Assignment 2

Math Fundamentals for Robotics 16-811, Fall 2018
DUE: Thursday, September 27, 2018

[redacted] (Andrew ID: [redacted])

1. Implement a procedure that interpolates $f(x)$ based on a divided difference approach. The procedure should take as input the following parameters:

$$x, x_0, ..., x_n, f(x_0), ..., f(x_n).$$

The procedure should compute an interpolated value for $f(x)$ based on the given data points. Note: The procedure should use all the data points $(x_i, f(x_i)), i = 0, ..., n$, effectively implementing an interpolating polynomial of degree n (or less, depending on the data).

SOLUTION: The code is implemented in `code/q1.m`.

In this script, function `interpolate` is implemented to calculate the coefficients of the interpolation polynomial of degree n (or less) based on a divided difference approach. The inputs of this function include `x`, `data_x` (namely, x_0, x_1, ..., x_n) and `data_y` (namely, $f(x_0)$, $f(x_1)$, ..., $f(x_n)$). It returns `coefficient` as the coefficients of the interpolation polynomial. Then function `estimate_y` is implemented to interpolate $f(x)$ at the given x and get the interpolation result. It returns `y` as the interpolation result at the given x.

(a) Use your procedure to interpolate $(\log_6 x)^{3/2}$ at $x = 2.25$, based on known values of $(\log_6 x)^{3/2}$ at $x = 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0$.

For this part, in script `code/q1.m`, function `get_fx_1` is implemented to calculate $f(x_0), ..., f(x_n)$ given $x_0, ..., x_n$ where $f(x) = (\log_6 x)^{3/2}$. The input of this function is `data_x` and the returned result is `data_y`. Then we call the function `estimate_y` and finally get the result of interpolating $(\log_6 x)^{3/2}$ at $x = 2.25$. The result is **0.3046**.

(b) Now consider the function

$$f(x) = \frac{6}{1 + 25x^2}$$

with input data given at the points

$$x_i = i\frac{2}{n} - 1, \quad i = 0, ..., n$$

For this part, in script `code/q1.m`, function `get_x` is implement to get interpolating data `data_x` (namely, x_0, ..., x_n) given the input parameter n. Function `get_fx_2` is implemented to calculate $f(x_0), ..., f(x_n)$ given $x_0, ..., x_n$ where

$f(x) = 6/(1 + 25x^2)$. The input of `get_fx_2` is `data_x`, which is obtained by calling `get_x`, and the returned result is `data_y`. Then call function `estimate_y` and finally get the result of interpolating $f(x) = 6/(1 + 25x^2)$ at the given `x`.

• Use your procedure to estimate $f(x)$ at $x = 0.05$, with $n = 2$.

The returned result is **5.9856**.

• Use your procedure to estimate $f(x)$ at $x = 0.05$, with $n = 4$.

The returned result is **5.9360**.

• Use your procedure to estimate $f(x)$ at $x = 0.05$, with $n = 40$.

The returned result is **5.6471**.

• What is the actual value of $f(0.05)$?

The actual value of $f(0.05)$ is **5.6471** (rounded).

(c) In this part, you are to (numerically) estimate the maximum interpolation error

$$E_n = \max_{-1 \le x \le 1} |f(x) - p_n(x)|$$

(You don't need to do anything fancy; simply discretize the interval $[-1, 1]$ finely and compute errors at the resulting discrete points.)
Estimate E_n for $n = 2, 4, 6, 8, 10, 12, 14, 16, 18, 20$, and 40, for the function $f(x) = 6/(1 + 25x^2)$ given above.
Do the error estimates make sense? Explain your results.

n	E_n	n	E_n
2	3.8774	14	43.1527
4	2.6301	16	86.3176
6	3.7015	18	175.1139
8	6.2710	20	358.6100
10	11.4939	40	626231.6170
12	21.9780		

The error estimation makes sense because this function $f(x) = 6/(1 + 25x^2)$ is not a polynomial function but a rational function. So polynomial interpolation does not work well for this function. Since we know that for every $x \in [-1, 1]$, there exists $\xi \in (-1, 1)$ such that:

$$e_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!} \prod_{i=0}^{n} (x - x_i)$$

So with a higher value of n, $f^{(n+1)}(\xi)$ would tends to be larger and even tends to be infinity. So the interpolation error would be large and the error estimation results make sense.

2. Suppose you wish to build an interpolation table with entries of the form $(x, f(x))$ for the function $f(x) = \sin x$ over the interval $[0, 2\pi]$. Please use uniform spacing between points. How fine must the table spacing be in order to ensure 6 decimal digit accuracy, assuming that you will use linear interpolation between adjacent points in the table? How fine must it be if you will use quadratic interpolation? In each case, how many entries do you need in the table?

SOLUTION: Suppose the interval $[0, 2\pi]$ is evenly splitted into n intervals so that the table contains values $f(x_i), i = 0, 1, ..., n$ with $h = \frac{2\pi}{n}$ and $x_i = i * h$

(a) If use linear interpolation:

If $\bar{x} \in [x_i, x_{i+1}]$ then we approximate $f(\bar{x})$ with $p_1(\bar{x})$ where $p_1(\bar{x})$ is the polynomial of degree 1 that interpolates $f(x)$ at x_i and $x_{i+1}]$. So the error is:

$$e_1(\bar{x}) = \frac{f''(\xi)}{2!}(\bar{x} - x_i)(\bar{x} - x_{i+1})$$

where ξ depends on $\bar{x}$. Now we have that:

$$|f''(\xi)| \leq \max_{x\in[0,2\pi]} |f''(x)| = \max_{x\in[0,2\pi]} |-\sin x| = 1$$

$$\begin{aligned} |(\bar{x} - x_i)(\bar{x} - x_{i+1})| &= |y(y-h)| \\ &\leq \max_{0\leq y\leq h} |y(y-h)| = \frac{1}{4}h^2 \end{aligned}$$

So, we estimate the error as:

$$e_1(\bar{x}) \leq \frac{1}{2} \times \frac{1}{4}h^2 = \frac{1}{8}h^2$$

For 6 decimal digit accuracy, the value of h should be chosen such that:

$$\frac{1}{8}h^2 \leq 5 \times 10^{-7}$$

then $h = 0.002, n = 3142$, so **3143** entries are needed in the table.

