16-811: Math Fundamentals for Robotics, Fall 2025

Assignment 2

DUE: Thursday, September 25, 2025, by 11:59pm

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

$$x, x_0, \ldots, x_n, f(x_0), \ldots, f(x_n) \qquad \text{(with distinct } x_0, x_1, \ldots, x_n\text{)}.$$

The procedure should compute an interpolated value for $f(x)$ based on the given data points $(x_0, f(x_0)), (x_1, f(x_1)), \ldots, (x_n, f(x_n))$.

Note: The procedure should use all the data points $(x_i, f(x_i))$, $i = 0, \ldots, n$, effectively implementing an interpolating polynomial of degree n (or less, depending on the data).

(b) Use your procedure to interpolate $\cos(\pi x)$ at $x = \frac{3}{10}$, based on known values of $(x, \cos(\pi x))$ at the following x locations: 0, $\frac{1}{8}$, $\frac{1}{4}$, $\frac{3}{8}$, $\frac{1}{2}$.

(c) Now consider the function

$$f(x) = \frac{2}{1 + 9\,x^2},$$

with input data given at the points

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

Use your procedure to estimate $f(x)$ at $x = 0.07$, with $n = 2$.
Use your procedure to estimate $f(x)$ at $x = 0.07$, with $n = 4$.
Use your procedure to estimate $f(x)$ at $x = 0.07$, with $n = 40$.
What is the actual value of $f(0.07)$?

(d) 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 $[-1, 1]$ very finely – much more finely than the discretization implied by n. Then 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 $f(x) = 2/(1 + 9\,x^2)$ given above and $p_n(x)$ the interpolating polynomial based on n using data as in part (c).

Do the error estimates make sense? Explain your results.

2. Suppose you wish to build a "sliding-window interpolation table" (as discussed in class) 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[1], assuming that you will use linear interpolation between adjacent points in the table? How fine must the table spacing be if you will use quadratic interpolation? In each case, how many entries do you need in the table?

[1] Terminology: n decimal digit accuracy means the error is less than $\frac{1}{2} \cdot 10^{-n}$, so in this case less than $5 \cdot 10^{-7}$.

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 to find the two solutions on either side of 15. In other words, find two solutions $x_{low} < 15 < x_{high}$ such that the interval $[x_{low}, x_{high}]$ contains no other solutions. — Use any techniques you need to start Newton in regions of convergence. Those regions may be very small.

4. Suppose ξ is a root of order 2 of $f(x)$, meaning $f(\xi) = 0$, $f'(\xi) = 0$ and $f''(\xi) \neq 0$.

   (a) Show that in this case Newton's method no longer converges quadratically. Do so by showing that the method now converges linearly.

   (b) We can modify Newton's method in this case: Show that the iteration

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

   does converge quadratically (or faster) when ξ is a root of order 2 of $f(x)$.

   Throughout this problem you may assume that $f(x)$ has as many continuous derivatives as you need, for instance that $f'''(x)$ is continuous in a neighborhood of ξ.

   Recommendation: Use a Taylor series for $h(x) = f(x)/f'(x)$, similar to what we did in lecture when examining convergence of Newton's method. You do not need anything more complicated. However, you will find it useful to remember L'Hôpital's rule from calculus when computing $h(\xi)$ and $h'(\xi)$. Do not worry about the exact value of $h''(\xi)$.

5. (a) Implement Müller's method.

   (b) Use Müller's method to find good numerical approximations for all the real and complex roots of the polynomial $p(x) = x^3 + x + 1$.

6. Consider the two univariate polynomials

$$\begin{aligned} p(x) &= x^3 - 3x^2 + x - 3 \\ q(x) &= x^2 + x - 12 \end{aligned}$$

   (a) Using the method of resultants, decide whether $p(x)$ 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.

7. Consider the two bivariate polynomials

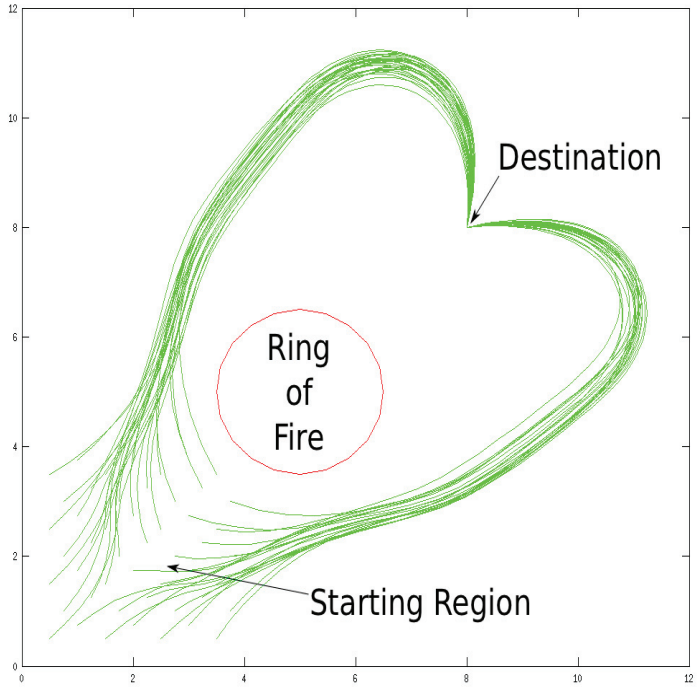
$$\begin{aligned} p(x,y) &= 2x^2 + 2y^2 - 4x - 4y + 3 \\ q(x,y) &= x^2 + y^2 + 2xy - 5x - 3y + 4 \end{aligned}$$

   (a) Draw the zero contour $p(x, y) = 0$ and the zero contour $q(x, y) = 0$, with x and y real.

   (b) Using the method of resultants, solve for the intersection points of these contours, i.e., find all (x, y) for which $p(x, y) = 0 = q(x, y)$, with x and y real. Do so by treating the polynomials p and q as functions of y, temporarily viewing x as a constant. Obtain a 4×4 matrix whose determinant, now a function of x, is the desired resultant. Find the x-roots of that determinant, thereby projecting the intersection points onto the x-axis. Use those x-roots to find the y-coordinates of the intersection points. Verify your intersection points.

