Assignment 2 Solutions

Robotics 811, Fall 2015

1. Prove that the first derivative $p_2'(x)$ of the parabola interpolating $f(x)$ at $x_0 < x_1 < x_2$ is equal to the straight line which takes on the value $f[x_{i-1}, x_i]$ at the point $(x_{i-1} + x_i)/2$, for $i = 1, 2$.

   We use the divided difference method to interpolate the parabola:

$$p_2(x) = A_0 + A_1(x - x_0) + A_2(x - x_0)(x - x_1)$$

   and differentiate

$$p_2'(x) = A_1 + 2A_2 x - A_2(x_0 + x_1)$$

   The derivative is linear in x and is uniquely defined by two points.

   $i = 1$:

$$\begin{aligned} p_2'(\tfrac{x_0+x_1}{2}) &= f[x_0, x_1] + 2f[x_0, x_1, x_2]\tfrac{(x_0+x_1)}{2} - f[x_0, x_1, x_2](x_0 + x_1) \\ p_2'(\tfrac{x_0+x_1}{2}) &= f[x_0, x_1] \end{aligned}$$

   $i = 2$:

$$\begin{aligned} p_2'(\tfrac{x_1+x_2}{2}) &= f[x_0, x_1] + 2f[x_0, x_1, x_2]\tfrac{(x_1+x_2)}{2} - f[x_0, x_1, x_2](x_0 + x_1) \\ p_2'(\tfrac{x_1+x_2}{2}) &= f[x_0, x_1] - f[x_0, x_1, x2](x_0 - x_2) \\ p_2'(\tfrac{x_1+x_2}{2}) &= f[x_0, x_1] - \tfrac{f[x_0,x_1]-f[x_1,x_2]}{x_0-x_2}(x_0 - x_2) \\ p_2'(\tfrac{x_1+x_2}{2}) &= f[x_0, x_1] \end{aligned}$$

2. In your favorite programming language implement a procedure that interpolates $f(x)$ based on a divided difference approach.

   The procedure should take as input the following $2n + 4$ numbers:

$$n, x, x_0, \ldots, x_n, f(x_0), \ldots, f(x_n).$$

   The procedure should compute the interpolated value $f(x)$.

   Here is an implementation in Python:

```
def interp_div_dif(input):
  n = input[0]
  interp_x = input[1]
  x = []
  y = []
  # pull out x,ys from input
  for i in range(n):
    x += [input[i+2]]
    y += [input[n+i+2]]
  d = [y]
  a = [x[0]]
  # construct the matrix
  for j in range(n-1,0,-1):
    t = []
    for i in range(j):
      t += [(d[n-1-j][i] - d[n-1-j][i+1]) / (x[i] - x[i+n-j])];
      d += [t]
      a += [t[0]]
  m = 1
  ans = y[0]
  # evaluate the polynomial
  for i in range(1,len(a)):
    m *= interp_x - x[i-1]
    ans += a[i] * m
  return ans
```

(a) Use your procedure to interpolate $\log_7^2 x$ at $x = 2.25$, based on known values of $\log_7^2 x$ at $x = 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0$.

The interpolation of $\log_7^2 2.25$ gives **0.1737**.

(b) Now consider the function

$$f(x) = \frac{5}{1 + 36\,x^2},$$

with input data given at the points

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

Use your procedure to estimate $f(x)$ at $x = 0.05$, with $n = 2$. **4.9878**.

Use your procedure to estimate $f(x)$ at $x = 0.05$, with $n = 4$. **4.9442**.

Use your procedure to estimate $f(x)$ at $x = 0.05$, with $n = 40$. **4.5872**.

What is the actual value of $f(0.05)$? **4.5872**. This is the exact value of the interpolation with $n = 40$, which is not surprising since $x_{21} = 0.05$.

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

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

(do so numerically by discretizing the interval $[-1, 1]$ finely).

Estimate E_n for $n = 2, 4, 6, 8, 10, 12, 14, 16, 18, 20$, and 40, for the function $f(x) = 5/(1 + 36\,x^2)$ given above.

Do the error estimates make sense? Explain your results.

We observe that the error at the endpoints increases almost exponentially as we increase n. This does make sense. We know that the maximum error is given by the second theorem on page 16 of the Polynomial Approximations Interpolation lecture notes.