(b) If use quadratic interpolation:

If $\bar{x} \in [x_{i-1}, x_{i+1}]$ then we approximate $f(\bar{x})$ with $p_2(\bar{x})$ where $p_2(\bar{x})$ is the polynomial with degree 2 that interpolates $f(x)$ at x_{i-1}, x_i and x_{i+1}. So the error is:

$$e_2(\bar{x}) = \frac{f'''(\xi)}{3!}(\bar{x} - x_{i-1})(\bar{x} - x_i)(\bar{x} - x_{i+1})$$

where ξ depends on $\bar{x}$. Now we have:

$$|f'''(\xi)| \leq \max_{x\in[0,2\pi]} |f'''(x)| = \max_{x\in[0,2\pi]} |-\cos x| = 1$$

$$|(\bar{x} - x_{i-1})(\bar{x} - x_i)(\bar{x} - x_{i+1})| = |(y+h)y(y-h)|$$
$$\leq \max_{-h \leq y \leq h} |(y+h)y(y-h)| = \frac{2}{3\sqrt{3}}h^3$$

So, we estimate the error as:

$$e_2(\bar{x}) \leq \frac{1}{6} \times \frac{2}{3\sqrt{3}}h^3 = \frac{1}{9\sqrt{3}}h^3$$

For 6 decimal digit accuracy, the value of h should be chosen such that:

$$\frac{1}{9\sqrt{3}}h^3 \leq 5 \times 10^{-7}$$

then $h \approx 0.01983, n = 317$, so **318** entries are needed in the table.

3. Implement Newton's Method. Consider the following equation:

$$x = \tan x$$

There are an infinite number of solutions x to this equation. Use Newton's method (and any techniques you need to start Newton in regions of convergence) to find the two solutions on either side of 11. (Said differently: Find one solution x_{low} less than 11 and one solution x_{high} greater than 11 such that the interval $[x_{low}, x_{high}]$ contains no other solutions.)

SOLUTION: The implementation is in `code/q3.m`

To solve equation $x = \tan x$, first we transfer the problem to be finding the roots of function $f(x) = \tan x - x$. So we have $f'(x) = \frac{1}{\cos^2 x} - 1$ and $f''(x) = \frac{2 \sin x}{\cos^3 x}$.

For solution x_{low} where $x_{low} < 11$, I first plot function $f(x) = \tan x - x$ in interval $[10, 11]$ and also the x-axis on the image to find the convergence region. The curve is shown as Fig. 1

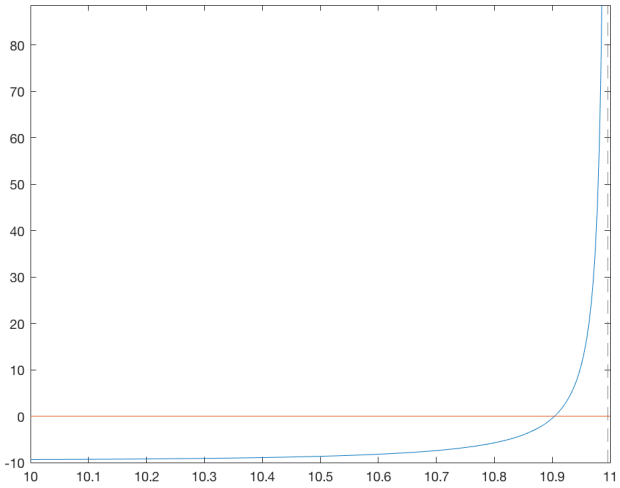


Figure 1: $f(x) = \tan x - x$ in region $[10, 11]$

We can see from the image that the root is around 10.9 and we have $f(10.9) \approx -0.4688 < 0$ and $f(10.95) > 0$. Select the region $[10.9, 10.95]$. Since
for all $x \in [10.9, 10.95]$, $f'(x) \neq 0$ and $f''(x) > 0$
$|f(10.9)/f'(10.9)| = 0.0043 < 10.95 - 10.9 = 0.05$
$|f(10.95)/f'(10.95)| = 0.0228 < 10.95 - 10.9 = 0.05$.
So $[10.9, 10.95]$ is a convergence region. Set the initial root guess as $x_0 = 10.9$, and then call function `zero_solution`, finally get the result $x_{low} = 10.9041$.

For solution x_{high} where $x_{high} > 11$, we can infer that the next root for $f(x)$ would be around 14 because of the periodicity of $\tan x$. So plot $f(x) = \tan x - x$ in interval $[13, 15]$, the curve is shown as Fig. 2.

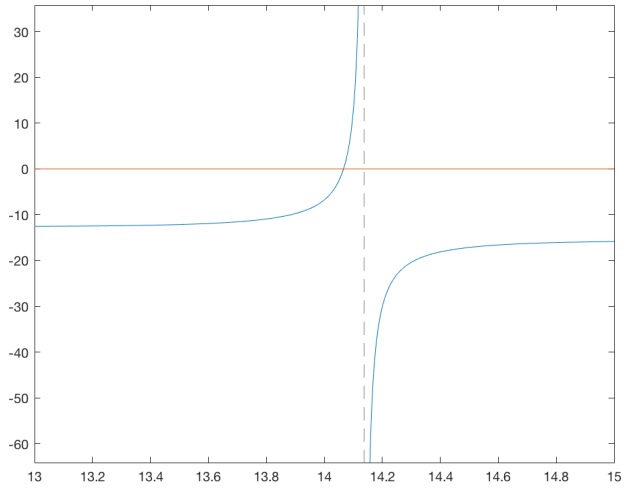


Figure 2: $f(x) = \tan x - x$ in region [13, 15]

Since
$f(14) < 0$, $f(14.15) \approx 12.7932 > 0$
for all $x \in [14, 14.15]$, $f'(x) \neq 0$ and $f''(x) > 0$
$|f(14)/f'(14)| = 0.1287 < 14.15 - 14 = 0.15$
$|f(14.15)/f'(14.15)| = 0.0152 < 14.15 - 14 = 0.15$

So $[14, 14.15]$ is a convergence region. Set the initial root guess as $x_0 = 14$, and then call function `zero_solution`, finally get the result $x_{high} = 14.0662$.

So, finally get solutions: $x_{low} =$ `10.9041`, $x_{low} =$ `14.0662`

4. If ξ is a root of $f(x)$ of order 2, then $f(\xi) = 0$, $f'(\xi) = 0$ and $f''(\xi) \neq 0$.
Show that in this case Newtons method no longer converges quadratically. Do so by showing that the method now converges linearly. (You may assume that $f''(x)$ is continuous in a neighborhood of ξ.)
Now suppose that $f'''(x)$ is continuous in a neighborhood of ξ. Show that the iteration

$$x_{n+1} = x_n - 2\frac{f(x_n)}{f'(x_n)}$$

does converge quadratically.