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 ring 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 $(8, 8)$. The Ring of Fire is a circle of radius 1.5 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:

$$\begin{array}{ccccc} x_0^{(1)} & x_1^{(1)} & \dots & x_{48}^{(1)} & x_{49}^{(1)} \\ y_0^{(1)} & y_1^{(1)} & \dots & y_{48}^{(1)} & y_{49}^{(1)} \\ x_0^{(2)} & x_1^{(2)} & \dots & x_{48}^{(2)} & x_{49}^{(2)} \\ y_0^{(2)} & y_1^{(2)} & \dots & y_{48}^{(2)} & y_{49}^{(2)} \\ \dots & \dots & \dots & \dots & \dots \end{array}$$

   (Notation: the superscripts $(1), (2), \dots$ are path indices, not derivatives.)

(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.)

(b) Implement an algorithm to create a path for the unicycle as follows:

[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. Exactly what time scale you choose is up to you, so long as the unicycle's motion starts at time $t = 0$.]

- 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`), subject to some constraints described next.
- The three paths should all lie on the same side of the ring of fire, all passing either to the left or all to the right of the ring.
- 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.
- You should choose these weights so that $p(0)$ lies within the triangle formed by the starting locations of the three paths that your algorithm picks, that is, within the triangle formed by the points $p^{(i)}(0)$, $p^{(j)}(0)$, and $p^{(k)}(0)$.
- Your algorithm should be able to produce a value $p(t)$ for all relevant (continuous) times t prior to reaching the Destination, not just for the discrete time snapshots given in `paths.txt`. (It is here that interpolation comes into play. Use whatever interpolation method you find useful. Something simple is fine.)
- At all times, $p(t)$ should lie outside the ring of fire.

Comment: The unicycle's starting position should fall within the triangle formed by three paths' starting points, but there may be many valid such triples of paths. (You may assume there is at least one.) You should develop your own criteria for choosing one such triple of paths, $p^{(i)}$, $p^{(j)}$, $p^{(k)}$.

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.

(c) Discuss any decisions you made in your implementation, for example: How did you pick the triple of paths $p^{(i)}$, $p^{(j)}$, $p^{(k)}$? How did you choose the weights $\alpha_i, \alpha_j, \alpha_k$? How did you decide on a time scale for t? How did you decide on an interpolation method?

(d) 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. The paths should not touch the ring of fire.

(e) Discuss how your algorithm would need to be modified (if at all) if more obstacles like the ring of fire were to be introduced.

HW2

ruiyangz

Question1

a) a divided difference approach test

```
x = [0 1 2];
y = [1 3 2];
a = divided_diff_coeffs(x, y);
p = newton_eval(x, a, 1.5)
```

```
p =
2.8750
```

b) cos nterpolate

```
x = [0, 1/8, 1/4, 3/8, 1/2]
```

```
x = 1×5
         0    0.1250    0.2500    0.3750    0.5000
```

```
y = cos(pi * x)
```

```
y = 1×5
    1.0000    0.9239    0.7071    0.3827    0.0000
```

```
a = divided_diff_coeffs(x, y);

xstar = 3/10;

p_interp = newton_eval(x, a, xstar);

f_true = cos(pi * xstar);

% Error
err = abs(f_true - p_interp);

% Display results
fprintf('Interpolation of cos(pi*x) at x = %.2f\n', xstar);
```

```
Interpolation of cos(pi*x) at x = 0.30
```

```
fprintf('Interpolated value p(x) = %.8f\n', p_interp);
```

```
Interpolated value p(x) = 0.58785675
```

```
fprintf('True value f(x) = %.8f\n', f_true);
```

```
True value f(x) = 0.58778525
```

```
fprintf('Absolute error = %.3e\n', err);
```

```
Absolute error = 7.150e-05
```

c) function 2 / (1+9x^2)

```
f  = @(x) 2 ./ (1 + 9*x.^2);

x0 = 0.07;                      % evaluation point
ns = [2, 4, 40];                % degrees n

approx  = zeros(numel(ns),1);   % p_n(x0)
trueval = f(x0) * ones(numel(ns),1);
abserr  = zeros(numel(ns),1);

for t = 1:numel(ns)
    n = ns(t);
    xi = linspace(-1, 1, n+1);          % xi = 2*i/n - 1
    yi = f(xi);
    a  = divided_diff_coeffs(xi, yi);   % Newton DD coefficients
    approx(t) = newton_eval(xi, a, x0); % p_n(x0)
    abserr(t) = abs(approx(t) - trueval(t));
end

% Print a neat table
fprintf('\nPart (c): Interpolation of f(x)=2/(1+9x^2) at x0=%.4f\n', x0);
```

```
Part (c): Interpolation of f(x)=2/(1+9x^2) at x0=0.0700
```

```
fprintf('   n        p_n(x0)          f(x0)          |error|\n');
```

```
   n        p_n(x0)          f(x0)          |error|
```

```
for t = 1:numel(ns)
    fprintf('%4d   %14.10f  %14.10f   %10.3e\n', ns(t), approx(t), trueval(t),
abserr(t));
end
```

```
   2     1.9911800000    1.9155253328    7.565e-02
   4     1.9668750652    1.9155253328    5.135e-02
  40     1.9155253271    1.9155253328    5.687e-09
```

d) Numerical max interpolation error on [-1,1]

```
f  = @(x) 2 ./ (1 + 9*x.^2);
ns = [2 4 6 8 10 12 14 16 18 20 40];

xx = linspace(-1, 1, 20001);         % ~2e4 points
fx = f(xx);

En = zeros(numel(ns),1);

for t = 1:numel(ns)
```

```
    n  = ns(t);
    xi = linspace(-1, 1, n+1);       % equispaced nodes (same as part (c))
    yi = f(xi);
    a  = divided_diff_coeffs(xi, yi);

    % Evaluate interpolant on the fine grid
    % (arrayfun calls newton_eval at each xx)
    pxx = arrayfun(@(z) newton_eval(xi, a, z), xx);

    % Max absolute error on the grid
    En(t) = max(abs(fx - pxx));
end

% Print a neat table
fprintf('\nPart (d): Max interpolation error En on [-1,1]\n');
```

```
Part (d): Max interpolation error En on [-1,1]
```