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

Let

$$A = \frac{f^{(n+1)}(\xi)}{(n+1)!} \text{ and } B = \prod_{i=0}^{n}(\bar{x} - x_i)$$

For our f, as n increases, A increases. For the error to decrease, the decrease in B must be larger than the increase in A. If you plot $\bar{x}$ versus A, you will see that the largest values are near the extremities of the interval. This is a problem with interpolating with uniformly spaced points. This is also the intuition for choosing Chebyshev points which have more sampling density near the ends of the intervals.

However, with uniform spacing, A does not decrease as fast as B increases and hence the error increases with it. This is known as *Runge's phenomenon*.

n	$\|e_n(\bar{x})\|$
2	3,4911
4	2,3576
6	3,5341
8	6,4869
10	12,9690
12	26,7578
14	58,3866
16	124,8190
18	268,6232
20	625,7294
40	2,105,800

3. Suppose you wish to build an interpolation table with entries of the form $(x, f(x))$ for the function $f(x) = \cos x$ over the interval $[-\frac{\pi}{2}, \frac{3\pi}{2}]$. 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?

We only need table entries for $[-\pi/2, 0]$ because we can do sign and phase shifts for the other quadrants. By the second theorem on page 16 of the Polynomial Approximations – Interpolation handout, the error using linear interpolation is

$$\begin{aligned} e_1(\bar{x}) &= \tfrac{f''(\xi)}{2}(\bar{x} - x_i)(\bar{x} - x_{i+1}) \\ &= \tfrac{-\cos\xi}{2}(\bar{x} - x_i)(\bar{x} - x_{i+1}) \end{aligned}$$

We don't know ξ, but the maximum $-\cos\xi/2 = 1/2$, and the maximum $(\bar{x}-x_i)(\bar{x}-x_i+1) \leq \max_{0\leq y\leq h} |(y-h)y|$ which we can quickly see (by taking the derivative and setting it to zero) is when $y = h/2$. So

$$\begin{aligned} e_1(\bar{x}) &\leq |\tfrac{1}{2}(\tfrac{h}{2})^2| \\ &\leq |\tfrac{h^2}{8}| \end{aligned}$$

For our desired accuracy, $\frac{h^2}{8} = 5e^{-7} \rightarrow h = 0.002$, and for the range from $[-pi/2, 0]$ we need **786** entries in the table. Or, if we forego the phase/sign trickery, we will need **3143** entries. Please note that you always need to round up if you want the desired accuracy.

Now for quadratic interpolation, we have

$$\begin{aligned} e_2(\bar{x}) &= \tfrac{f'''(\xi)}{6}(\bar{x} - x_{i-1})(\bar{x} - x_i)(\bar{x} - x_{i+1}) \\ &= \tfrac{\sin\xi}{2}(\bar{x} - x_{i-1})(\bar{x} - x_i)(\bar{x} - x_{i+1}) \end{aligned}$$

Once again, we can see that the maximum $\sin\xi/6 = 1/6$, and from page 18 of the handout, we know that

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

So that

$$\begin{aligned} e_2(\bar{x}) &\leq \tfrac{1}{6}\tfrac{2}{3}\tfrac{1}{\sqrt{3}}h^3 \\ &\leq \tfrac{h^3}{9\sqrt{3}} \end{aligned}$$

For our desired accuracy, $h = \sqrt[3]{9\sqrt{3}5e^{-7}} = 0.01983$, and for the range from $[-pi/2, 0]$ we only need **80** entries in the table. Or, if we forego the phase/sign trickery, we will need **318** entries.

4. Implement Newton's Method. Find the solution of the equation

$$x = \tan x$$

that is closest to 200, using Newton's method (and any techniques you need to start Newton in a region of convergence).

The easiest way to find a suitable interval is by graphing the function at high precision and looking for a nicely shaped area to start Newton's Method. Another way to kick-start Newton's method is to use bisection to get in the close neighborhood of the root.

Another way to solve the problem is to use the periodicity of the tan function and the Applicability Theorem (page 16 of the Solutions to Nonlinear Equations handout) to find good intervals. Using Newton's method from good starting points gives us the roots: **199.486121** (the root closest to 200) and 202.62779.

