assn2a

May 28, 2023

0.1 [Q1, 4 marks]

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.

We first take 2 inputs and check whether the are 2 arguments or not. If no then exit. Then we check that is second argument less then equal to 200.

Now we invoke sage. First we define a Data Type with 669 bits and get the π and e value upto 200 digits. Here 669 is chosen(By reducing the value starting from 1000 and then putting wc instead of cut) in order to get only the value upto 200 digit after decimal.

Then you it charater wise to get the required output.

[ ]:
```
#!/usr/bin/bash
#ae22b062
c=`expr $2 + 2`
if [ $# -eq 2 -a $c -lt 203 ]
then
        sage -c "R= RealField(669);a=R($1);print(a)" | cut -b $c
else
        echo "Give Proper input"
        exit
fi
```

assn2b

May 28, 2023

0.1 [Q2. 4 marks]

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

We Import 2 function 1. dict from math to find distace between two points 2. cmp_to_key to get the so that we can pass a compare just like it is possible in C++ and Python2

To make a list of random points we use randint. We make a list 2 randint and treat them as a cordinate and repeat this process 20

[1]:
```
from math import dist
from functools import cmp_to_key
random_points = [[randint(1, 100),randint(1,100)] for i in range(20)];
```

Now we need a way to find closest points to a given point. For this we find the distance of a single point from all the points (including itself). And we make a list and sort the list from distance and add the first four item of the list to other list. As the distance of the point from itself will be 0. The four elements persent in this list are closest point and the point itself.

Now we have the list. We clean the list by removing the the distance present in the list.

[2]:
```
def closest_pts(random_points):
    pts = []
    for p in random_points:
        distance_from_a_single_pt = []
        for q in random_points:

            d = dist(p,q)
            dum=[]
            dum.append(d)
            dum.append(q)
            distance_from_a_single_pt.append(dum)

        distance_from_a_single_pt.sort()
        pts.append(distance_from_a_single_pt[:4])
        distance_from_a_single_pt.clear
    fpts=[]
    for elment in pts:
        dum=[]
        for i in elment:
            dum.append(i[1])
        fpts.append(dum)
    return fpts
```

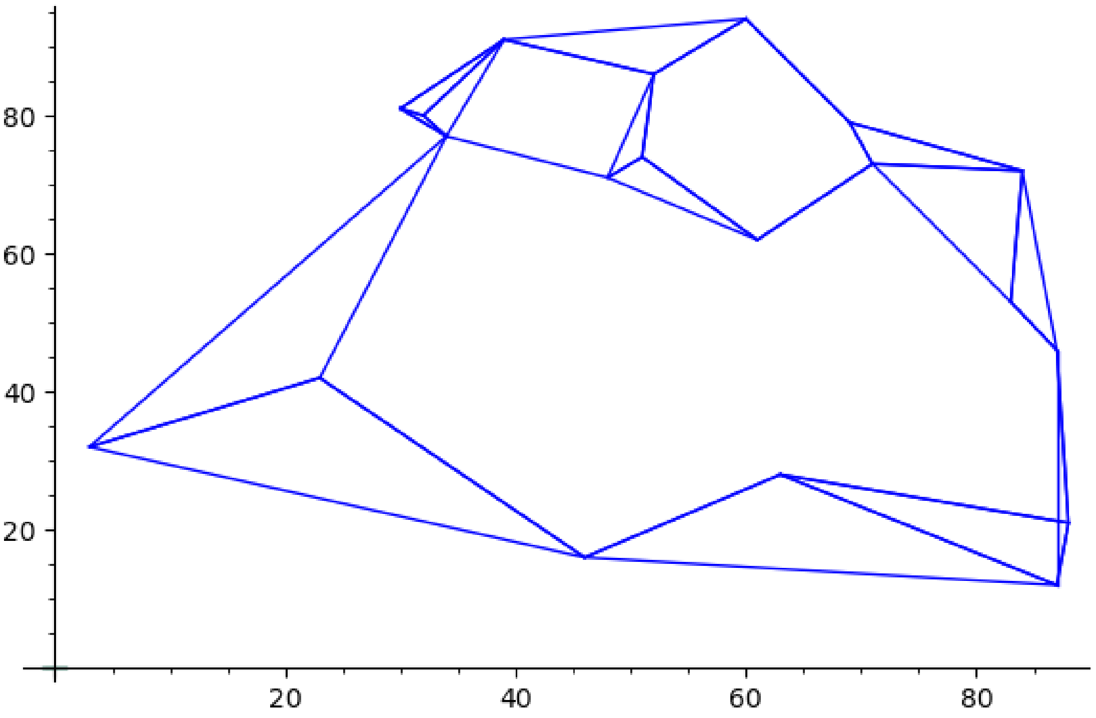
```
[3]: pairs = closest_pts(random_points)
```