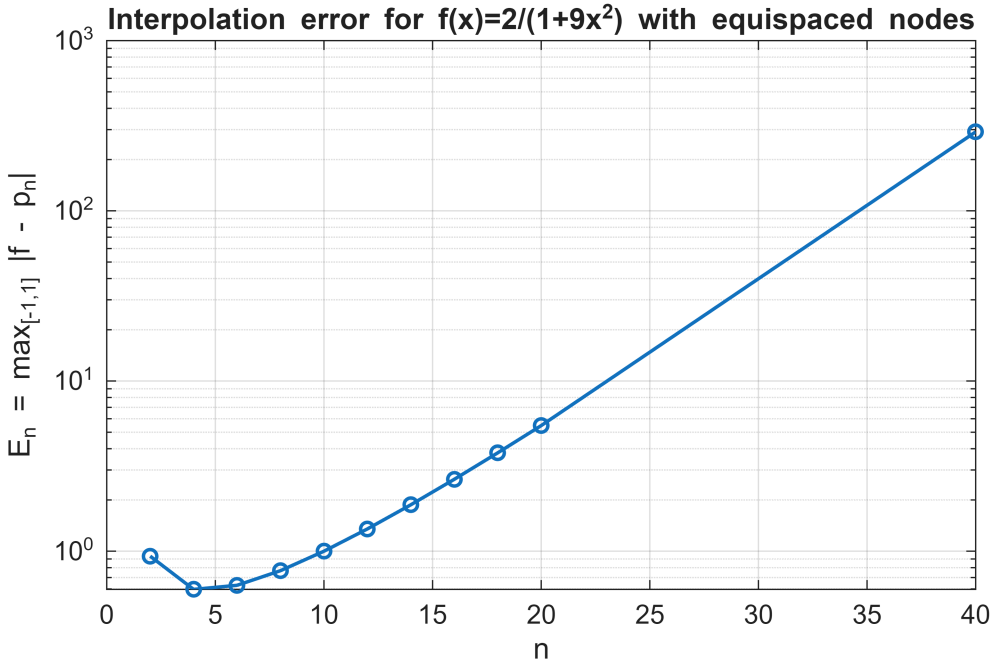
```
fprintf('   n            En (max |f - p_n|)\n');
```

```
   n           En (max |f - p_n|)
```

```
for t = 1:numel(ns)
    fprintf('%4d      %14.6e\n', ns(t), En(t));
end
```

```
   2       9.350889e-01
   4       5.963194e-01
   6       6.302535e-01
   8       7.681906e-01
  10       9.996674e-01
  12       1.351557e+00
  14       1.873958e+00
  16       2.645197e+00
  18       3.784085e+00
  20       5.469649e+00
  40       2.908908e+02
```

```
% Convergence plot
figure; semilogy(ns, En, '-o', 'LineWidth', 1.3, 'MarkerSize', 5);
grid on;
xlabel('n');
ylabel('E_n = max_{[-1,1]} |f - p_n|');
title('Interpolation error for f(x)=2/(1+9x^2) with equispaced nodes');
```



Question 2

Written

Question 3

```
f  = @(x) tan(x) - x;
fp = @(x) tan(x).^2;

x = linspace(13, 17.5, 1000);
y = f(x);

figure;
plot(x, y, 'b-', 'LineWidth', 1.5); hold on;
yline(0, 'k--');
xline(15, 'r--');
ylim([-20 20]);
xlabel('x');
ylabel('f(x) = tan(x) - x');
title('f(x) = tan(x) - x with roots near x=15');
```

2. Suppose you wish to build a "sliding-window interpolation table" (as discussed in class) 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[1], assuming that you will use linear interpolation between adjacent points in the table? How fine must the table spacing be if you will use quadratic interpolation? In each case, how many entries do you need in the table?

[1]Terminology: n decimal digit accuracy means the error is less than $\frac{1}{2} \cdot 10^{-n}$, so in this case less than $5 \cdot 10^{-7}$.

$f(x) = \sin x \quad [0, 2\pi]$

a). linear interpolation.

$$f(x) - p_1(x) = \frac{f''(\xi)}{2}(x-a)(x-b) \qquad \xi \in (a, b)$$

$h = b - a \quad x = a + th \quad \text{s.t.} \quad t \in [0, 1]$.

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

$$|(x-a)(x-b)| = |(a+th-a)(a+th-b)|$$

$$= |th(th-h)|$$

$$= |th \cdot h(t-1)|$$

$$= t \cdot h^2 (t-1)$$

since. $t \in [0, 1]$ $\max t(t-1) = 1/4$.

$$\Rightarrow h^2/4$$

$$|f(x) - p(x)| \le \frac{1}{2} \cdot \frac{h^2}{4} = h^2/8$$

for $h^2/8 \le 5 \times 10^{-7}$, $h \le 0.002$ $\quad n = \frac{2\pi}{h} + 1$

$$= 3141.59 + 1$$

$$\approx 3143.$$

b). x_{i-1}, x_i, x_{i+1}

$x_{i-1} = x_i - h$.

$x_{i+1} = x_i + h$

let. $x = x_i + \theta h \quad (\theta \in [-1, 1])$

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

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

$$|(x - x_{i-1})(x - x_i)(x - x_{i+1})| = |(x_i + \theta h - x_{i+h})(x_i + \theta h - x_i)(x_i + \theta h - x_i - h)|$$