SOLUTION:

• For Newtons method, we have

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Let $x_0, x_1, ...$ be a sequence which converges to ξ. Let

$$\varepsilon_{n+1} = \xi - x_{n+1}, \quad \varepsilon_n = \xi - x_n$$

So, there is

$$\begin{aligned}\varepsilon_{n+1} - \varepsilon_n &= x_n - x_{n+1} \\ &= \frac{f(x_n)}{f'(x_n)} \\ &= \frac{f(\xi - \varepsilon_n)}{f'(\xi - \varepsilon_n)}\end{aligned}$$

Expand $f(x)$ about the root ξ with Taylor expansion, we obtain:

$$f(\xi + \varepsilon) = f(\xi) + \varepsilon f'(\xi) + \frac{\varepsilon^2}{2}f''(\xi) + \frac{\varepsilon^3}{6}f'''(\xi) + ...$$

So,

$$\begin{aligned}\varepsilon_{n+1} - \varepsilon_n &= \frac{f(\xi - \varepsilon_n)}{f'(\xi - \varepsilon_n)} \\ &= \frac{f(\xi) - \varepsilon_n f'(\xi) + \frac{\varepsilon_n^2}{2}f''(\xi) - \frac{\varepsilon_n^3}{6}f'''(\xi) + ...}{f'(\xi) - \varepsilon_n f''(\xi) + \frac{\varepsilon_n^2}{2}f'''(\xi) + ...} \\ &= \frac{\frac{\varepsilon_n^2}{2}f''(\xi) - \frac{\varepsilon_n^3}{6}f'''(\xi) + ...}{-\varepsilon_n f''(\xi) + \frac{\varepsilon_n^2}{2}f'''(\xi) + ...} \\ &= \varepsilon_n\left(-\frac{1}{2} + \frac{\frac{\varepsilon_n}{12}f'''(\xi) + ...}{-f''(\xi) + \frac{\varepsilon_n}{2}f'''(\xi) + ...}\right)\end{aligned}$$

Thus, we have:

$$\varepsilon_{n+1} = \varepsilon_n\left(\frac{1}{2} + \frac{\frac{\varepsilon_n}{12}f'''(\xi) + ...}{-f''(\xi) + \frac{\varepsilon_n}{2}f'''(\xi) + ...}\right)$$

$$\lim_{n\to\infty}\frac{|\varepsilon_{n+1}|}{|\varepsilon_n|}=C$$

So, in this case Newtons method no longer converges quadratically, this method now converges linearly.

• For the iteration

$$x_{n+1}=x_n-2\frac{f(x_n)}{f'(x_n)}$$

there is

$$\begin{aligned}
\varepsilon_{n+1}-\varepsilon_n &= x_n-x_{n+1}\\
&= 2\frac{f(x_n)}{f'(x_n)}\\
&= 2\frac{f(\xi-\varepsilon_n)}{f'(\xi-\varepsilon_n)}\\
&= 2\frac{f(\xi)-\varepsilon_n f'(\xi)+\frac{\varepsilon_n^2}{2}f''(\xi)-\frac{\varepsilon_n^3}{6}f'''(\xi)+...}{f'(\xi)-\varepsilon_n f''(\xi)+\frac{\varepsilon_n^2}{2}f'''(\xi)+...}\\
&= \frac{\varepsilon_n^2 f''(\xi)-\frac{\varepsilon_n^3}{3}f'''(\xi)+...}{-\varepsilon_n f''(\xi)+\frac{\varepsilon_n^2}{2}f'''(\xi)+...}\\
&= \varepsilon_n\frac{f''(\xi)-\frac{\varepsilon_n}{3}f'''(\xi)...}{-f''(\xi)+\frac{\varepsilon_n}{2}f'''(\xi)+...}\\
&= \varepsilon_n\left(-1+\frac{\frac{\varepsilon_n}{6}f'''(\xi)+...}{-f''(\xi)+\frac{\varepsilon_n}{2}f'''(\xi)+...}\right)\\
&= -\varepsilon_n+\frac{\varepsilon_n^2}{6}\times\frac{f'''(\xi)+...}{-f''(\xi)+\frac{\varepsilon_n}{2}f'''(\xi)+...}
\end{aligned}$$

So,

$$\varepsilon_{n+1}=\frac{\varepsilon_n^2}{6}\times\frac{f'''(\xi)+...}{-f''(\xi)+\frac{\varepsilon_n}{2}f'''(\xi)+...}$$

$$\lim_{n\to\infty}\frac{|\varepsilon_{n+1}|}{|\varepsilon_n|^2}=C$$

Thus, with this iteration $x_{n+1}=x_n-2f(x_n)/f'(x_n)$, it converges quadratically.

5. (a) Implement Müller's method.

SOLUTION: The implementation is in `code/q5.m`

In the script, the core function of this implementation is function `muller`, which is used for finding all the real and complex roots for the polynomial. In order to reach the goal, three additional functions, namely `fx`, `gx` and `get_coefficient` are implemented. Below is the details about these functions.

`muller`: The inputs of this function include `x0`, `x1`, `x2` and `n`, where `x0`, `x1`, `x2` are the initial guesses of the roots, `n` is the degree of the polynomial, by which we would know how many roots the polynomial has in total. The output of this function is `approx` with size $(1, n)$, each entry in `approx` is a root of the polynomial. For each root, start with the initial guesses and calculate x_i according to the strategy of Müller's method until the termination condition. In this implementation, I choose $\Delta x_i \leq 5 \times 10^{-5}$ as the termination condition. After the root is found, deflate the function and then find the next root until all the n roots are found.

`fx`: This function is used to define the polynomial that we need to find roots for. The input of this function is `x`, the output `y` is the corresponding value of the polynomial given the input `x`.

`gx`: This function is used for deflating the polynomial defined in `fx`. The inputs of this function are `x` and `roots`, where `roots` is an vector where each entry is a known root of the polynomial $f(x)$. The deflated function can be represented as $g(x) = f(x)/(\prod_{i=1}^{k}(x - r_k))$, where r_k is one of the entry of `roots`. The output of this function `gx` is the value of the deflated polynomial given the input `x`.

`get_coefficient`: This function is used to get the coefficients a, b and c of the interpolating polynomial $p(x) = c + b(x - x_2) + a(x - x_2)^2$ in Müller's method. The inputs of this function are `x0`, `x1`, `x2`, `y0`, `y1` and `y2`, where (`x0`, `y0`), (`x1`, `y1`), (`x2`, `y2`) are three points used for interpolating. The output is `[a, b, c]`.