We get a empty plot by making the origin

```
[4]: final_plot = plot([0,0])
```

Now we use a for loop to join all the points one graph and then append it over and over

```
[5]: for item in pairs:
         p1= line([item[0],item[1]])
         p2= line([item[0],item[2]])
         p3= line([item[0],item[3]])
         final_plot = final_plot + p1 + p2 + p3
     final_plot.show()
```



We first define fuction to compare the distance from the origin. Now if the distance of the first point is less than that of the second point then we return -1. Any negative return value puts the first input ahead of the second input in the sort funtion when the compare function is called. And vice versa for positive return.

[6]:
```python
def compare(a,b):
    if dist(a,(0,0)) < dist(b,(0,0)) :
        return -1
    else:
        return 1

ascending_point= sorted(random_points,key=cmp_to_key(compare))
```

Now write the list to a text file

[7]:
```python
with open("assn2b.txt", "w") as f:
    for s in ascending_point:
        f.write(str(s) +"\n")
```

[ ]:
```
```

assn2c

May 28, 2023

0.1 [Q3. 4 marks]

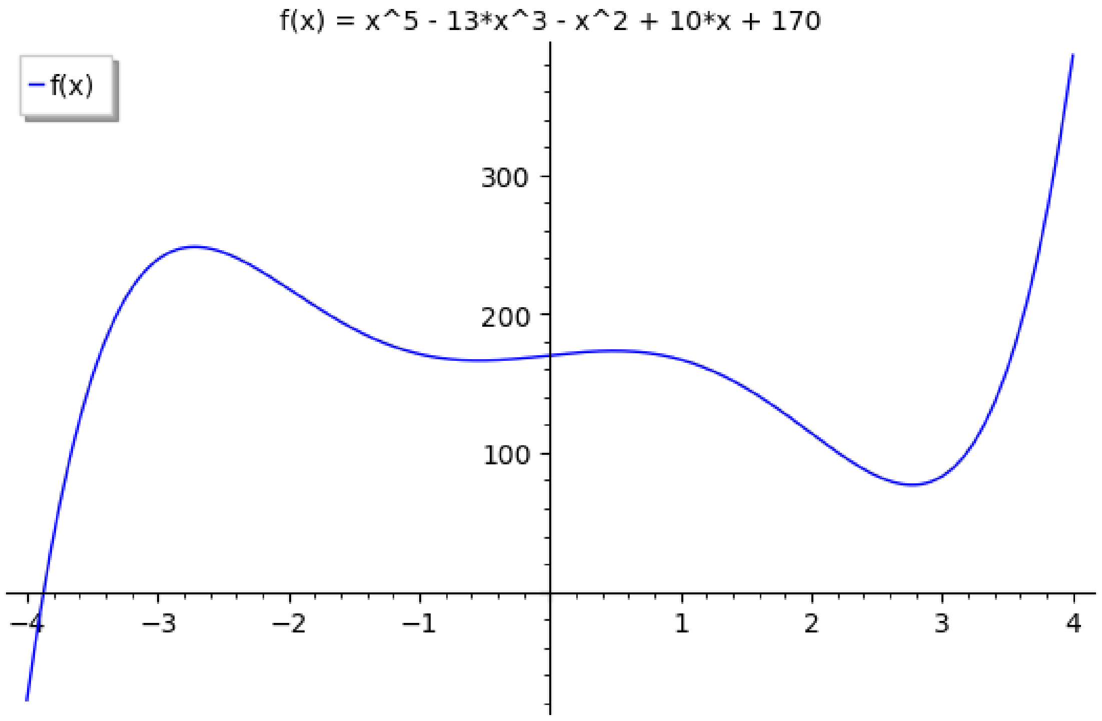
Consider a function a follows :

$$f = x^5 - 13x^3 - x2 + 10x + 170$$

How many real roots does this function have? Estimate all the roots. Visualize the roots using Gerschgorin-Circle. Include a plot of the function itself, complete with labels and title in your report.