$$= (\theta h + h)\,\theta h\,(\theta h - h)$$

$$= h(\theta + 1) \cdot \theta h \cdot h(\theta - 1)$$

$$= h^3 \cdot \theta \cdot (\theta + 1)(\theta - 1)$$

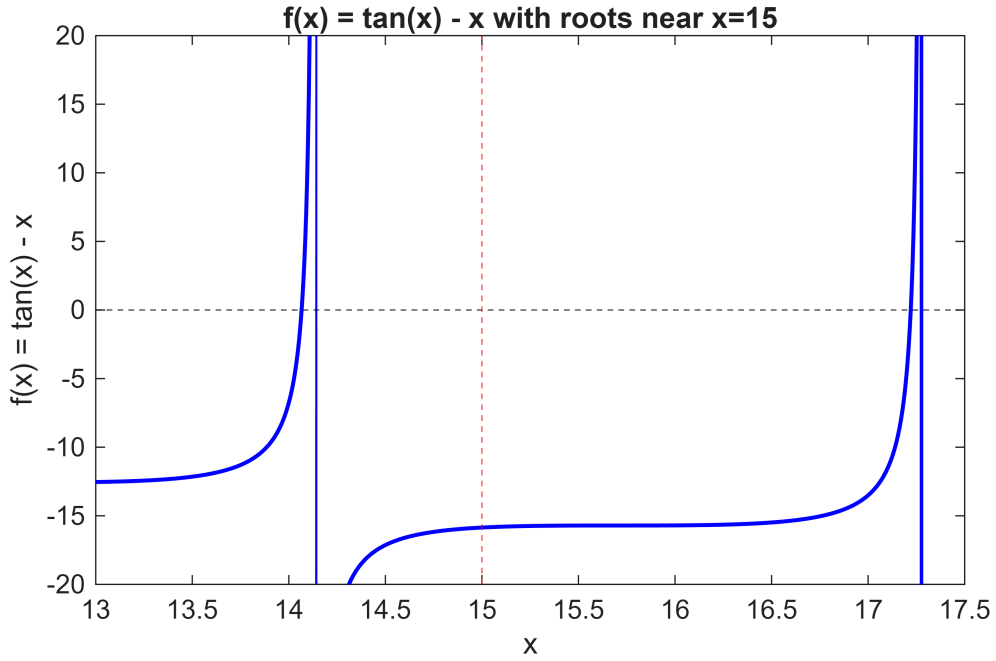
$$\max |\theta(\theta - 1)(\theta + 1)| \in [-1, 1] = \frac{2}{3\sqrt{3}} \quad \text{at } \theta = \pm \frac{1}{\sqrt{3}}.$$

$$= \frac{2}{3\sqrt{3}} h^3.$$

$$e_2(x) \le \frac{1}{6} \times \frac{2}{3\sqrt{3}} h^3 = \frac{1}{9\sqrt{3}} h^3 \le 5 \times 10^{-7}$$

$$h \le 0.01983$$

$$n = \frac{2\pi}{h} + 1 = 317.85 \approx 318.$$




```
x_low  = newton(f, fp, 14);
x_high = newton(f, fp, 17.25);

fprintf('x_low  = %.15f\n', x_low);
```

```
x_low  = 14.066193912831473
```

```
fprintf('x_high = %.15f\n', x_high);
```

```
x_high = 17.220755271930770
```

```
function root = newton(f, fp, x0, tol, max_iter)
    if nargin < 4, tol = 1e-12; end
    if nargin < 5, max_iter = 50; end

    x = x0;
    for k = 1:max_iter
        x_new = x - f(x)/fp(x);
        if abs(x_new - x) < tol
            root = x_new;
            return;
        end
        x = x_new;
    end
    error('Newton did not converge');
end
```

Question 4

4. Suppose ξ is a root of order 2 of $f(x)$, meaning $f(\xi) = 0$, $f'(\xi) = 0$ and $f''(\xi) \neq 0$.

   (a) Show that in this case Newton's method no longer converges quadratically. Do so by showing that the method now converges linearly.

   (b) We can modify Newton's method in this case: Show that the iteration

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

   does converge quadratically (or faster) when ξ is a root of order 2 of $f(x)$.

Throughout this problem you may assume that $f(x)$ has as many continuous derivatives as you need, for instance that $f'''(x)$ is continuous in a neighborhood of ξ.

Recommendation: Use a Taylor series for $h(x) = f(x)/f'(x)$, similar to what we did in lecture when examining convergence of Newton's method. You do not need anything more complicated. However, you will find it useful to remember L'Hôpital's rule from calculus when computing $h(\xi)$ and $h'(\xi)$. Do not worry about the exact value of $h''(\xi)$.

$f(\xi) = 0 \quad f'(\xi) = 0 \quad f''(\xi) \neq 0.$

For newton methods:

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

$f(x) = (x-\xi)^2 h(x) \qquad h(\xi) \neq 0 \Rightarrow f''(x) \neq 0.$

$f'(x) = 2(x-\xi)h(x) + (x-\xi)^2 h'(x).$

$\varepsilon_{n+1} = \xi - x_{n+1}$

$\varepsilon_n = \xi - x_n$

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

$$= \frac{f(x_n)}{f'(x_n)}$$

$$= \frac{f(\xi - \varepsilon_n)}{f'(\xi - \varepsilon_n)}$$

Taylor Expension:

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

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

$$\Rightarrow \varepsilon_{n+1} - \varepsilon_n = \frac{f(\xi-\varepsilon_n)}{f'(\xi-\varepsilon_n)}$$

$$= \frac{f(\xi) - \varepsilon_n f'(\xi) + \frac{\varepsilon_n^2}{2} f''(\xi) - \frac{\varepsilon_n^3}{6} f'''(\xi) + \cdots}{f'(\xi) - \varepsilon_n f''(\xi) + \frac{\varepsilon_n^2}{2} f'''(\xi) - \frac{\varepsilon_n^3}{6} f''''(\xi) + \cdots}$$

