16811 A1

September 2022

1.

(a) The implementation is in the q1.py. Run *python q1.py* to get the interpolation result.

(b) By running *python q1.py*, I got the interpolate result for $sin(2\pi x)$ at $x = \frac{1}{10}$ as 0.5877882222892551.

(c) By running *python q1.py*, I got the estimate results:
at n = 2, 1.0882;
at n = 4, 1.048946673207547;
at n = 40, 0.7157134629956571.
The actual value of $f(0.09)$ is 0.7158708568974157.

(d) the interpolation error for n = 2 is 1.9589807426941546
the interpolation error for n = 4 is 5.418917597832672
the interpolation error for n = 6 is 15.731031225174867
the interpolation error for n = 8 is 45.68235961893594
the interpolation error for n = 10 is 131.19950253756573
the interpolation error for n = 12 is 372.06087588229434
the interpolation error for n = 14 is 1043.1298787688495
the interpolation error for n = 16 is 2896.750208459296
the interpolation error for n = 18 is 7981.511500692966
the interpolation error for n = 20 is 21851.558191480788
the interpolation error for n = 40 is 427327258.30265874
The error estimate results show that the error increases as n increases. This demonstrates that larger n does not necessary leads to a better result. This make sense, because the accuracy of polynomial interpolation depends on how close the interpolated point is to the middle of the x values of the set of points used. In the case for example we are interpolating $f(x)$ where $x \in [-1, 1]$, as n increases, the middle becomes farther from the interpolated points, and thus the error gets larger.

2.

$$f^{'}(x) = 2\pi cos(2\pi x); f^{''}(x) = -(2\pi)^2 sin(2\pi x) f^{'''}(x) = -(2\pi)^3 cos(2\pi x)$$

1. linear interpolation

$$\max_{0\le x\le 1}|f^{''}(x)| \le (2\pi)^2$$
$$\max_{0\le x\le 1}(x-x_i)(x-x_{i+1}) = \max_{0\le y\le h}|y(y-h)| = \frac{h^2}{4}$$
$$\max_{0\le x\le 1}|e_1(x)| \le \frac{1}{4}\cdot(2\pi)^2\cdot\frac{h^2}{4} = \frac{(\pi)^2h^2}{4}$$
$$\text{want } \frac{(\pi)^2h^2}{4} \le 5\cdot 10^{-7} \Rightarrow h \le 0.000112539$$
$$\Rightarrow N = \frac{1}{h} = 8885.8$$

Therefore, need 8886 data points (or 8885 intervals).

2. quadratic interpolation

$$\max_{0\le x\le 1}|f^{'''}(x)| \le (2\pi)^3$$
$$\max_{0\le x\le 1}(x-x_{i-1})((x-x_i)(x-x_{i+1}) = \max_{-h\le y\le h}|(y+h)y(y-h)| = \frac{2h^3}{3\sqrt{3}}$$
$$\max_{0\le x\le 1}|e_1(x)| \le \frac{1}{6}\cdot(2\pi)^3\cdot\frac{2h^3}{3\sqrt{3}} = \frac{8\pi^3h^3}{9\sqrt{3}}$$
$$\text{want } \frac{8\pi^3h^3}{9\sqrt{3}} \le 5\cdot 10^{-7} \Rightarrow h \le 0.0031555702$$
$$\Rightarrow N = \frac{1}{h} = 316.9$$

Therefore, need 317 data points (or 316 intervals).

3.

The implementation is in the q3.py. Run *python q3.py* to get the result.
The solutions are: $x_{low} = 14.0661939, x_{high} = 17.2208$.

4.