[1]:
```
f = x^5 -13*x^3 -x^2 +10*x+170
```

[2]:
```
P=plot(f,(-4,4),legend_label="f(x)",title=f'f(x) = {f}')
P.show()
```



By observation we can tell that there is only one real root and rest of the roots are complex

[3]:
```
A=  f.roots(ring=CC)
A=list(map(lambda x : x[0],A))
A
```

[3]:
```
[-3.87312795888908,
 -1.01844653691890 - 1.91371886668762*I,
 -1.01844653691890 + 1.91371886668762*I,
 2.95501051636344 - 0.779461796772987*I,
 2.95501051636344 + 0.779461796772987*I]
```

Now we find the roots of the function by the in built function roots. There the rings CC tell the function to also fing the complex roots

[4]:
```
a = f.coefficients(sparse = False)
```

Now we find out the coefficients of the polynomial $f(x)$. By using the coefficients function. In the coefficients“sparse = False” tells to take 0 for the missing indices.

Now we get to Gerschgorin-Circle. Gerschgorin-Circles are drawn for a Matrix. Now we are given are Equation $f(x)$. We will treat this Equation as the characteristics equation. And make a Companion Matrix

[5]:
```
A = companion_matrix(a,format='right');A
```

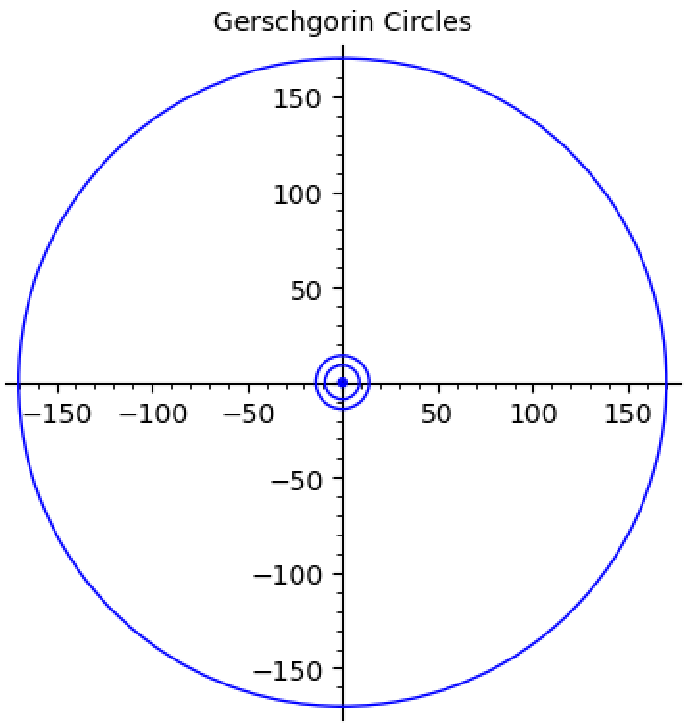
[5]:
```
[   0    0    0    0 -170]
[   1    0    0    0  -10]
[   0    1    0    0    1]
[   0    0    1    0   13]
[   0    0    0    1    0]
```

Once we get the Matrix we know that Gerschgorin Circles are given by the formula $|\lambda - a_{ii}| <= \sum_{i \neq j} a_{ij}$.

There must lie atleast one rooot lying inside this region.