Since $f(\xi)$ & $f'(\xi) = 0$

$$= \frac{\frac{\varepsilon_n^2}{2} f''(\xi) - \frac{\varepsilon_n^3}{6} f'''(\xi) + \cdots}{-\varepsilon_n f''(\xi) + \frac{\varepsilon_n^2}{2} f'''(\xi) + \cdots}$$

$$= \frac{\frac{\varepsilon_n^2}{2} f''(\xi)\left(1 - \frac{\varepsilon_n}{3} \cdot \frac{f'''(\xi)}{f''(\xi)} + \cdots\right)}{-\varepsilon_n f''(\xi)\left(1 + \frac{\varepsilon_n}{2} \frac{f'''(\xi)}{f''(\xi)} + \cdots\right)}$$

$$= \varepsilon_n\left(-\frac{1}{2}\right).$$

Written

Question 5

a) implement muller method

```
function [x, xhist] = muller(f, x0, x1, x2, tol, maxit)

    if nargin < 6, maxit = 100; end
    if nargin < 5, tol   = 1e-12; end

    xhist = zeros(maxit+2,1);
    xhist(1:3) = [x0; x1; x2];

    for k = 1:maxit
        f0 = f(x0);  f1 = f(x1);  f2 = f(x2);

        h0 = x1 - x0;
        h1 = x2 - x1;

        d0 = (f1 - f0) / h0;
        d1 = (f2 - f1) / h1;

        a = (d1 - d0) / (h1 + h0);
        b = a*h1 + d1;
        c = f2;

        disc = sqrt(b.^2 - 4*a*c);
        if abs(b + disc) > abs(b - disc)
            denom = b + disc;
        else
            denom = b - disc;
        end

        if abs(denom) == 0
            x3 = x2;
        else
            x3 = x2 + (-2*c) / denom;
        end

        xhist(k+3) = x3;

        if abs(x3 - x2) <= tol*(1 + abs(x3)) || abs(f(x3)) <= tol
            xhist = xhist(1:k+3);
            x = x3;
            return;
        end

        x0 = x1;  x1 = x2;  x2 = x3;
    end
```

6. Consider the two univariate polynomials $p(x) = x^3 - 3x^2 + x - 3$, $q(x) = x^2 + x - 12$

(a) Using the method of resultants, decide whether $p(x)$ 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.

$p(x) = x^3 - 3x^2 + x - 3$

$q(x) = x^2 + x - 12$

$x^4 - 3x^3 + x^2 - 3x = 0$

$x^3 - 3x^2 + x - 3 = 0$

$x^4 + x^3 - 12x^2 = 0$

$x^3 + x^2 - 12x = 0$

$x^2 + x - 12 = 0$

$$\Rightarrow \begin{bmatrix} 1 & -3 & 1 & -3 & 0 \\ 0 & 1 & -3 & 1 & -3 \\ 1 & 1 & -12 & 0 & 0 \\ 0 & 1 & 1 & -12 & 0 \\ 0 & 0 & 1 & 1 & -12 \end{bmatrix} \begin{bmatrix} x^4 \\ x^3 \\ x^2 \\ x \\ 1 \end{bmatrix} = 0.$$

Q_2

Q_1

Q.

$\det Q = 0 \Rightarrow$ if that so. $p(x)$ & $q(x)$ share a common root.

b) Since p & q have common root.

we could setup. Q_1 & Q_2

$$Q_1 = \begin{bmatrix} -3 & 1 & -3 & 0 \\ 1 & -3 & 1 & -3 \\ 1 & -12 & 0 & 0 \\ 1 & 1 & -12 & 0 \end{bmatrix} \qquad Q_2 = \begin{bmatrix} 1 & 1 & -3 & 0 \\ 0 & -3 & 1 & -3 \\ 1 & -12 & 0 & 0 \\ 0 & 1 & -12 & 0 \end{bmatrix}$$

$$x = -\frac{\det(Q_1)}{\det(Q_2)} = 3$$

```
    x = x2;
    xhist = xhist(1:maxit+2);
end
```

b)

Since the function is a 3rd degree polynomial. I initially guess [-1, -0.5, 0] to approximate the real root.

```
f = @(x) x.^3 + x + 1;

tol   = 1e-12;
maxit = 100;

[r1, hist1] = muller(f, -1.0, -0.5, 0.0, tol, maxit);

p = [1 0 1 1];
[q,~] = deconv(p, [1, -r1]);

a = q(1);
b = q(2);
c = q(3);
r2 = (-b + sqrt(b^2 - 4*a*c)/(2*a));
r3 = (-b - sqrt(b^2 - 4*a*c)/(2*a));


fprintf('root1 ≈ %.5e%+.5ei (iters=%d)\n', real(r1), imag(r1), hist1);
```

```
root1 ≈ -6.82328e-01+0.00000e+00i (iters=-1)
root1 ≈ -5.00000e-01+0.00000e+00i (iters=-6.666667e-01)
root1 ≈ -6.83190e-01-6.82324e-01i (iters=-6.823278e-01)
root1 ≈ -6.82328e-01
```

```
fprintf('root2 ≈ %.5e%+.5ei ', real(r2), imag(r2));
```

```
root2 ≈ 6.82328e-01+1.16154e+00i
```

```
fprintf('root3 ≈ %.5e%+.5ei ', real(r3), imag(r3));
```

```
root3 ≈ 6.82328e-01-1.16154e+00i
```

Question 6

Written

Question 7

```
syms x y real
p = 2*x^2 + 2*y^2 - 4*x - 4*y + 3;
q = x^2 + y^2 + 2*x*y - 5*x - 3*y + 4;

%% (a) Plot zero-contours p=0 and q=0
figure('Color','w'); hold on; grid on;
fimplicit(p==0, [-1 3 -1 3], 'LineWidth',1.8);
fimplicit(q==0, [-1 3 -1 3], 'LineWidth',1.8);
```

7. Consider the two bivariate polynomials

$$p(x,y) = 2x^2 + 2y^2 - 4x - 4y + 3$$
$$q(x,y) = x^2 + y^2 + 2xy - 5x - 3y + 4$$

(a) Draw the zero contour $p(x,y) = 0$ and the zero contour $q(x,y) = 0$, with x and y real.