5. 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 Newton's method no longer converges quadratically.

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.

Expand both $f(x)$ and $f'(x)$ about the root ξ:

$$\begin{aligned} f(x+\epsilon) &= f(x) + \epsilon f'(x) + \frac{\epsilon^2}{2!}f''(x) + \frac{\epsilon^3}{3!}f'''(x) + ... && (1) \\ f'(x+\epsilon) &= f'(x) + \epsilon f''(x) + \frac{\epsilon^2}{2!}f'''(x) + ... && (2) \end{aligned}$$

Now, writing the error in the n^{th} step as $\epsilon_n = (\xi - x_n)$, we can rewrite the update rule for Newton's method as

$$\epsilon_{n+1} = \epsilon_n + \frac{f(\xi - \epsilon_n)}{f'(\xi - \epsilon_n)}$$

and subste in our expansions for $f(x)$ and $f'(x)$:

$$\begin{aligned}
\epsilon_{n+1} &= \epsilon_n + \frac{f(x) - \epsilon_n f'(x) + \frac{\epsilon_n^2}{2!}f''(x) - \frac{\epsilon_n^3}{3!}f'''(x) + ...}{f'(x) - \epsilon_n f''(x) + \frac{\epsilon_n^2}{2!}f'''(x) + ...} \\
\epsilon_{n+1} &= \epsilon_n + \frac{\frac{\epsilon_n^2}{2!}f''(x) - \frac{\epsilon_n^3}{3!}f'''(x) + ...}{-\epsilon_n f''(x) + ...} \\
\epsilon_{n+1} &\approx \epsilon_n - \frac{\epsilon_n}{2!} + \frac{\epsilon_n^2}{3!}\frac{f'''(x)}{f''(x)} + ... \\
\epsilon_{n+1} &\approx \frac{\epsilon_n}{2} + \frac{\epsilon_n^2}{3!}\frac{f'''(x)}{f''(x)} + ...
\end{aligned}$$

So Newton's method converges **linearly**.

Now, using the accelerated version of Newton's method, we have

$$\epsilon_{n+1} = \epsilon_n + 2\frac{f(\xi - \epsilon_n)}{f'(\xi - \epsilon_n)}$$

In this case the substitution leads to

$$\begin{aligned}
\epsilon_{n+1} &= \epsilon_n + 2\frac{f(x) - \epsilon_n f'(x) + \frac{\epsilon_n^2}{2!}f''(x) - \frac{\epsilon_n^3}{3!}f'''(x) + ...}{f'(x) - \epsilon_n f''(x) + \frac{\epsilon_n^2}{2!}f'''(x) + ...} \\
\epsilon_{n+1} &= \epsilon_n + 2\frac{\frac{\epsilon_n^2}{2!}f''(x) - \frac{\epsilon_n^3}{3!}f'''(x) + ...}{-\epsilon_n f''(x) + \frac{\epsilon_n^2}{2!}f'''(x)...} \\
\epsilon_{n+1} &= \frac{\epsilon_n(-\epsilon_n f''(x) + \frac{\epsilon_n^2}{2!}f'''(x)...) + 2(\frac{\epsilon_n^2}{2!}f''(x) - \frac{\epsilon_n^3}{3!}f'''(x) + ...)}{-\epsilon_n f''(x) + \frac{\epsilon_n^2}{2!}f'''(x)...} \\
\epsilon_{n+1} &\approx \frac{\frac{\epsilon_n^3}{2}f'''(x) + ... - \frac{\epsilon_n^3}{3}f''' + ...}{-\epsilon_n f'' + \frac{\epsilon_n^2}{2}f''' + ...} \\
\epsilon_{n+1} &\approx \frac{\frac{\epsilon_n^2}{6}f''' + ...}{f'' - \frac{\epsilon_n}{2}f''' + ...} \\
\epsilon_{n+1} &\approx \frac{1}{6}\frac{f'''}{f''}\epsilon_n^2
\end{aligned}$$

So the accelerated Newton's method converges **quadratically** for a double root.

Note: Alternately, we could solve this problem by expanding $h(x) = \frac{f(x)}{f'(x)}$ directly.