[6]:
```
radii=[]
centres=[]
for i in range(5):
    radii.append(sum(A[i])-A[i][i])
    dum=[]
    dum.append(A[i][i].real())
    dum.append(A[i][i].imag())
    centres.append(dum)
```

[7]:
```
c =  circle(centres[0],radii[0],title="Gerschgorin Circles")
for i in range(1,5):
    c =   c + circle(centres[i],radii[i])
c.show()
```


Gerschgorin Circles
150
100
50
−50
−100
−150
−150
−100
−50
50
100
150

```
[ ]:
```

assn2d

May 28, 2023

1 Question 4

1.0.1 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

First we copy all the elements and store it in the an array

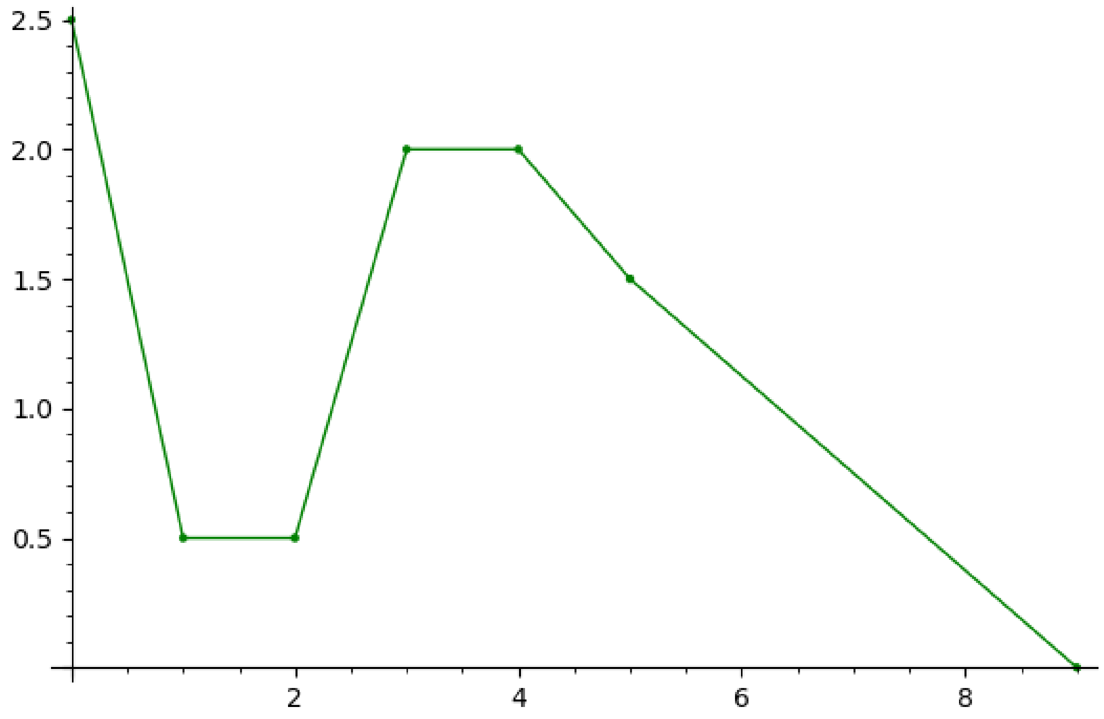
```
[1]: xarray=[0,1,2,3,4,5,9]
     yarray=[2.5,0.5,0.5,2,2,1.5,0]
```

Then we merge these points to create a 2-D array

```
[2]: myData = list(zip(xarray, yarray))
```

