Assignment-2

ID2090

Ganesh B
CH22B002

1 n^{th} digit of π/e

Write a script that takes two arguments – the first being either "pi" or "e" and the second being an integer between 1 and 200. If the first argument is x and second argument is n then your script is expected to display the nth digit of x after the decimal point. The script should display a message if n is outside the range and exit elegantly. If the number of arguments is less than 2, then too the script should display an appropriate message and exit properly

The script is pretty straightforward to construct. First, we handle the exceptions using if statements: the script accepts only 2 arguments and the second argument must be between 1 and 200(assuming both are inclusive).

Next, we find the n^{th} digit by using Sage's "n(digits=x)" command. Here, the digits have been set to 202 because "digits=x" yields only x-1 digits after decimal for π and e. 201 would have sufficed for this, but 202 is set to avoid any rounding-off error in the last digit.

We then extract the n^{th} by slicing the digits-string appropriately. This entire command has been executed by calling "sage -c".

Script:

```
#!/usr/bin/bash
if [ $# != 2 ]; then echo 'Invalid number of arguments
   !'; exit 0; fi;
if [ $2 -gt 200 ] || [ $2 -lt 1 ]; then echo 'Invalid
   digit argument!'; exit 0; fi;
sage -c "print(str(($1).n(digits=202))[(int($2)+1)])"
```

Test case:

The output is indeed as expected:

```
ch22b002@ID2090:~$ ./assn2a.sh pi 2
4
```

The script has been attached in the tar file. Note that the script must be given permission to be executed.

2 Connecting points

In a square box of size 100x100, place 20 points randomly. Connect each point to the nearest three and thus discretize the square into elements. Display the points and their connections using a plot. Save the locations of the points in a text file (assn2b.txt) in the increasing order of their distance from the origin.

Code:

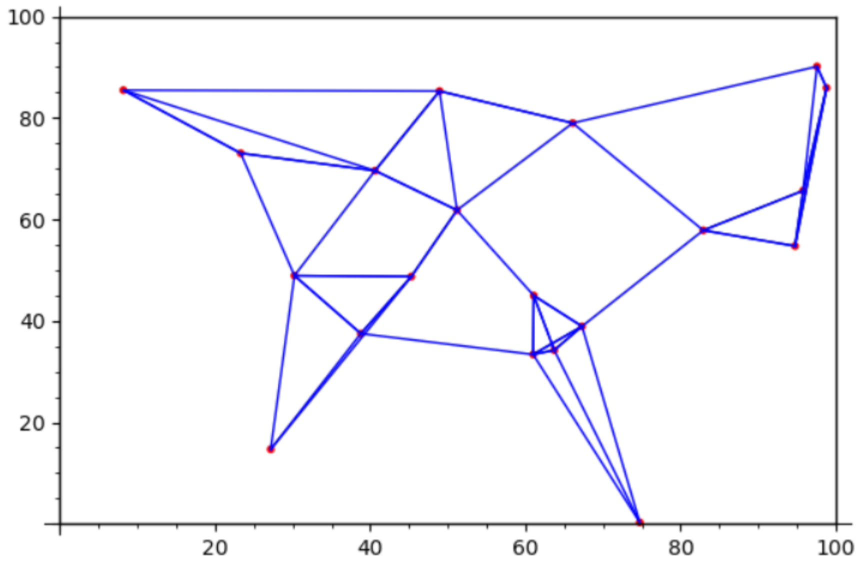
```
from heapq import nsmallest
pointlist=[vector([uniform(0,100), uniform(0,100)]) for _ in range(20)]
infolist=[]
for i in pointlist:
    distlist=[]
    for j in pointlist:
        distlist.append((i-j).norm())
    minlist=nsmallest(4, distlist)
    infolist.append([distlist.index(minlist[1]),distlist.index(minlist[2]),distlist.index(minlist[3])])
plt=Graphics()
for i in range(len(pointlist)):
    for j in range(3):
        plt+=line([pointlist[i], pointlist[infolist[i][j]]], color='blue')
plt+=line([(0,0), (100,0)], color='black')+line([(100,0), (100,100)], color='black')+line([(100,100), (0,100)],
    color='black')+line([(0,100), (0,0)], color='black')
plt+=point(pointlist, color='green')
plt.show()
```

We start by generating a set of 20 random floating points lying inside the 100×100 square using the random-uniform function. The ”vector()” function is used to enable the usage of the ”norm()” function.