(b) Use Müller's method to find all the real and complex roots of the polynomial

$$p(x) = x^3 - 4x^2 + 6x - 4.$$

SOLUTION: First define the target polynomial $p(x) = x^3 - 4x^2 + 6x - 4$ in function `fx`. We can know that this polynomial is in degree 3, so it has three roots (real or complex). I set the initial guesses of the root as: 0.5, 1.0, 1.5. So the inputs' values of function `muller` would be `x0` $= 0.5$, `x1` $= 1.0$, `x2` $= 1.5$ and `n` $= 3$. By calling the function `muller`, finally obtain returned value:

$$[1.0000 - 1.0000i, 1.0000 + 1.0000i, 2.0000 + 0.0000i]$$

So the roots of the polynomial $p(x) = x^3 - 4x^2 + 6x - 4$ are:

$$x_1 = 1 - i, \quad x_2 = 1 + i, \quad x_3 = 2$$

6. Consider the two univariate polynomials

$$p(x) = x^3 - 9x^2 + 26x - 24$$

$$q(x) = x^2 + 3x - 10$$

(a) Using resultants decide whether $p(x)$ and $q(x)$ share a common root.

SOLUTION:

To decide whether $p(x)$ and $q(x)$ share a common root, we set $p(x) = 0$ and $q(x) = 0$, and obtain the simultaneous equations:

$$\begin{cases} x^3 - 9x^2 + 26x - 24 = 0 \\ x^2 + 3x - 10 = 0 \end{cases}$$

Further, we can get the equations as below:

$$\begin{cases} x^4 - 9x^3 + 26x^2 - 24x = 0 \\ x^3 - 9x^2 + 26x - 24 = 0 \\ x^4 + 3x^3 - 10x^2 = 0 \\ x^3 + 3x^2 - 10x = 0 \\ x^2 + 3x - 10 = 0 \end{cases}$$

We can write the simultaneous equations above in matrix format as $Q\vec{x} = 0$:

$$\begin{bmatrix} 1 & -9 & 26 & -24 & 0 \\ 0 & 1 & -9 & 26 & -24 \\ 1 & 3 & -10 & 0 & 0 \\ 0 & 1 & 3 & -10 & 0 \\ 0 & 0 & 1 & 3 & -10 \end{bmatrix} \begin{bmatrix} x^4 \\ x^3 \\ x^2 \\ x \\ 1 \end{bmatrix} = 0$$

$Q\vec{x} = 0$ has a solution if and only if $\det Q = 0$

$$\det Q = \begin{vmatrix} 1 & -9 & 26 & -24 & 0 \\ 0 & 1 & -9 & 26 & -24 \\ 1 & 3 & -10 & 0 & 0 \\ 0 & 1 & 3 & -10 & 0 \\ 0 & 0 & 1 & 3 & -10 \end{vmatrix} = 0$$

So $p(x)$ and $q(x)$ share a common root.

(b) If the two polynomials share a common root, use the ratio method discussed in class to find that root.

SOLUTION:

The common root is:

$$x = -\frac{\det Q_1}{\det Q_2}$$

where Q_i is formed by deleting the i^{th} column and the last row. So,

$$Q_1 = \begin{bmatrix} -9 & 26 & -24 & 0 \\ 1 & -9 & 26 & -24 \\ 3 & -10 & 0 & 0 \\ 1 & 3 & -10 & 0 \end{bmatrix}, \quad Q_2 = \begin{bmatrix} 1 & 26 & -24 & 0 \\ 0 & -9 & 26 & -24 \\ 1 & -10 & 0 & 0 \\ 0 & 3 & -10 & 0 \end{bmatrix}$$

Thus, we have

$$x = (-1)\frac{\begin{vmatrix} -9 & 26 & -24 & 0 \\ 1 & -9 & 26 & -24 \\ 3 & -10 & 0 & 0 \\ 1 & 3 & -10 & 0 \end{vmatrix}}{\begin{vmatrix} 1 & 26 & -24 & 0 \\ 0 & -9 & 26 & -24 \\ 1 & -10 & 0 & 0 \\ 0 & 3 & -10 & 0 \end{vmatrix}} = 2$$

7. Consider the two bivariate polynomials

$$p(x, y) = 2x^2 + 2y^2 - 1$$

$$q(x, y) = x^2 + y^2 + 2xy - x + y$$

(a) Sketch the two zero contours $p(x, y) = 0$ and $q(x, y) = 0$, for real vales of x and y.

SOLUTION: Using the built-in function `fimplicit` in Matlab, the zero contours $p(x, y) = 0$ and $q(x, y) = 0$ are shown as Fig. 3.

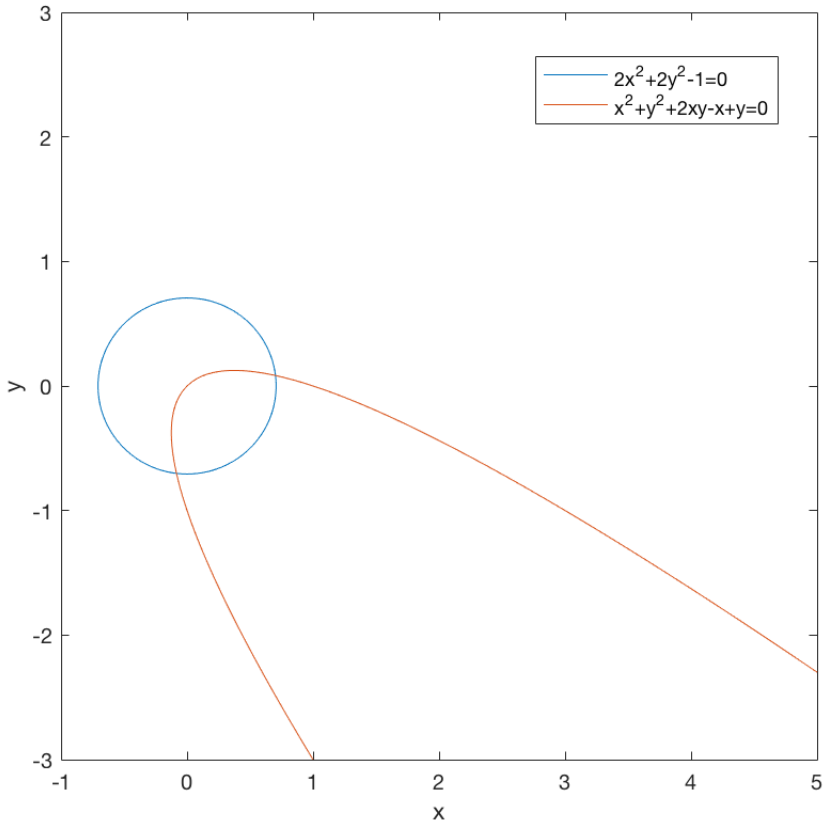


Figure 3: The zero contours of $p(x, y) = 0$ and $q(x, y) = 0$.