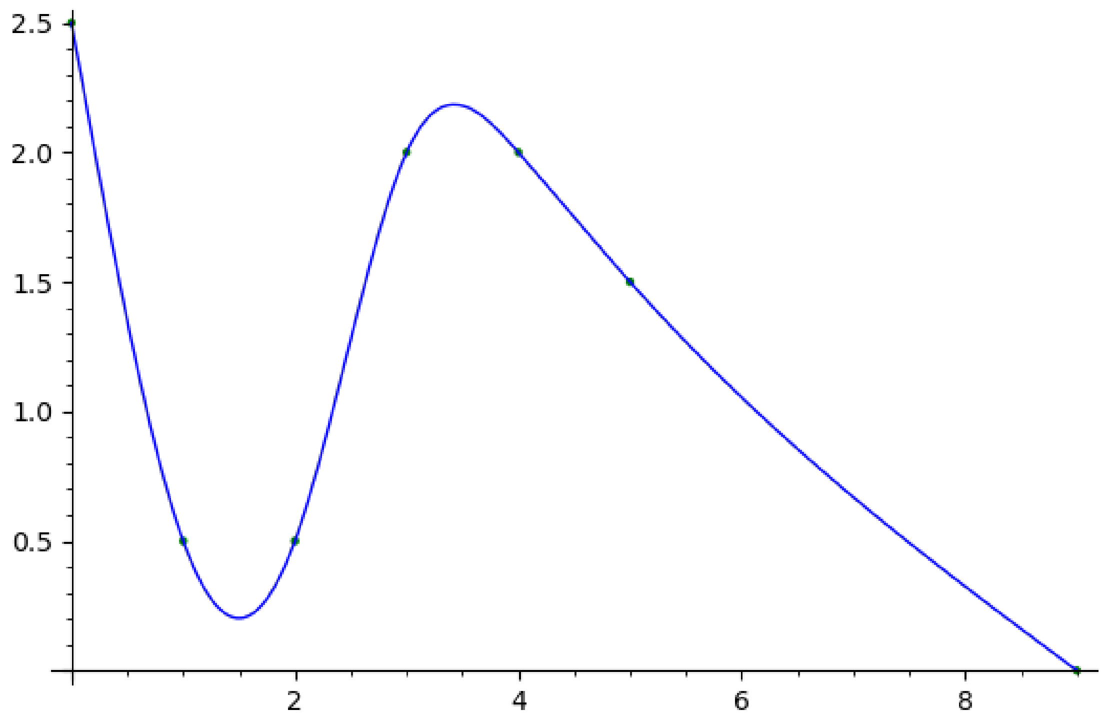
Then we plot the points in the Graph. And show the graph

```
[3]: points=list_plot(myData,color='green')
     points_connected=list_plot(myData,plotjoined=True,color='green')
     (points_connected+points).show()
```



then we use spline command to get the natural spline of the function

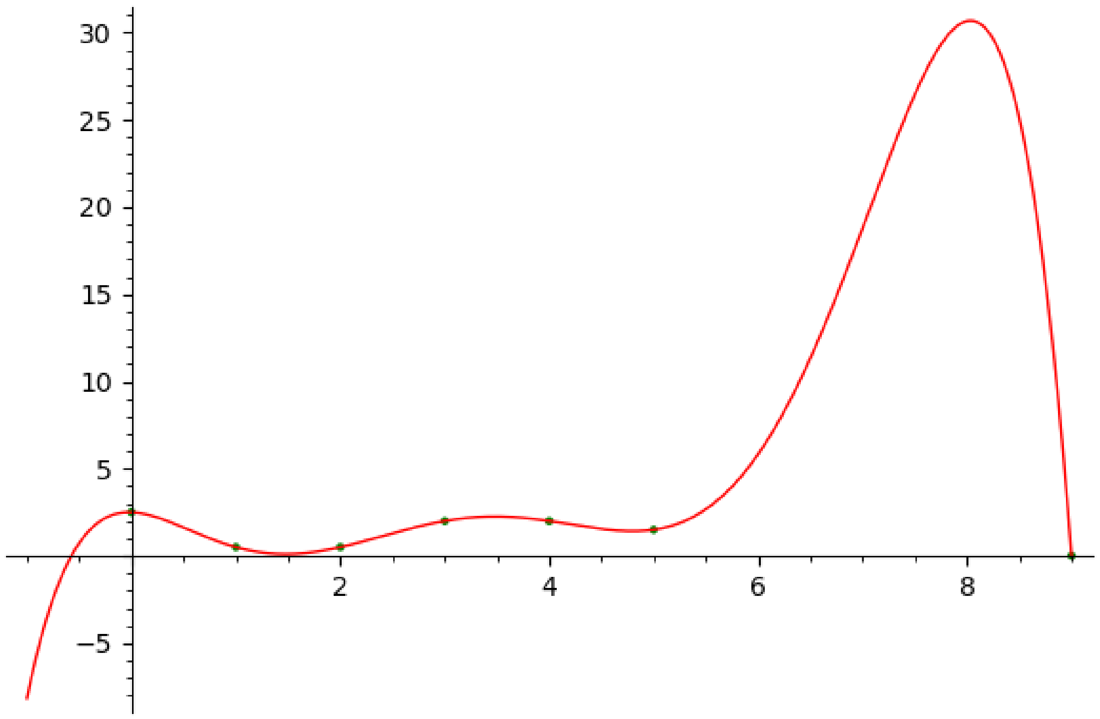
[4]:
```
natural_cubic_spline=plot(spline(myData),[0,9])
(natural_cubic_spline + points).show()
```