(a) Define $h(x) = \frac{f(x)}{f'(x)}$.
By Taylor's series, $h(\xi+\varepsilon) = h(\xi) + h^{'}(\xi)\varepsilon + \frac{1}{2}h^{''}(\xi)\varepsilon^2....$

$$
\begin{aligned}
h(\xi) &= \frac{f(\xi)}{f'(\xi)} = 0 \\
h'(\xi) &= 1 - f(\xi)\frac{f''(\xi)}{(f'(\xi))^2} = 1 - \frac{f'(\xi)f''(\xi) + f(\xi)f'''(\xi)}{2f'(\xi)f''(\xi)} \\
&= 1 - \frac{f''(\xi)f''(\xi) + f'(\xi)f'''(\xi) + f(\xi)f''''(\xi) + f'(\xi)f'''(\xi)}{2f''(\xi)f''(\xi) + 2f'(\xi)f'''(\xi)} = \frac{f^{''}(\xi)f^{''}(\xi)}{2f''(\xi)f''(\xi)} = \frac{1}{2}
\end{aligned}
$$

Rewrite $x_{n+1} = x_n - h(x_n)$ as $\xi + \varepsilon_{n+1} = \xi + \varepsilon_n - h(x_n)$.
$\Rightarrow \varepsilon_{n+1} = \varepsilon_n - h(x_n)$.
Then when $n \to \infty, x \to \xi$, we can get

$$
\begin{aligned}
\varepsilon_{n+1} &= \varepsilon_n - h(\xi) \\
&= \varepsilon_n - (h(\xi) + h'(\xi)\varepsilon_n \\
&= \varepsilon_n - (0 + \frac{1}{2}\varepsilon_n) \\
&= \varepsilon_n - (0 + \frac{1}{2}\varepsilon_n) \\
&= \frac{1}{2}\varepsilon_n
\end{aligned}
$$

Then we can get $\lim_{n\to\infty} \frac{\varepsilon_{n+1}}{\varepsilon_n} = \frac{1}{2}$
Thus, in this case ξ is a root of order 2 of $f(x)$, the Newton's method converges linearly, and no longer converges quadratically.

(b) From (a), we get $h(\xi) = 0, h'(\xi) = \frac{1}{2}$.

$$
\begin{aligned}
h''(\xi) &= -\frac{(f'(\xi))^2f''(\xi) + f(\xi)f'(\xi)f'''(\xi) - 2f(\xi)(f''(\xi))^2}{(f'(\xi))^3} \\
&= -\frac{(2f'(\xi))^2f'''(\xi) + f(\xi)f'(\xi)f'''(\xi) - 3f(\xi)f''(\xi)f'''(\xi)}{3(f'(\xi))^2f''(\xi)} \\
&= -\frac{(3f'(\xi))^2f''''(\xi) + f'(\xi)f''(\xi)f'''(\xi) - 3f(\xi)(f'''(\xi))^2 - 2f'(\xi)f''(\xi)f''''(\xi) + f(\xi)f'(\xi)f'''''(\xi)}{3(f'(x))^2f'''(\xi) + f'(\xi)(f''(\xi))^2} \\
&= -\frac{1}{6}\frac{f'''(\xi)}{f''(\xi)}
\end{aligned}
$$

Similarly, if modify Netwon as $x_{n+1} = x_n - 2h(x_n)$, then rewrite as $\varepsilon_{n+1} = \varepsilon_n - 2h(x_n)$.

Then when $n \to \infty, x \to \xi, \varepsilon_n \to 0$, we can get

$$\begin{aligned}
\varepsilon_{n+1} &= \varepsilon_n - 2h(\xi) \\
&= \varepsilon_n - 2(h(\xi) + h'(\xi)\varepsilon_n + \frac{1}{2}h''(\xi)(\varepsilon_n)^2) \\
&= \varepsilon_n - (0 + \frac{1}{2}\varepsilon_n + \frac{1}{2}(-\frac{1}{6}\frac{f'''(\xi)}{f''(\xi)})(\varepsilon_n)^2) \\
&= -\frac{1}{6}\frac{f'''(\xi)}{f''(\xi)}(\varepsilon_n)^2
\end{aligned}$$

Then we can get $\lim_{n\to\infty} \frac{\varepsilon_{n+1}}{(\varepsilon_n)^2} = -\frac{1}{6}\frac{f'''(\xi)}{f''(\xi)}$
Since $f''(\xi)) \neq 0$, in this case ξ is a root of order 2 of $f(x)$, the Newton's method does converge quadratically.