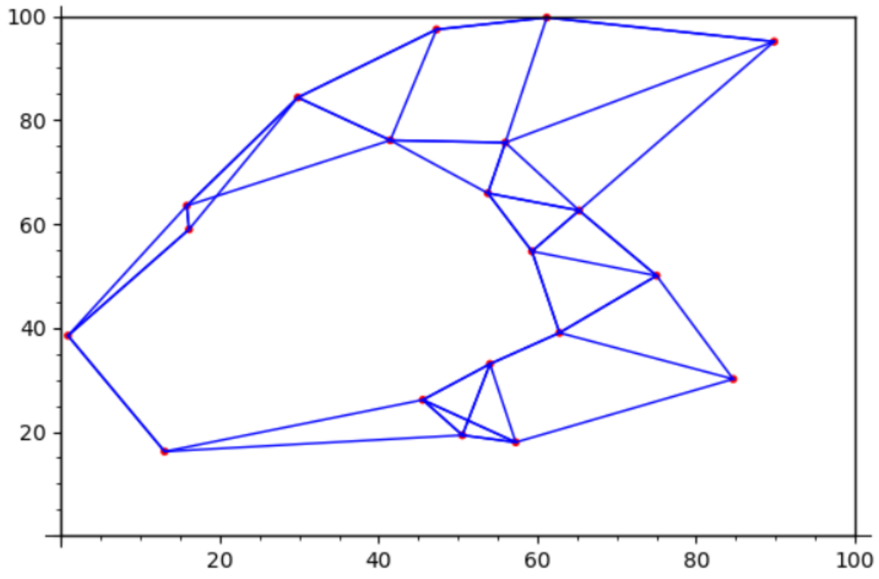
To find out the nearest 3 points for every point in the randomly generated list, we set up a nested for-loop. The distance between pairs of points is found using the ”norm()” function and this information for each point is stored in the list ”distlist”. The nearest 3 points are found by finding the second, third, and fourth smallest elements of ”distlist”(the smallest element is always the point itself). This information, for each point, is then stored in ”infolist” and is iterated to find the 3 nearest points of all points.

Next, we begin to plot by initializing the plot variable ”plt=Graphics()”. The square boundary and the points are plotted. To construct the line segments, we again set up a nested loop and use the ”line()” command. Thus the final plot is obtained as desired: 20 randomly generated points that discretize the square.

Output:



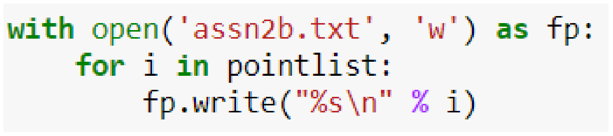
Sample Output 1



Sample Output 2

Saving the location of the points in a text file:

We simply write the elements of the list ”pointlist” to the text file ”assn2b.txt”



```
with open('assn2b.txt', 'w') as fp:
    for i in pointlist:
        fp.write("%s\n" % i)
```

The text file attached contains the locations of points in Sample Output 1.

3 Gershgorin Circles

The Gershgorin circle theorem identifies a region in the complex plane that contains all the eigenvalues of a complex square matrix. For an $n \times n$ matrix $\mathbf{A}$, define R_i to be the sum of the absolute values of the non-diagonal entries in the i-th row:

$$R_i = \sum_{\substack{j=1 \\ i \neq j}}^{n} |a_{ij}| \,.$$

Then each eigenvalue of A is in at least one of the disks

$$\{z : |z - a_{ii}| \leq R_i\} \,.$$

i.e., each eigenvalue lies in the union of these disks.
This theorem seems completely unrelated to locating the roots of a polynomial. However, there is a slick way to link these two: we construct an $n \times n$ matrix C whose eigenvalues are roots of the monic polynomial

$$f(x) = a_o + a_1 x + \cdots + x^n$$

The locations of eigenvalues of A are now the locations of the roots of f! The matrix C which corresponds to such a polynomial f is given by

$$C(f) = \begin{pmatrix} 0 & \cdots & 0 & 0 & -a_0 \\ 1 & 0 & \cdots & 0 & -a_1 \\ \vdots & \ddots & \ddots & \vdots & \vdots \\ 0 & \cdots & 1 & \ddots & -a_{n-2} \\ 0 & \cdots & 0 & 1 & -a_{n-1} \end{pmatrix}$$

or its transpose. C is called the companion matrix or the Frobenius matrix of the polynomial f.