(b) Using resultants, find the intersection points of those two contours (consider only real values of x and y). Do so by eliminating y and constructing a resultant that is a function of x.

SOLUTION:

$$\begin{cases} p(x, y) = 2x^2 + 2y^2 - 1 \\ q(x, y) = x^2 + y^2 + 2xy - x + y \end{cases}$$

$$\Rightarrow \quad \begin{cases} p(y) = 2y^2 + (2x^2 - 1) \\ q(y) = y^2 + (2x+1)y + (x^2 - x) \end{cases}$$

Further, we can get the equations as below:

$$\begin{cases} 2y^3 + 0y^2 + (2x^2 - 1)y + 0 = 0 \\ 2y^2 + 0y + (2x^2 - 1) = 0 \\ y^3 + (2x+1)y^2 + (x^2 - x)y + 0 = 0 \\ y^2 + (2x+1)y + (x^2 - x) = 0 \end{cases}$$

We can write the simultaneous equations above in matrix format as $Q\vec{y} = 0$:

$$\begin{bmatrix} 2 & 0 & 2x^2-1 & 0 \\ 0 & 2 & 0 & 2x^2-1 \\ 1 & 2x+1 & x^2-x & 0 \\ 0 & 1 & 2x+1 & x^2-x \end{bmatrix} \begin{bmatrix} y^3 \\ y^2 \\ y \\ 1 \end{bmatrix} = 0$$

Since these two contours have intersection points, $Q\vec{y} = 0$ has solution. So $\det Q = 0$.

$$\begin{aligned} \det Q &= \begin{vmatrix} 2 & 0 & 2x^2-1 & 0 \\ 0 & 2 & 0 & 2x^2-1 \\ 1 & 2x+1 & x^2-x & 0 \\ 0 & 1 & 2x+1 & x^2-x \end{vmatrix} \\ &= 2 \begin{vmatrix} 2 & 0 & 2x^2-1 \\ 2x+1 & x^2-x & 0 \\ 1 & 2x+1 & x^2-x \end{vmatrix} + (2x^2-1) \begin{vmatrix} 0 & 2 & 2x^2-1 \\ 1 & 2x+1 & 0 \\ 0 & 1 & x^2-x \end{vmatrix} \\ &= 16x^4 + 16x^3 - 12x - 1 = 0 \end{aligned}$$

By solving this equation, we can get two real roots as below:

$$x = \frac{\sqrt{5}-1}{4} \pm \sqrt{\frac{\sqrt{5}-1}{8}}$$

So we can get the approximate values of x:

$$x_1 \approx 0.7021, \quad x_2 \approx -0.0841$$

Plug in x into $p(y)$ and $q(y)$, finally can get the values of y:

$$y_1 \approx 0.0841 \quad y_2 \approx -0.7021$$

So the intersection points of these two contours are:

$$(0.7021, 0.0841), \quad (-0.0841, -0.7021)$$

(c) Mark the roots of the resultant constructed in part (b) on the x-axis of your sketch from part (a).

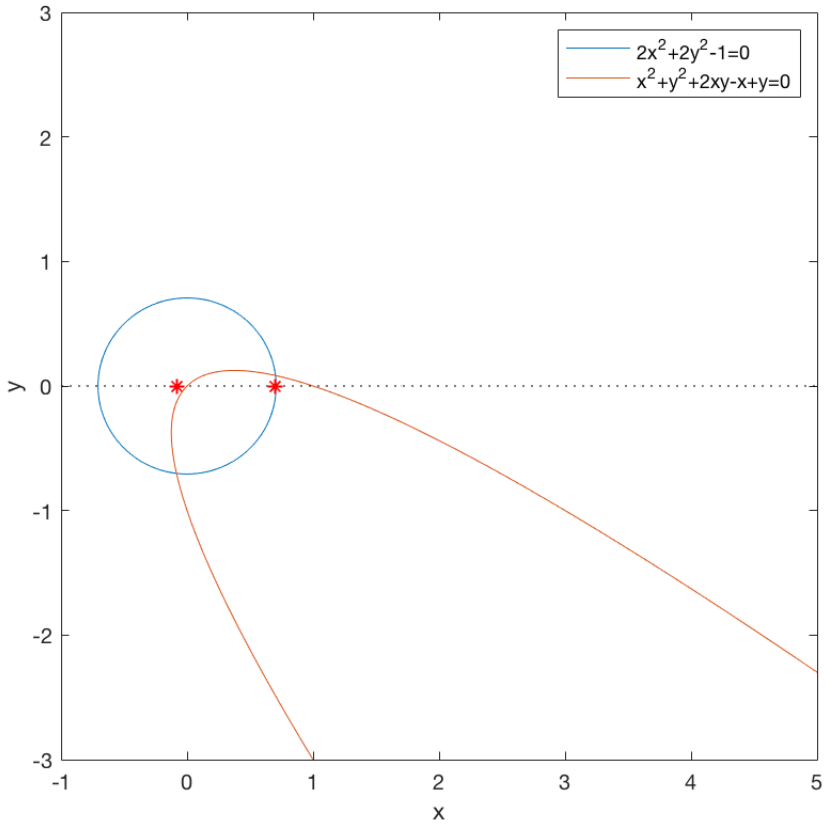
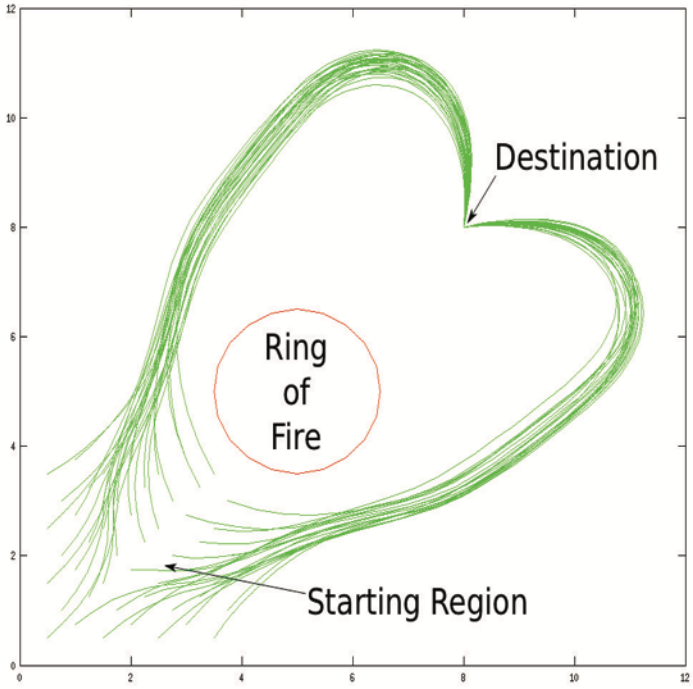


Figure 4: Plot the roots of the resultant on x-axis.