5.

The implementation is in the q5.py. Run *python q5.py* to get the approximation result.
The approximation result for $x^5+x^4+x^3+x^2+x+1$ is -1.0, $-0.5+0.866i$, $-0.5-0.866i$, $0.5+0.866i$, $0.5-0.866i$.

6.

(a) Define Q = $\begin{pmatrix} 1 & -3 & 1 & -3 & 0 \\ 0 & 1 & -3 & 1 & -3 \\ 1 & -1 & -6 & 0 & 0 \\ 0 & 1 & -1 & -6 & 0 \\ 0 & 0 & 1 & -1 & -6 \end{pmatrix}$

x = $\begin{pmatrix} x^4 \\ x^3 \\ x^2 \\ x \\ 1 \end{pmatrix}$

Using the method of resultants, and $Q\mathbf{x} = 0$ iff det(Q) $= 0$. Since $det(Q) = 0$, then $p(x)$ and $q(x)$ share common root.

(b) Using the ratio method, let Q be the an $(n-1)*n$ matrix of rank $n-1$, then a solution to Qx $= 0$ satisfies:

$$x = \frac{x^4}{x^3} = (-1)^{1+2}\frac{det(Q_i)}{det(Q_j)} = (-1)\frac{405}{-135} = 3$$

where Q_i is the (n-1)*(n-1) matrix obtained by deleting the i_th column.

7.

Treat x as constant, and rewrite:

$$p(y) = 1 \cdot y^2 + 0 \cdot y + (x^2 - 1)$$
$$q(y) = 0 \cdot y^2 + 1 \cdot y + x^2$$

Then using the method of resultants, define Q =
$\begin{pmatrix} 1 & 0 & x^2-1 & 0 \\ 0 & 1 & 0 & x^2-1 \\ 1 & x^2 & 0 & 0 \\ 0 & 1 & x^2 & 0 \end{pmatrix}$
let $det(Q) = (x^4 + x^2 - 1)(x^2 - 1) = 0$, we find the projection points $x = -1, 1, \pm\sqrt{\frac{-1+\sqrt{5}}{2}}, \pm\sqrt{\frac{-1-\sqrt{5}}{2}}$.
By plugging into $p(y) = q(y)$, we see roots $y = \frac{1}{2}(1+\sqrt{5}), \frac{1}{2}(1-\sqrt{5})$.

8.