Given the polynomial

$$f(x) = x^5 - 13x^3 - x^2 + 10x + 170$$

we determine its companion matrix to be:

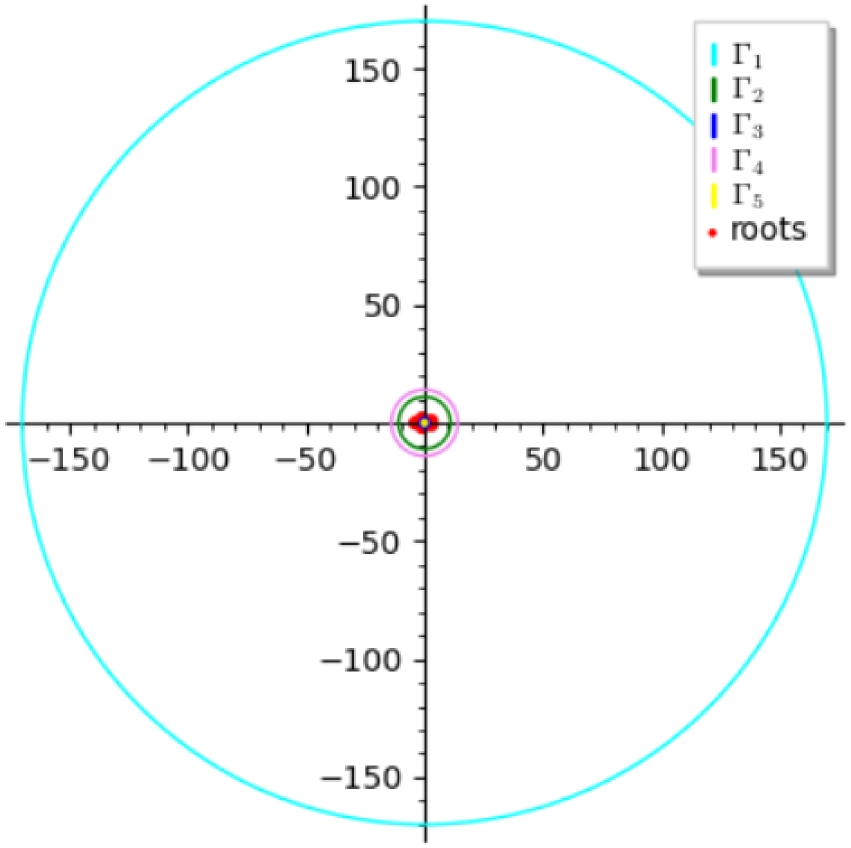
$$C(f) = \begin{pmatrix} 0 & 0 & 0 & 0 & -170 \\ 1 & 0 & 0 & 0 & -10 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 13 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Applying Gershgorin's theorem to this matrix, we get:

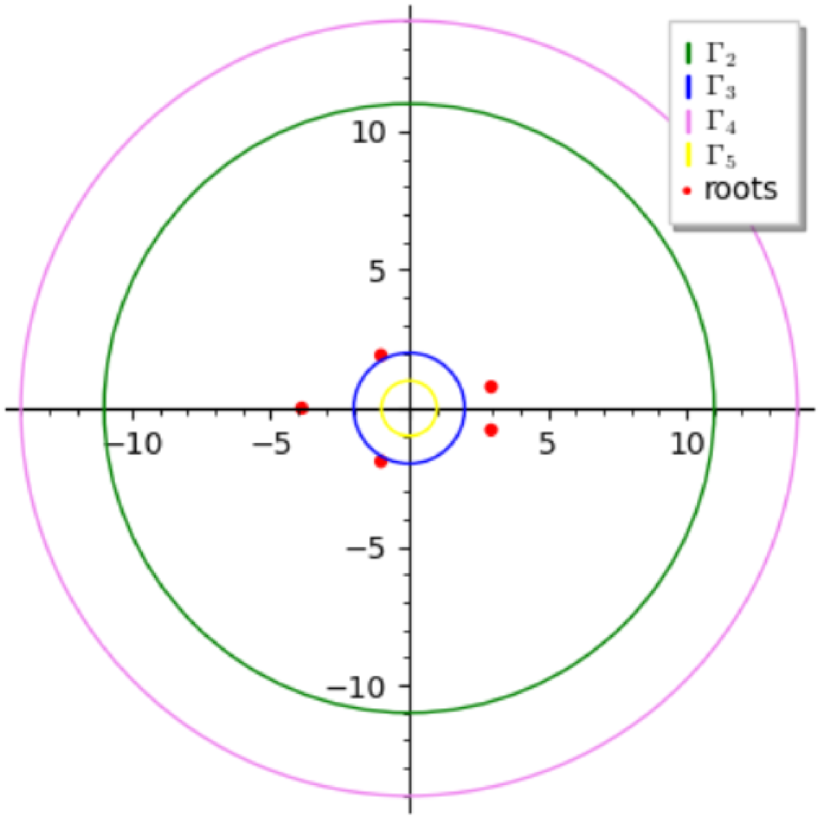
$$\Gamma_1 = \{z \in C : |z| \leq 170\}$$
$$\Gamma_2 = \{z \in C : |z| \leq 1 + 10 = 11\}$$
$$\Gamma_3 = \{z \in C : |z| \leq 1 + 1 = 2\}$$
$$\Gamma_4 = \{z \in C : |z| \leq 1 + 13 = 14\}$$
$$\Gamma_5 = \{z \in C : |z| \leq 1 + 0 = 1\}$$