8. You are preparing a robotic unicycle to take part in a circus act. For most of the show, a talented acrobat rides the unicycle. But at one point, the acrobat jumps off the unicycle onto a trapeze. After a short trapeze act, the acrobat leaps through the ring of fire in the center of the stage to land on the waiting unicycle, which has moved autonomously to the other side. Your job is to plan a path for the unicycle to take around the ring of fire to the acrobat's landing point.
   You are given a precomputed set of paths which all begin at different points, avoid the circle of fire, and end at the destination (shown below). You must mimic these paths as closely as possible, since they are precisely choreographed for the circus act. However, the acrobat is only human, and does not position the unicycle precisely at any of the paths starting points. You will need to interpolate a new path from other paths with nearby starting points.



   The destination point is $(8, 8)$, and the ring of fire, a circle of radius 1.5, is centered at $(5, 5)$. The precomputed paths are given in the text file `paths.txt`. Every pair of lines in the text file represents a path, which is a sequence of 50 points. The first line contains the x coordinates, and the second contains the y coordinates for one path. The file format is:

$$\begin{array}{ccccc} x_1^{(1)} & x_2^{(1)} & \dots & x_{49}^{(1)} & x_{50}^{(1)} \\ y_1^{(1)} & y_2^{(1)} & \dots & y_{49}^{(1)} & y_{50}^{(1)} \\ x_1^{(2)} & x_2^{(2)} & \dots & x_{49}^{(2)} & x_{50}^{(2)} \\ y_1^{(2)} & y_2^{(2)} & \dots & y_{49}^{(2)} & y_{50}^{(2)} \\ \dots & \dots & \dots & \dots & \dots \end{array}$$

(a) Write a system of linear equations (in the form $Av = b$, with v representing variables of some sort, appropriately chosen) and constraints (for instance, $v_1 \geq 0$) that will help you determine whether a 2D point (x, y) falls within the triangle formed by three 2D points $(x^{(i)}, y^{(i)}), (x^{(j)}, y^{(j)})$, and $(x^{(k)}, y^{(k)})$. (Part of the problem is to think about how you might do this.)

Denote the three 2D points as p^i, p^j and p^k respectively. Namely, $p^i = (x^{(i)}, y^{(i)})$, $p^j = (x^{(j)}, y^{(j)})$, $p^k = (x^{(k)}, y^{(k)})$. So we have vector:

$$\overrightarrow{p^i p^j} = (x^{(j)} - x^{(i)}, y^{(j)} - y^{(i)})$$

$$\overrightarrow{p^i p^k} = (x^{(k)} - x^{(i)}, y^{(k)} - y^{(i)})$$

Choose $\overrightarrow{p^i p^j}$ and $\overrightarrow{p^i p^k}$ as the basis in the 2D space, then each 2D point can be represented as:

$$p = (x, y) = p_i + v_1 \overrightarrow{p^i p^j} + v_2 \overrightarrow{p^i p^k}$$

and can be shown as in Fig. 5

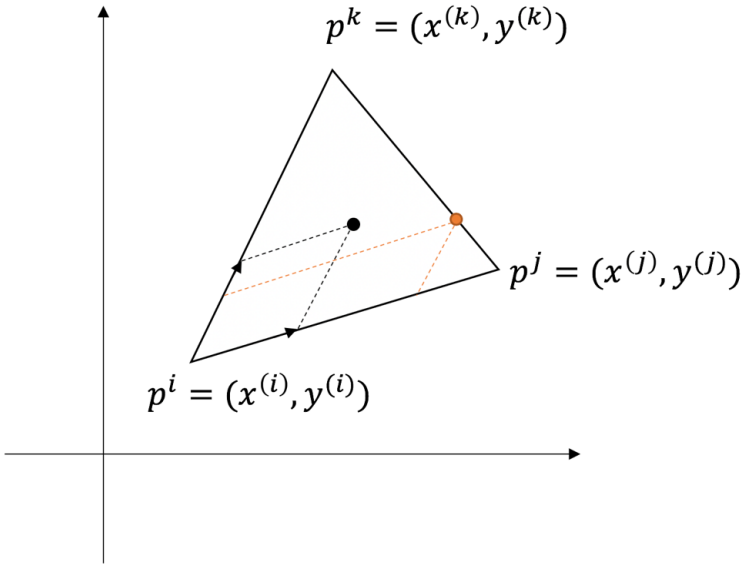


Figure 5: Points in 2D space.

From the equation above, we can get that:

$$x = x^{(i)} + v_1(x^{(j)} - x^{(i)}) + v_2(x^{(k)} - x^{(i)})$$

$$y = y^{(i)} + v_1(y^{(j)} - y^{(i)}) + v_2(y^{(k)} - y^{(i)})$$

To make sure that point p is in the triangle formed by p^i, p^j and p^k, obviously, there are constrains: $0 \leq v_1 \leq 1$, $0 \leq v_2 \leq 1$ and $v_1 + v_2 \leq 1$.

So the system of linear equations (in the form $Av = b$) can be represented as:

$$\begin{bmatrix} x^{(j)} - x^{(i)} & x^{(k)} - x^{(i)} \\ y^{(j)} - y^{(i)} & y^{(k)} - y^{(i)} \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} x - x^{(i)} \\ y - y^{(i)} \end{bmatrix}$$

with constrains:

$$0 \leq v_1 \leq 1, \quad 0 \leq v_2 \leq 1, \quad \textit{and} \quad v_1 + v_2 \leq 1$$

(b) Implement an algorithm to interpolate three paths. Assume the algorithm is given the unicycle's starting location at time $t = 0$.

Your algorithm should first pick three paths $p^{(i)}$, $p^{(j)}$, and $p^{(k)}$ (from the file `paths.txt`) to interpolate.

Then your algorithm should construct a new path p as a weighted sum of these three paths. So, at each time t, $p(t) = \alpha_i p^{(i)}(t) + \alpha_j p^{(j)}(t) + \alpha_k p^{(k)}(t)$, with $p(0)$ the given starting location of the unicycle and with the weights α_i, α_j, α_k fixed throughout.