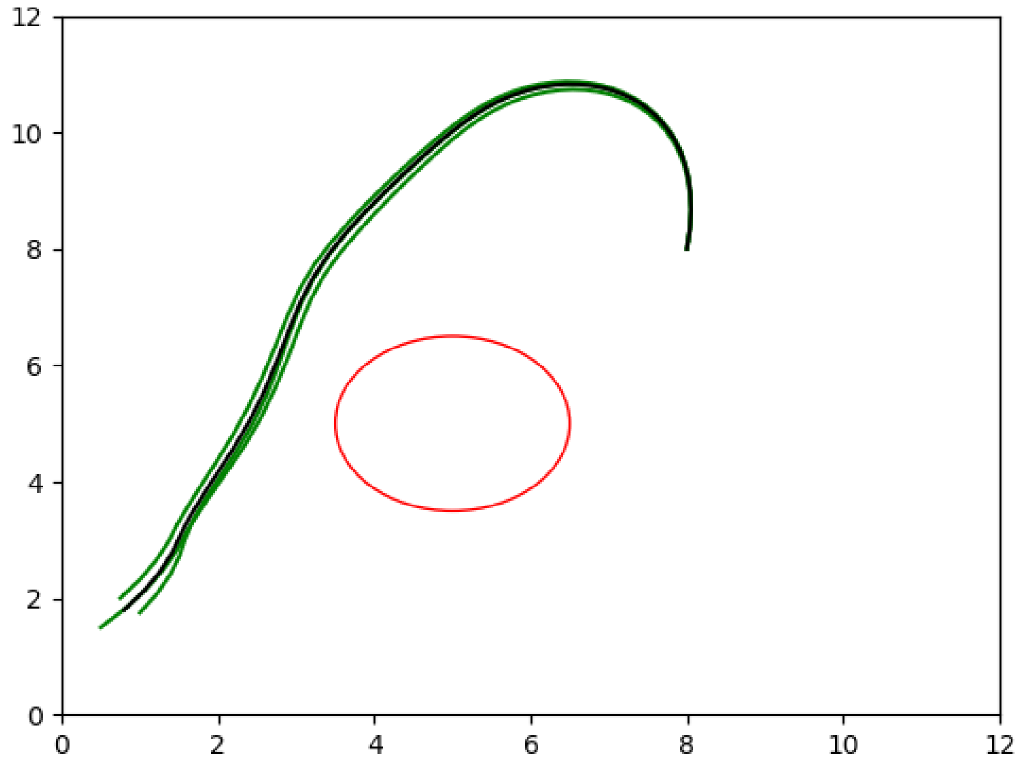
(a) The implementation is in the *get_w* function at q8.py.
Explanation:
To determine whether a 2D point (x, y) falls within the triangle formed by three 2D points:
Define A =
$$\begin{pmatrix} x^{(i)} & x^{(j)} & x^{(k)} \\ y^{(i)} & y^{(j)} & y^{(k)} \\ 1 & 1 & 1 \end{pmatrix}$$
b =
$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$
Then calculate v such that $Av = b$. If the components of vector v are all positive, then point (x, y) falls within the triangle formed by $(x^{(i)}, y^{(i)}), (x^{(j)}, y^{(j)}), (x^{(k)}, y^{(k)})$.

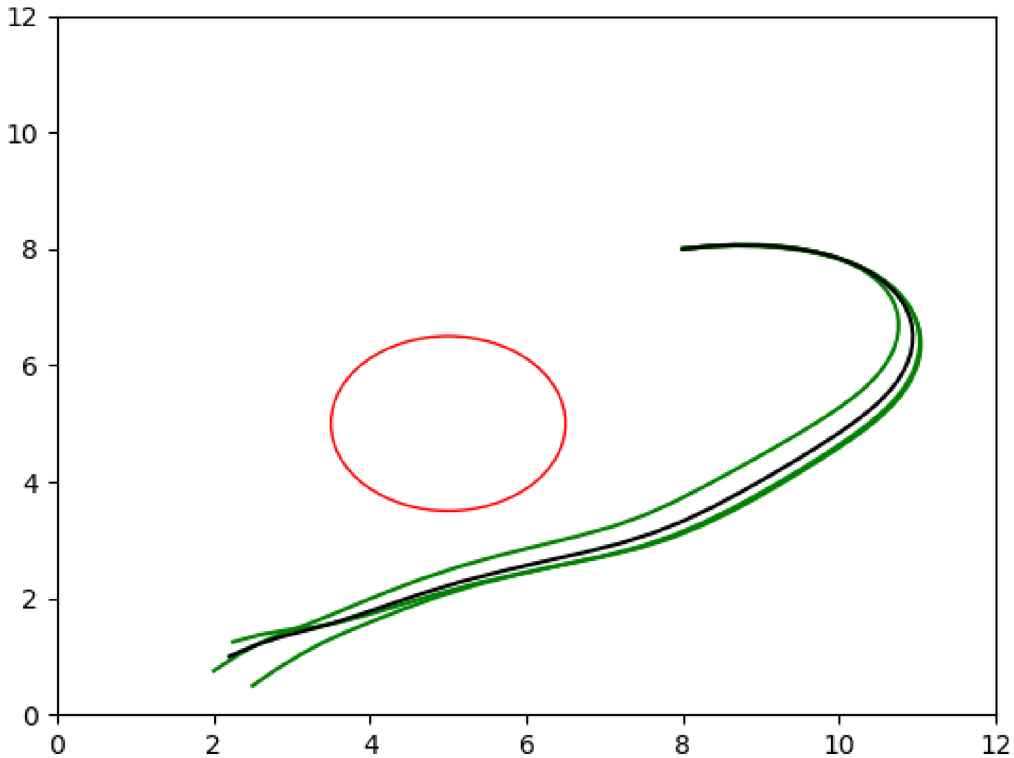
(b) The implementation is in q8.py. Run *python q8.py* to get the result and plots.