Therefore all the roots of f strictly lie inside the circle $\Gamma_1 = \{z \in C : |z| \leq 170\}$

Let us plot the roots and the circles on the complex plane:



Omitting Γ_1 for clarity, we see that the roots indeed lie inside the union of these circles:



This theorem is incredibly useful in finding bounds for eigenvalues of a matrix or roots of a polynomial, as seen above. The bounds can further be strengthened by optimizing the theorem.

References:

- Bell, H. E. (1965). Gershgorin's Theorem and the Zeros of Polynomials. The American Mathematical Monthly, 72(3), 292–295. https://doi.org/10.2307/2313703

- Bounds for Polynomial's Roots from Hessenberg Matrices and Gershgorin's Disks. https://doi.org/10.4236/apm.2021.1112062

- https://mathworld.wolfram.com/GershgorinCircleTheorem.html

4 Interpolation

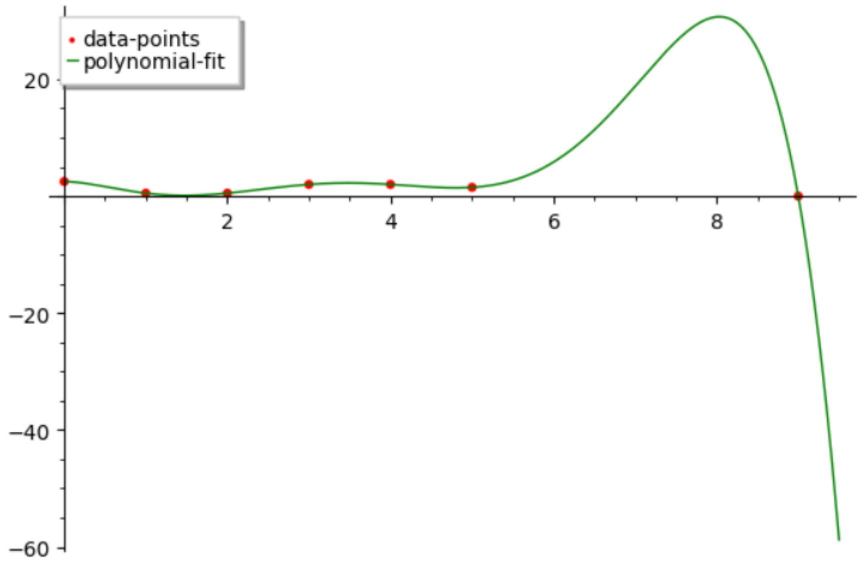
Plot the natural cubic spline function that interpolates the following set of data points. Compare this graph with the one obtained from a polynomial interpolation for the same set of points. Your report should include the functions that were used to make the plots.

x	0	1	2	3	4	5	9
y	2.5	0.5	0.5	2	2	1.5	0

Polynomial interpolation

Given n sample points, it is always possible to construct an(albeit complex) polynomial of degree $n-1$, since it is always possible to find the n coefficients of the $n-1$ degree polynomial, given n equations.