[Notation: For a path p, $p(t)$ is the 2D point $(x(t), y(t))$ that describes the location of the path at time t.

Your algorithm should be able to produce a value $p(t)$ for all relevant times t prior to reaching the Destination, not just for the discrete time snapshots given in `paths.txt`. These points $p(t)$ should avoid the ring of fire.]

Hint: The unicycle's starting position should fall within the triangle formed by the three paths' starting points, but there may be many valid sets of such starting points. So you should develop criteria to choose one set.

Discuss any decisions you made in your implementation. How did you pick a set of paths? How did you choose the weights α_i, α_j, α_k? How did you decide on a time scale for t?

Specificity: You only need to write code to solve the particular problem (with the particular destination, ring of fire, and precomputed paths) described here, not a general purpose algorithm.

SOLUTION: The implementation is in `code/q8.m`

The main idea to solve this problem is that, first separate the paths into two groups. One group includes all the paths that are above the ring of fire; another group contains all the paths that are below the ring of fire. The separation result can be seen as Fig. 6.

When we try to select three paths, we would just select paths from the same group so that the constructed path would not pass through the ring of fire. Calculate

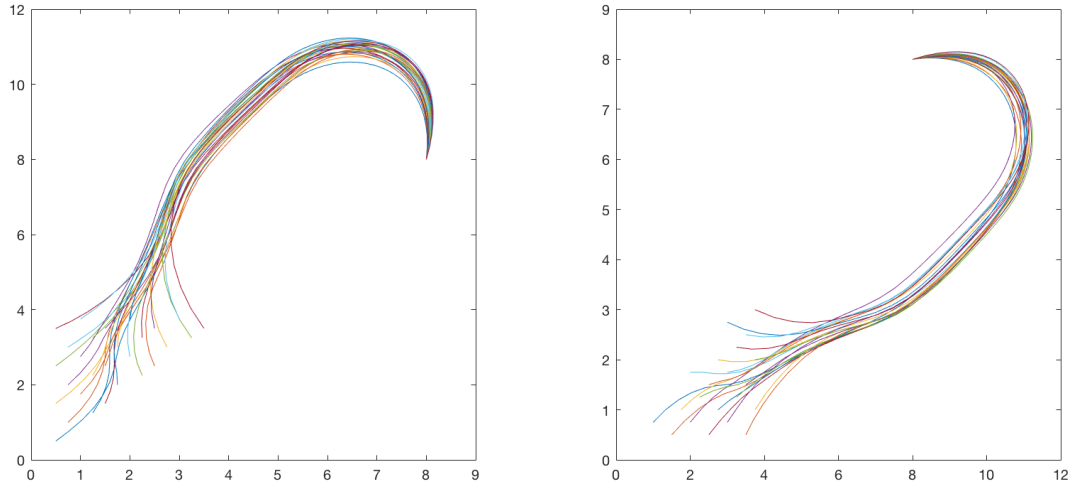


Figure 6: Separation of the paths.

the center point of paths' starting points for each group, then when given a start point, we can determine which path group to choose by calculating the distance between start point and the center points. Search all the possible combinations of three paths in the chosen group, select the path sets where the start point is located within the triangle formed by the three paths' starting points. All these path sets are candidates. Then select the path set that makes the start point is closest to the centroid of the triangle. This set of paths would finally be used for interpolation.

After determine the path set, let the starting points of these three paths are (x_1, y_1), (x_2, y_2), (x_3, y_3) respectively. Denote the actual starting point as (x, y). Since we need to finally reach the destination point, there must be $\alpha_1+\alpha_2+\alpha_3 = 1$. Then we have:

$$\alpha_1 x_1 + \alpha_2 x_2 + (1 - \alpha_1 - \alpha_2)x_3 = x$$

$$\alpha_1 y_1 + \alpha_2 y_2 + (1 - \alpha_1 - \alpha_2)y_3 = y$$

So, it can be written as:

$$\begin{bmatrix} x_1 - x_3 & x_2 - x_3 \\ y_1 - y_3 & y_2 - y_3 \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix} = \begin{bmatrix} x - x_3 \\ y - y_3 \end{bmatrix}$$

By solving this equation, we can finally obtain the values of $\alpha_1, \alpha_2, \alpha_3$.

After getting the values of $\alpha_1, \alpha_2, \alpha_3$, we can calculate 50 discrete data points for the new path. In order to get the location of the path at any time t, we can interpolate $x(t)$ at points $(t_1, x_1^{new}), (t_2, x_2^{new}), ..., (t_50, x_50^{new})$, and also interpolate $y(t)$ at points $(t_1, y_1^{new}), (t_2, y_2^{new}), ..., (t_{50}, y_{50}^{new})$ to get the interpolation polynomial with degree of 2. Use the function implemented in problem 1 and select 3 points from all the points that make the error of interpolation polynomial reaches the

smallest. The interpolation polynomial then can describe location for all relevant times t before reaching the destination.

For the path interpolation, we assume that the discrete points are evenly distributed respect to the time. Then for each time t, we can scale it to the range (0, 1) according to the total time and then find the nearest two discrete points. By linear interpolation between the two discrete points, we can get the estimated point at time t.

In this script, there are 7 functions implemented to solve the problem. The details are listed as below:

• `main`: The main function to solve the problem.
The inputs include:
`paths` (the loaded path data from `paths.txt`);
`start_point` (the coordinates of the start point);
`fire_center` (the coordinates of the fire center point).
The outputs include:
`interp_x` (the estimated values of x with the interpolation);
`interp_y` (the estimated values of y with the interpolation).

• `split_path`: This is used for path separation. One is the path group that above the fire ring, another path group include all paths below the fire ring.
The inputs include:
`paths` (the loaded path data from `paths.txt`);
`fire_center` (the coordinates of the fire center point)
The outputs include:
`up_path_x` (the x values of paths that are above fire ring);
`up_path_y`(the y values of paths that are above fire ring);
`low_path_x`(the x values of paths that are below fire ring);
`low_path_y`(the y values of paths that are below fire ring)

• `is_in_triangle`: This is to judge whether the start point is in the triangle (or on the line) formed by the given 3 points. If it is in the triangle or on the line formed by the 3 points, return true, else return false.
The inputs include:
`sp` (the coordinates of the start point)
`p1, p2, p3` (the coordinates of the 3 points that forms a triangle or a line)
The output is:
`is_in` (true or false)

• `pick_path`: This is used to pick 3 paths from all the paths in order to construct the new path.
The inputs include:
`up_path_x, up_path_y, low_path_x, low_path_y` (the x and y values of paths that are above and below the fire ring)
`start_point` (the coordinates of the start point)
The outputs include:

