tuple which contains the point to which the distance is being calculated.
8. The third element is the distance between the 2 points.
9. Using the lambda function then, the rows are sorted in the order of increasing distances, which is what is represented by the syntax tup: tup[2].
10. I defined 4 lines, which will mark the boundaries of the 100 by 100 plotting ground.
11. After that, using a for loop, we join the point (x1,y1) in every row to the nearest 4 points, in which 1 is 0 distance, i.e. connection with self.
12. It is displayed at the end, along with all the points.
13. Using file.write() functions, etc. the file assn2b.txt is generated if it doesn’t exist and is written, or else if it exists then it is overwritten.

Q.3.

1. The function which was provided is a quintic equation, and is solved using the solve() function.
2. It is converted to a matrix using the companion matrix method, but it has diagonal elements zero. This matrix’s eigenvalues are the same as the roots of the root function.
3. I converted the matrix D, to a matrix E, by multiplying it with A, and postmultiplying it with inverse(A).
4. This ensures that the eigenvalues do not change, but the diagonal elements are non-zero.

5. I did this, so that the Gerschgorin Circle theorem is visually verified, and concentric circles are not plotted, which won't allow us to visualise the theorem's applicability.
6. The function is plotted using the plot function, and the axes are labelled as x and f(x).
7. The points are plotted by defining a function.
8. After that using arrays, and tuples, the centres and radii are collected and plotted respectively, along with the roots of the equation.
9. At the end, a circle with radius as the largest norm of the roots is also drawn along with the roots.





Q.4.

1. Xdata and ydata were defined, and then zipped to solve them.

2. find_fit() is the function which has been used and degree 6 and degree 3 polynomials are defined.
3. $f(x) = -0.03685205975477035x^3 + 0.4561218676417699x^2 - 1.358811442795239x + 2.105109375918615$
4. $h(x) = -0.008573082010582269x^6 + 0.18276289682540203x^5 - 1.3745453042328417x^4 + 4.3664434523810725x^3 - 4.9531911375663045x^2 - 0.21289682539674581x + 2$.
5. The above are the two polyfitted functions.