(b) Using the method of resultants, solve for the intersection points of these contours, i.e., find all (x,y) for which $p(x,y) = 0 = q(x,y)$, with x and y real. Do so by treating the polynomials p and q as functions of y, temporarily viewing x as a constant. Obtain a 4×4 matrix whose determinant, now a function of x, is the desired resultant. Find the x-roots of that determinant, thereby projecting the intersection points onto the x-axis. Use those x-roots to find the y-coordinates of the intersection points. Verify your intersection points.

b) $p(x,y) = 2x^2 + 2y^2 - 4x - 4y + 3 \Rightarrow 2y^2 - 4y + (2x^2 - 4x + 3)$

$q(x,y) = x^2 + y^2 + 2xy - 5x - 3y + 4. \Rightarrow y^2 + (2x-3)y + (x^2 - 5x + 4)$.

$$S(x) = \begin{bmatrix} 2 & -4 & 2x^2-4x+3 & 0 \\ 0 & 2 & -4 & 2x^2-4x+3 \\ 1 & 2x-3 & x^2-5x+4 & 0 \\ 0 & 1 & 2x-3 & x^2-5x+4 \end{bmatrix}$$

$R(x) = \det(S(x)) = 16x^4 - 48x^3 + 48x^2 - 28x + 11$

$\Rightarrow x_1 = 1.7021 \quad x_2 = 0.9016$

$$\begin{cases} 2y^2 - 4y + (2x^2 - 4x + 3) = 0 & ① \\ y^2 + (2x-3)y + (x^2 - 5x + 4) = 0 & ② \end{cases}$$

① − 2× ②.

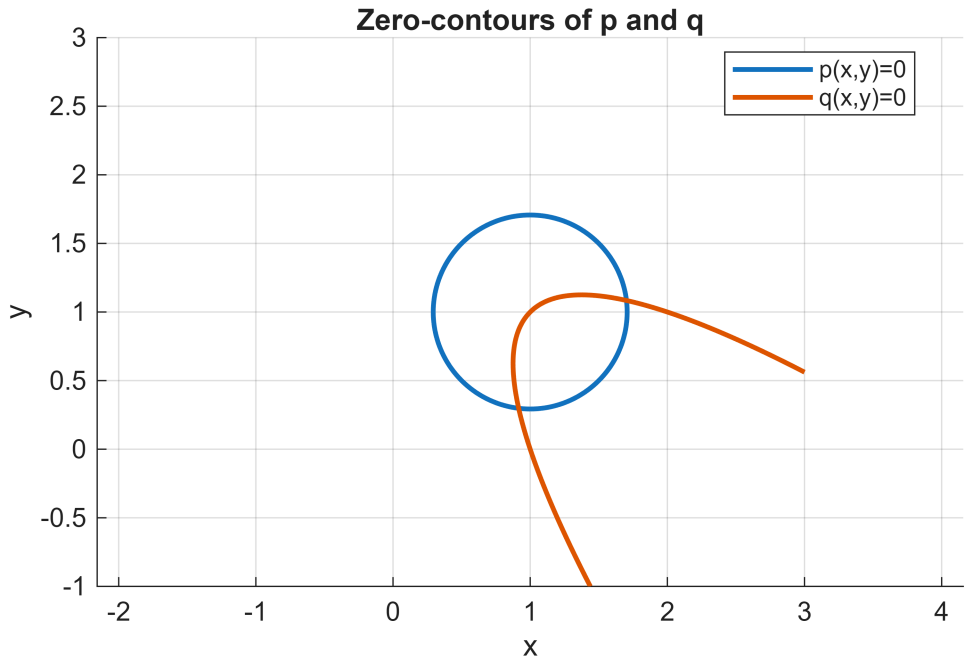
$(-4 - 4x + 6)y + 2x^2 - 4x + 3 - 2x^2 + 10x - 8 = 0$

$(-4x + 2)y + 6x - 5 = 0$

$$y = \frac{5 - 6x}{2 - 4x}$$

for $x = 1.7021$, $y = 1.08$; for $x = 0.9016$ $y = 0.298$.

```
axis equal; xlabel('x'); ylabel('y');
legend('p(x,y)=0','q(x,y)=0','Location','best');
title('Zero-contours of p and q');
```



b) Written

Question 8

a) Written

b)

See q8_partb_.asv

c)

Choosing the paths ((p^{(i)}, p^{(j)}, p^{(k)}):

I divided the path into above and below the fire's center. I also selected the side whose starting points that were closest to the provided start points. Within that side, I looked for triples of paths whose initial points formed a non-degenerate triangle containing the start point.

Choosing the weights αi,αj,αk:

The weights were determined using barycentric coordinates of the start point with respect to the chosen triangle. This guarantees that the weights are nonnegative, sum to one, and naturally represent the influence of each path.

Deciding on a time scale for t:

8. a)

set. $P_i = (x_i, y_i)$ $P_j = (x_j, y_j)$ $P_k = (x_k, y_k)$

For identify the new P is in the tricngle or not.

we set a function that

$$\begin{cases} X = V_1 x_i + V_2 x_j + V_3 x_k \\ Y = V_1 y_i + V_2 y_j + V_3 y_k \\ V_1 + V_2 + V_3 = 1 \end{cases}$$

$$\Rightarrow \begin{bmatrix} x_i & x_j & x_k \\ y_i & y_j & y_k \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \\ V_3 \end{bmatrix} = \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}$$

That if $P(x, y)$ is in the tricngle

$V_1, V_2, V_3 \geq 0$.

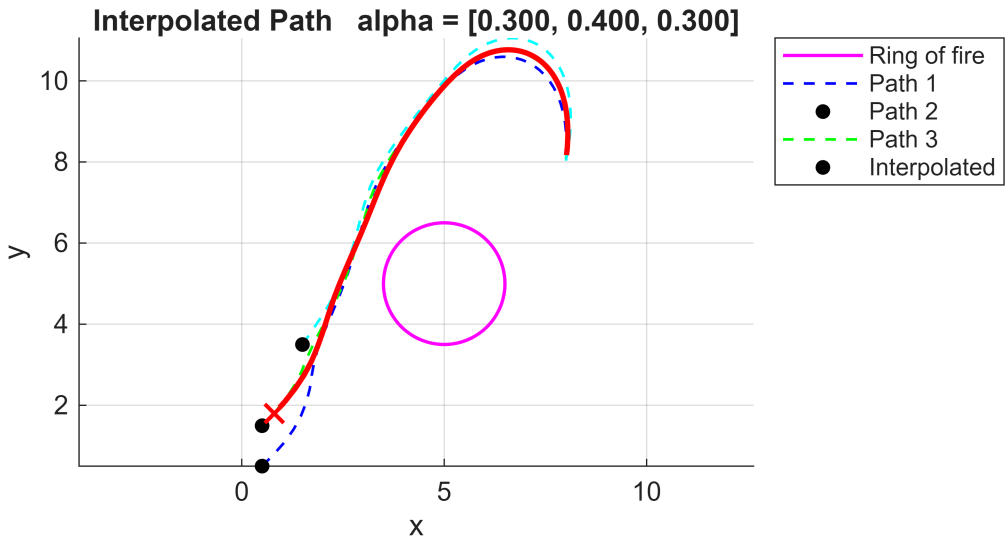
I normalized the interpolation parameter to the range $t \in [0,1]$. A step size of 0.01 was used, giving a good balance between smoothness of the trajectory and computational efficiency.