6. (a) Implement Müller's method.

   Here is an implementation in Python:

```
def root_muller(f, x0=1.1, x1=1.2, x2=1.3):
  A = ((f(x0) - f(x1))/(x0 - x1) - (f(x1) - f(x2)) / (x1 - x2))/(x0-x2)
  B = (f(x0) - f(x1))/(x0 - x1) + A * (x2 - x1)
  C = f(x2)
```

```
# flatten complex roots
denom_plus = B+sqrt(B**2 - 4 * A * C).real
denom_minus = B-sqrt(B**2 - 4 * A * C).real
if abs(denom_plus) > abs(denom_minus):
  denom = denom_plus
else:
  denom = denom_minus
if denom == 0.:
  print denominator is zero, bailing
  return
x3 = x2 - ( 2*C ) / denom
if x1 == x3 or x2 == x3: return x2
#polishing not required if using exact version
#if (fabs(x2 - x3) < 0.0000001): return x2
return root_muller(f, x1, x2, x3)
```

(b) Use Müller's method to find the first three strictly positive (real) solutions of the equation

$$x = \tan x.$$

The first three strictly positive roots are **4.4934**, **7.7253**, and **10.9041**. You need to be careful with where you start the bracket because otherwise you will converge not to the root but to infinity.

(c) Bessel's function of order 1 is defined by the series

$$J_1(z) = \frac{z}{2} \sum_{k=0}^{\infty} \frac{(-z^2/4)^k}{k!\,(k+1)!}.$$

Find the first four strictly positive (real) roots of this function using Müller's method. Start by truncating the series with $k = 3$, and then increase k until you are satisfied that you have the correct roots.

Convergence is a somewhat loose term, but if we assume convergence means that the root is stable to four decimal places, here are the first four roots of Bessel's function and the values of k for which they converged:

k	Root
8	3.8317
12	7.0156
17	10.1735
21	13.3237

7. Consider the two univariate polynomials

$$\begin{aligned} p(x) &= x^3 - 12x^2 + 41x - 42 \\ q(x) &= x^2 - 2x - 35 \end{aligned}$$

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

Using Sylvester's method:

$$Q = \begin{pmatrix} 1 & -12 & 41 & -42 & 0 \\ 0 & 1 & -12 & 41 & -42 \\ 1 & -2 & -35 & 0 & 0 \\ 0 & 1 & -2 & -35 & 0 \\ 0 & 0 & 1 & -2 & -35 \end{pmatrix}$$

$$det(Q) = 0$$

So yes, there is a common root.

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

As in the notes, consider the partial system

$$Q = \begin{pmatrix} 1 & -12 & 41 & -42 & 0 \\ 0 & 1 & -12 & 41 & -42 \\ 1 & -2 & -35 & 0 & 0 \\ 0 & 1 & -2 & -35 & 0 \end{pmatrix} \begin{pmatrix} x^4 \\ x^3 \\ x^2 \\ x \\ 1 \end{pmatrix} = 0$$

And now using the matrices found by omitting the first and then second columns

$$Q_1 = \begin{pmatrix} -12 & 41 & -42 & 0 \\ 1 & -12 & 41 & -42 \\ -2 & -35 & 0 & 0 \\ 1 & -2 & -35 & 0 \end{pmatrix}, \; Q_2 = \begin{pmatrix} 1 & 41 & -42 & 0 \\ 0 & -12 & 41 & -42 \\ 1 & -35 & 0 & 0 \\ 0 & -2 & -35 & 0 \end{pmatrix}$$