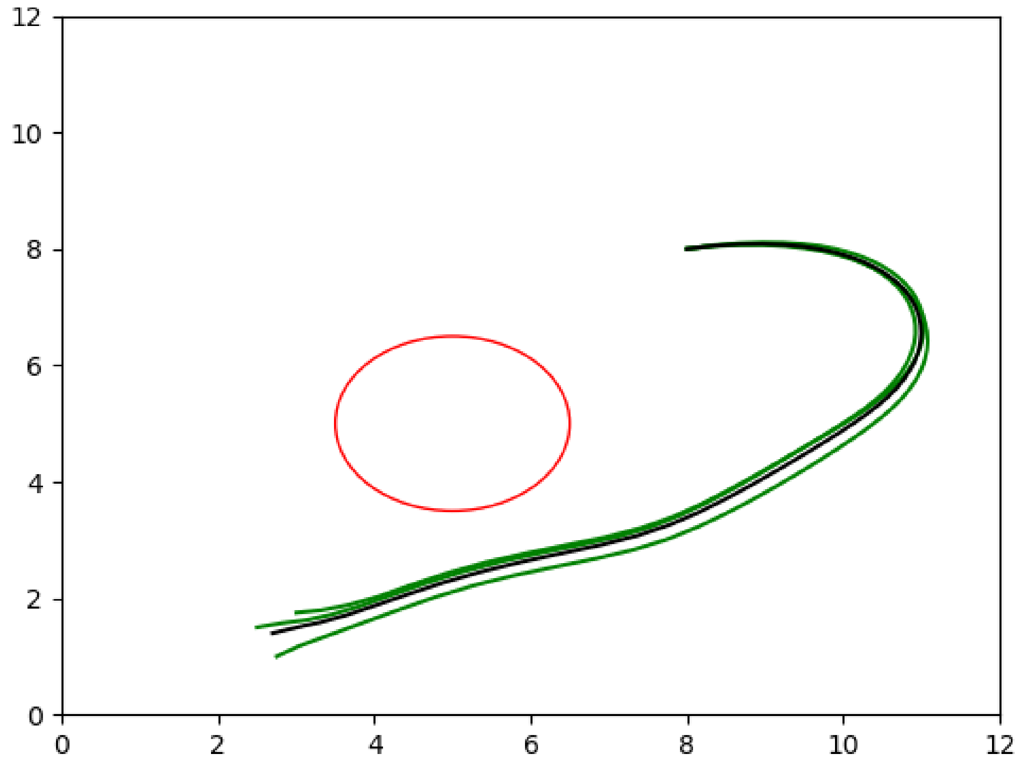
(c) I first load the data from paths.txt. Then picked the 3 optimal paths: 1. pick 3 paths from left or right side, and the picking criteria is that the starting points of the 3 paths should be closet to the starting point; the starting point falls within the triangle formed by the three starting points of the path selected.
2. Decide left or right 3 paths, which are the better choice. Calculate the weights using the method in (a): Av = b, and v contains the three weights we want to solve.
3. The time scale for t is [0, 1, .. 49], the same as the interpolated paths.
4. I used linear interpolation and found it already gave a good result, therefore, there's no need to use a more complex method for simplicity

(d) the green lines are the selected 3 paths, and back line the interpolated result. Figures are in the last pages of this pdf.

(e) If there are more obstacles or different shapes of obstacles, then I need to modify my code that I can no longer just devide to left or right cases. There would be more complicated cases to consider, e.g. the paths to be selected will have more complex shapes.


12
10
8
6
4
2
0
0
2
4
6
8
10
12


0
2
4
6
8
10
12
0
2
4
6
8
10
12


12
10
8
6
4
2
0
0
2
4
6
8
10
12