We employ Sage's find_fit() function to achieve this:



The following polynomial function was found to pass through the given points

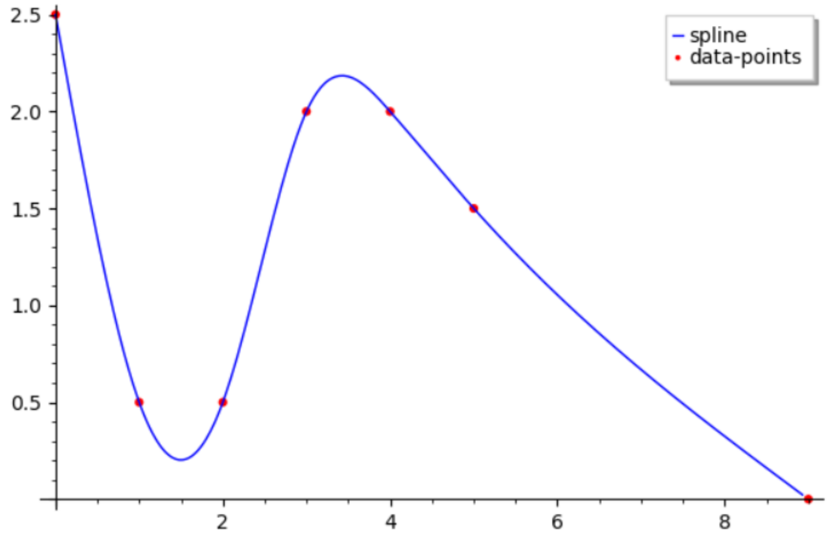
$$f(x) = a + bx + cx^2 + dx^3 + fx^4 + gx^5 + hx^6$$

where

$$a = 2.5,\quad b = -0.21289682539674581,\quad c = -4.9531911375663045,$$
$$d = 4.3664434523810725,\quad f = -1.3745453042328417,$$
$$g = 0.18276289682540203,\quad h = -0.008573082010582269$$

Spline interpolation

Splines are piecewise defined polynomials used in interpolation problems. Spline interpolation is often preferred to polynomial interpolation because it yields similar results, even when using low-degree polynomials.
Given these sample points, we use Sage's in-built "spline()" function. The default order of the spline in Sage is 3.



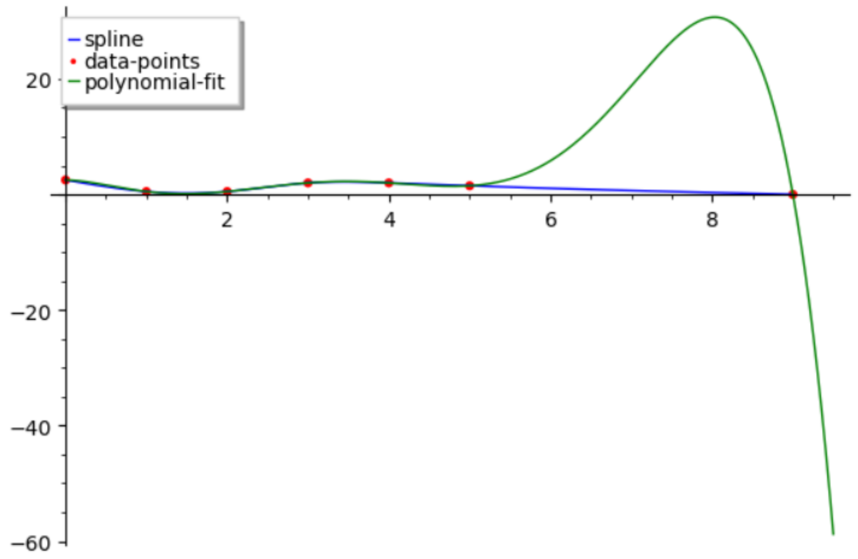
There are various algorithms to compute splines. These are usually solved using computers. The general expression For a C^2 interpolating cubic spline is given by

$$S_i(x) = \frac{z_i\left(x-t_{i-1}\right)^3}{6h_i} + \frac{z_{i-1}\left(t_i-x\right)^3}{6h_i} + \left[\frac{f\left(t_i\right)}{h_i} - \frac{z_i h_i}{6}\right](x-t_{i-1}) + \left[\frac{f\left(t_{i-1}\right)}{h_i} - \frac{z_{i-1} h_i}{6}\right](t_i-x)$$

where

- $z_i = f''\left(t_i\right)$ are the values of the second derivative at the ith knot.
- $h_i = t_i - t_{i-1}$
- $f\left(t_i\right)$ are the values of the function at the ith knot.

Comparing the models:


spline
data-points
polynomial-fit