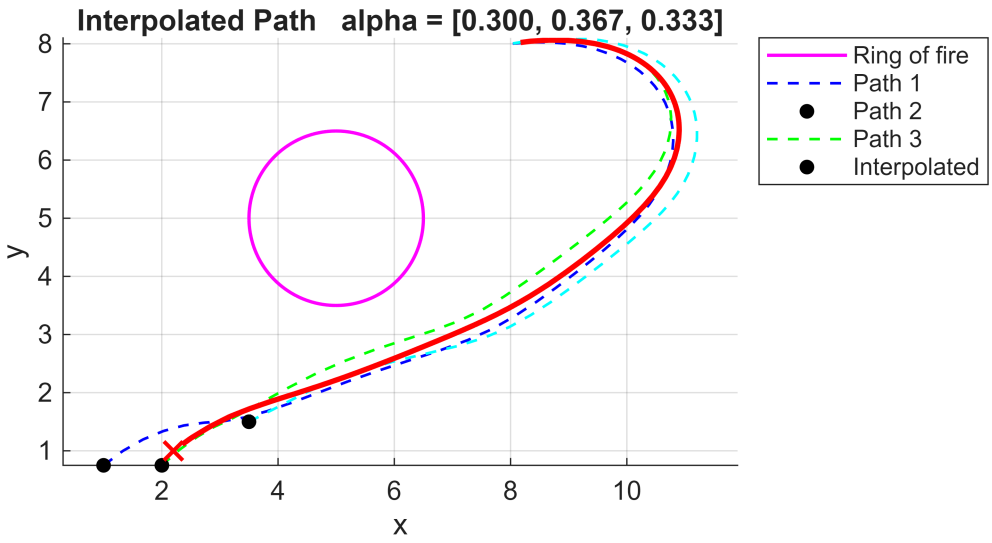
Interpolation method:

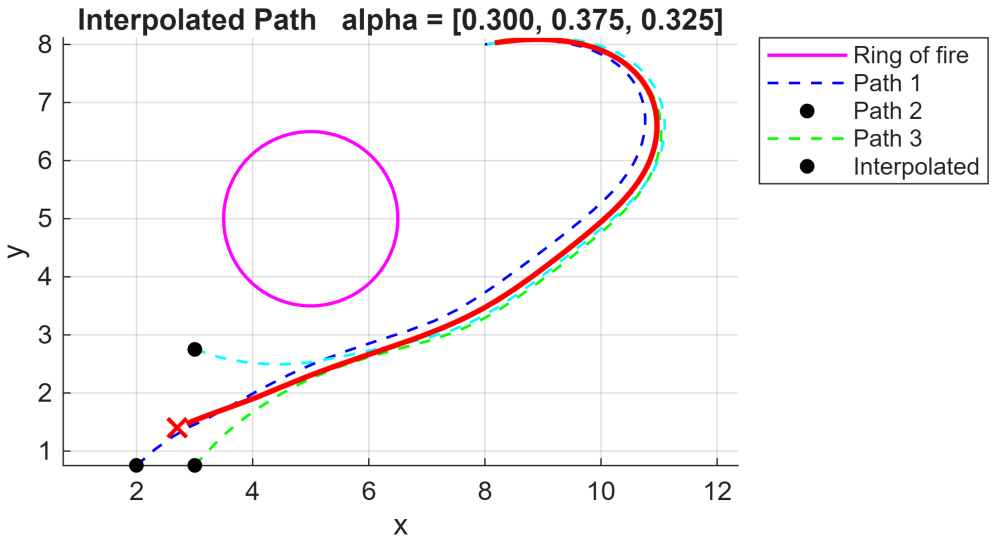
I applied piecewise-linear interpolation along the blended path. This method is simple, numerically stable, and ensures that the interpolated trajectory always remains inside the convex hull of the chosen reference paths.

d)

```
start_points = [0.8, 1.8;
                    2.2, 1.0;
                    2.7, 1.4];
for i = 1:size(start_points,1)
    q8_part_b(start_points(i,:));
end
```







e)

If more obstacles are added, the main change is to extend the collision check so that the blended path is validated against all obstacles rather than just one. We could let the time step be smaller for safety improvement.

```
function q8_part_b(start_point)

    paths = load('paths.txt');
    assert(mod(size(paths,1),2)==0, 'paths must have alternating x/y rows.');

    fire_center = [5, 5];
    fire_radius = 1.5;

    %start_point = [0.8, 1.8];
    %start_point = [2.2, 1.0];
    % start_point = [2.7, 1.4];

    [up_x, up_y, down_x, down_y] = split_paths_up_down(paths, fire_center);

    [path_set_x, path_set_y] = pick_path_triple(up_x, up_y, down_x, down_y, start_point);

    p1 = [path_set_x(1,1), path_set_y(1,1)];
    p2 = [path_set_x(2,1), path_set_y(2,1)];
    p3 = [path_set_x(3,1), path_set_y(3,1)];
    alpha = get_alpha(start_point, p1, p2, p3);

    [new_path_x, new_path_y] = blend_paths(alpha, path_set_x, path_set_y);

    t = 0.01:0.01:0.99;
    interp_x = polyline_interp(new_path_x, t);
    interp_y = polyline_interp(new_path_y, t);

    d2 = (interp_x - fire_center(1)).^2 + (interp_y - fire_center(2)).^2;
    if any(d2 < fire_radius^2)
        warning('Interpolated path touches the ring of fire.');
    end

    figure('Name','Q8(b) Interpolation','Color','w'); hold on; axis equal; grid on;
    title(sprintf('Interpolated Path   alpha = [%.3f, %.3f, %.3f]', alpha(1), alpha(2),
alpha(3)));
    xlabel('x'); ylabel('y');

    % ring of fire
    th = linspace(0, 2*pi, 360);
    plot(fire_center(1) + fire_radius*cos(th), fire_center(2) + fire_radius*sin(th), 'm-',
'LineWidth', 1.25);

    cols = {'b--','g--','c--'};
    for i = 1:3
        plot(path_set_x(i,:), path_set_y(i,:), cols{i}, 'LineWidth', 1.0);
        plot(path_set_x(i,1), path_set_y(i,1), 'ko', 'MarkerSize', 5, 'MarkerFaceColor', 'k');
    end

    plot(interp_x, interp_y, 'r-', 'LineWidth', 2.0);
    plot(start_point(1), start_point(2), 'rx', 'MarkerSize', 10, 'LineWidth', 1.5);

    legend({'Ring of fire','Path 1','Path 2','Path 3','Interpolated'}, 'Location','bestoutside');
end


function [up_x, up_y, down_x, down_y] = split_paths_up_down(paths, center)

    [m, n] = size(paths);
    X = paths(1:2:m, :);
    Y = paths(2:2:m, :);