`path_set_x` (the x values of the 3 selected paths)
`path_set_y` (the y values of the 3 selected paths)

• `get_alpha`: This is to determine the weights $\alpha_1, \alpha_2, \alpha_3$.
The inputs include:
`sp` (the coordinates of the start point)
`p1, p2, p3` (the coordinates of the 3 points that forms a triangle or a line)
The output is:
`a` ($[\alpha_1, \alpha_2, \alpha_3]$)

• `get_new_path`: This is to generate the discrete points for new path.
The inputs include:
`a` ($[\alpha_1, \alpha_2, \alpha_3]$)
`path_set_x, path_set_y` (the x and y values respectively of the 3 selected paths)
The output include:
`new_path_x, new_path_y` (the value of x and y respectively for the new path)

• `interpolate`: This is to find the interpolation value given the discrete points and the expected time t.
The inputs include:
`data`: The values of the discrete points.
`t`: The time t that is scaled to the range (0, 1).
The outputs include:
`interp_result`: The estimated value at the given time t.

(c) Interpolate paths for the starting points (0.8, 1.8), (2.2, 1.0) and (2.7, 1.4). For each starting point, on the same graph, plot the ring of fire, the three paths being interpolated, and the interpolated path. Your paths should not intersect the ring of fire.

SOLUTION:

• For starting point (0.8, 1.8), the selected paths and the interpolating path are shown as Fig. 7:

In this figure, the 3 green paths are the selected paths for interpolating. The blue line is the final interpolating result.

• For starting point (2.2, 1.0), the selected paths and the interpolating path are shown as Fig. 8:

In this figure, the 3 green paths are the selected paths for interpolating. The blue line is the final interpolating result.

• For starting point (2.7, 1.4), the selected paths and the interpolating path are shown as Fig. 9:

In this figure, the 3 green paths are the selected paths for interpolating. The blue

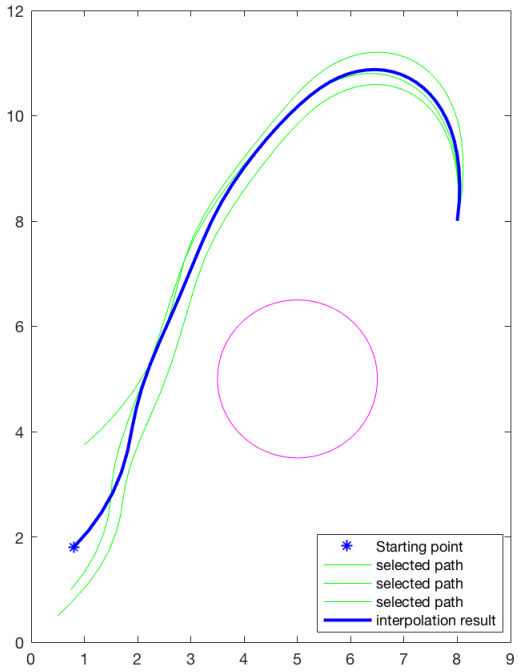


Figure 7: The selected paths and the interpolating path for starting point (0.8, 1.8)

line is the final interpolating result.

(d) Discuss how your algorithm would need to be modified (if at all) if more obstacles were to be introduced.

SOLUTION: If more obstacles were to be introduced, then at the beginning, the separation of the paths would need to be modified so that each separated group of paths would only have paths that gather together, without paths locating at different sides of the obstacles.

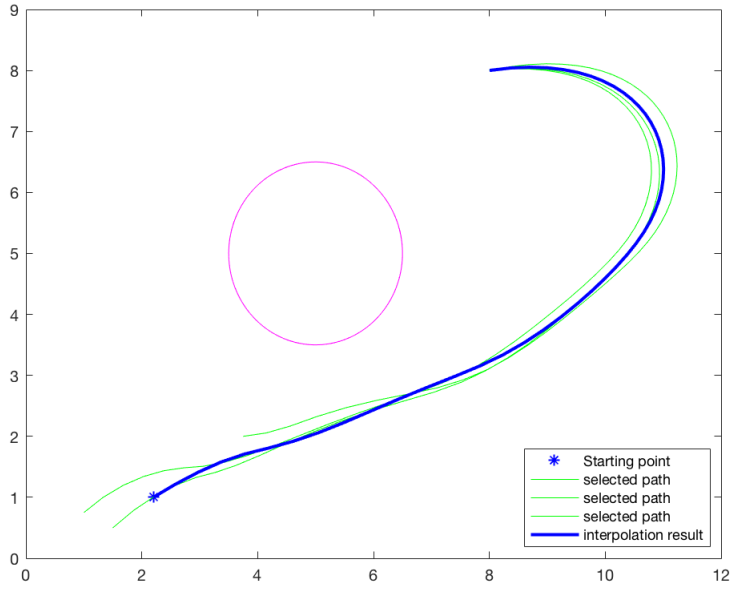


Figure 8: The selected paths and the interpolating path for starting point (2.2, 1.0)

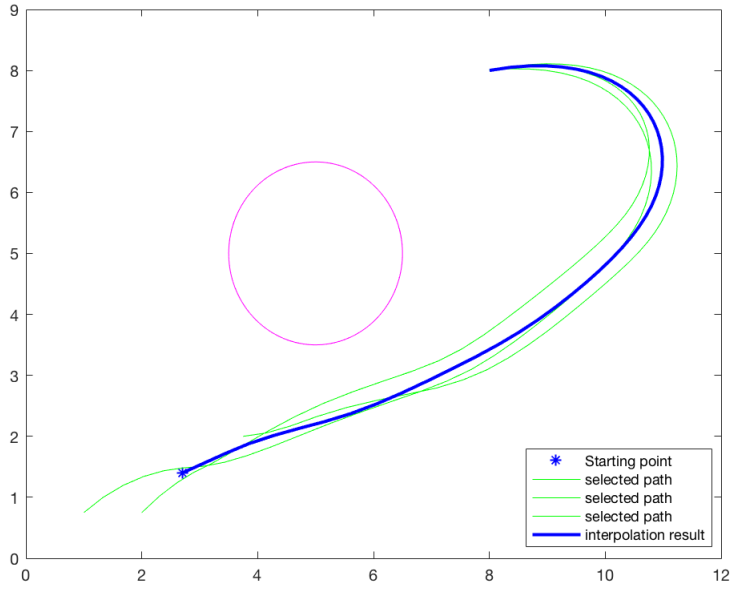


Figure 9: The selected paths and the interpolating path for starting point (2.7, 1.4)