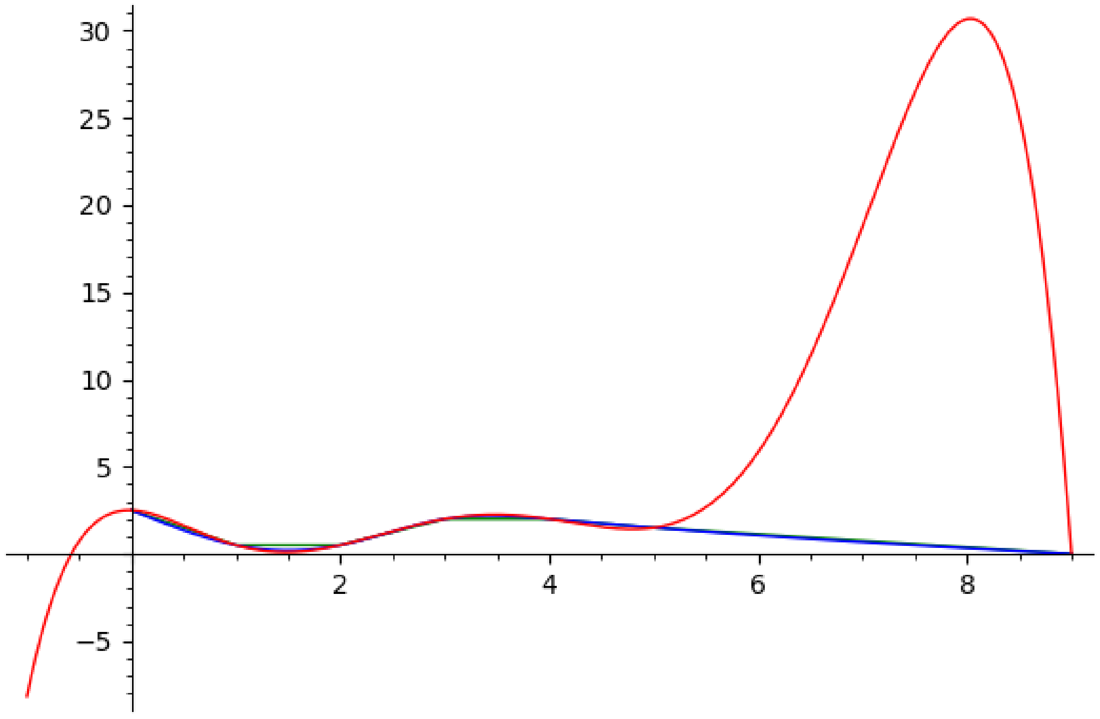
Now as we konw that 7 values are given then the best representation of the figure can be obtained by a 6 degre polynomial of form $ gx^6 + fx^5 + ex^4 + dx^3 + cx^2 + bx + a $

[5]:
```
var('a, b, c, d, e, f, g , x')
model(x) = a + b*x + c*x**2 + d*x**3 + e*x**4 + f*x**5 + g*x**6
sol = find_fit(myData,model)
show(sol)
f(x) = model(a=sol[0].rhs(),b=sol[1].rhs(),c=sol[2].rhs(),d=sol[3].
 ↪rhs(),e=sol[4].rhs(),f=sol[5].rhs()
,g=sol[6].rhs())
polynomial_plot=plot(f(x),x,[-1,9],color='red')
(polynomial_plot + points).show()
```

$[a = 2.5, b = (-0.21289682539674581)\,, c = (-4.9531911375663045)\,, d = 4.3664434523810725, e = (-1.374545304$



[6]:
```
(points_connected+natural_cubic_spline+polynomial_plot).show()
```



All the plots together