```

```
    mid = max(1, round(n/3));

    up_rows   = find(Y(:, mid) > center(2));
    down_rows = find(Y(:, mid) <= center(2));

    up_x   = X(up_rows, :);
    up_y   = Y(up_rows, :);
    down_x = X(down_rows, :);
    down_y = Y(down_rows, :);

    % figure('Name','Up/Down Split','Color','w');
    % subplot(1,2,1); hold on; axis equal; title('LEFT of ring'); grid on;
    % for i = 1:size(up_x,1), plot(up_x(i,:), up_y(i,:)); end
    % subplot(1,2,2); hold on; axis equal; title('RIGHT of ring'); grid on;
    % for i = 1:size(down_x,1), plot(down_x(i,:), down_y(i,:)); end
end



function [path_set_x, path_set_y] = pick_path_triple(up_x, up_y, down_x, down_y, sp)

    side_center_left  = [mean(up_x(:,1)),  mean(up_y(:,1))];
    side_center_right = [mean(down_x(:,1)), mean(down_y(:,1))];

    if isempty(up_x)
        use_up = false;
    elseif isempty(down_x)
        use_up = true;
    else
        use_up = sum((sp - side_center_left).^2) <= sum((sp - side_center_right).^2);
    end

    if use_up
        px = up_x;  py = up_y;
    else
        px = down_x; py = down_y;
    end

    m = size(px, 1);
    if m < 3
        error('Not enough paths on the chosen side.');
    end

    candidates = [];
    for i = 1:m-2
        for j = i+1:m-1
            for k = j+1:m
                p1 = [px(i,1), py(i,1)];
                p2 = [px(j,1), py(j,1)];
                p3 = [px(k,1), py(k,1)];
                if triangle_area(p1,p2,p3) <= 1e-10, continue; end
                if point_in_triangle(sp, p1, p2, p3, 1e-12)
                    candidates = [candidates; i, j, k];
                end
            end
        end
    end

    if isempty(candidates)
        error('No valid triangle contains the start point on the chosen side.');
    end
```

```
121
122     nC = size(candidates, 1);
123     dists = zeros(nC,1);
124     for c = 1:nC
125         i = candidates(c,1); j = candidates(c,2); k = candidates(c,3);
126         p1 = [px(i,1), py(i,1)];
127         p2 = [px(j,1), py(j,1)];
128         p3 = [px(k,1), py(k,1)];
129         center = (p1 + p2 + p3) / 3;
130         dists(c) = sum((center - sp).^2);
131     end
132     [~, ind] = min(dists);
133
134     path_set_x = [px(candidates(ind,1), :);
135                   px(candidates(ind,2), :);
136                   px(candidates(ind,3), :)];
137     path_set_y = [py(candidates(ind,1), :);
138                   py(candidates(ind,2), :);
139                   py(candidates(ind,3), :)];
140 end
141
142
143 function [inside, alpha] = point_in_triangle(p, p1, p2, p3, tol)
144
145     if nargin < 5, tol = 1e-12; end
146
147     A = [p1(1) p2(1) p3(1);
148          p1(2) p2(2) p3(2);
149          1      1      1   ];
150     b = [p(1); p(2); 1];
151
152     detA = det(A);
153     if abs(detA) < tol
154         inside = false;
155         alpha  = [NaN; NaN; NaN];
156         return;
157     end
158
159     alpha = A \ b;
160
161     nonneg = all(alpha >= -tol);
162     sum1   = abs(sum(alpha) - 1) <= 1e-9;
163
164     inside = nonneg && sum1;
165
166     if inside
167         alpha(alpha < 0 & alpha > -tol) = 0;
168         s = sum(alpha);
169         if abs(s - 1) > 1e-12
170             alpha = alpha / s;
171         end
172     end
173 end
174
175 function alpha = get_alpha(sp, p1, p2, p3)
176
177     M = [ (p1 - p3); (p2 - p3) ]';
178     b = (sp - p3)';
179     if abs(det(M)) < 1e-14
180         error('get_alpha: degenerate triangle.');
181     end
```

```
    ab = M \ b;
    alpha = [ab(1); ab(2); 1 - ab(1) - ab(2)];
end


function [nx, ny] = blend_paths(alpha, path_set_x, path_set_y)
    nx = alpha' * path_set_x;
    ny = alpha' * path_set_y;
end


function y = polyline_interp(data, t)

    N = numel(data);
    s = (N - 1) * t;
    k = floor(s) + 1;
    k(k >= N) = N - 1;
    frac = s - floor(s);
    y = data(k) + (data(k+1) - data(k)) .* frac;
end


function A = triangle_area(p, q, r)
    A = 0.5 * abs( (q(1)-p(1))*(r(2)-p(2)) - (q(2)-p(2))*(r(1)-p(1)) );
end

```