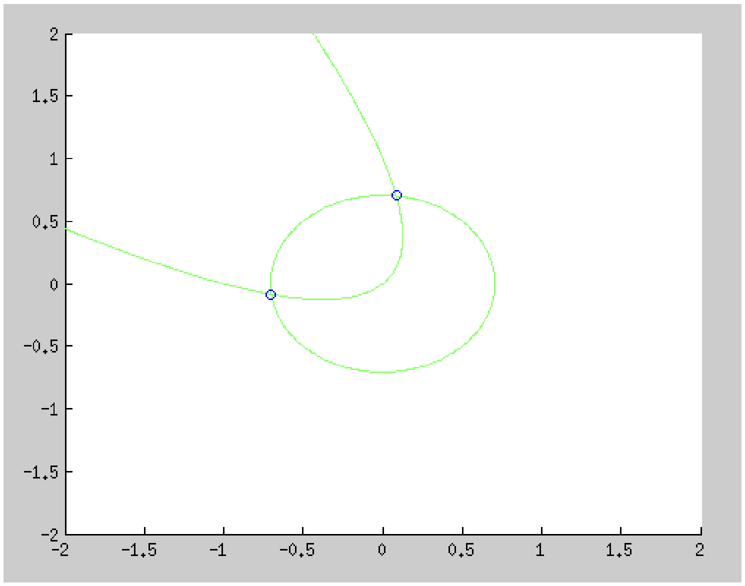
$$-1^{(1+2)} \frac{det(Q_1)}{det(Q_2)} = -1\frac{806736}{-115248} = 7$$

The root is at $x = 7$.

8. Consider the two bivariate polynomials

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

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



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

First, we pretend that x is constant and rewrite p and q as functions of y.

$$\begin{aligned} p(y) &= 2y^2 + 0y + (2x^2 - 1) \\ q(y) &= y^2 + (2x+1)y + (x^2 + x) \end{aligned}$$

We then use Sylvester's method to form the matrix Q.

$$Q = \begin{pmatrix} 2 & 0 & 2x^2 - 1 & 0 \\ 0 & 2 & 0 & 2x^2 - 1 \\ 1 & 2x - 1 & x^2 + x & 0 \\ 0 & 1 & 2x - 1 & x^2 + x \end{pmatrix}$$

So $p(y)$ and $q(y)$ will have a common zero when

$$det(Q) = 16x^4 - 16x^3 + 12x - 1 = 0$$

and solving for x, we obtain the real solutions $x = -0.7021, 0.0841$. We plug these values back into $p(y)$ and $q(y)$, yielding the solutions

$$\begin{aligned} (x = 0.0841, \quad y = &\quad 0.7021) \\ (x = -0.7021, \quad y = &\quad -0.0841) \end{aligned}$$

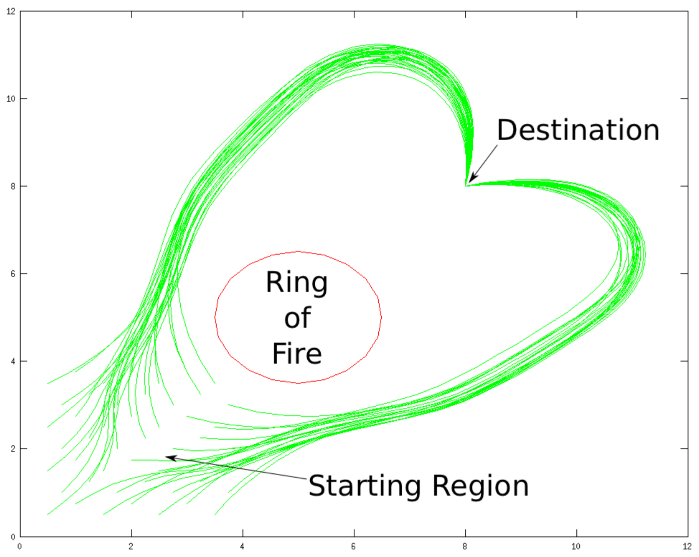
(c) Mark the roots of the resultant constructed in part (b) on the x-axis of your sketch from part (a).

See the figure for part (a).

9. 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 choregraphed 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 $Ax = b$) and constraints (i.e., $x_1 > 0$) to determine whether a point p falls within the triangle formed by the points p_1, p_2, and p_3.

We'll use barycentric coordinates, a weighted sum of the three vertices of the triangle. The point is inside the triangle if the weights are all between 0 and 1.

$$\begin{bmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$0 \le \alpha_1, \alpha_2, \alpha_3 \le 1$$

(b) In your favorite programming language, write an algorithm to interpolate three paths. First, pick the three paths p_1, p_2, and p_3 to interpolate. The unicycle's starting position should fall within the triangle formed by the three paths' starting points. There may be many valid sets of starting points, so develop criteria to choose one set.

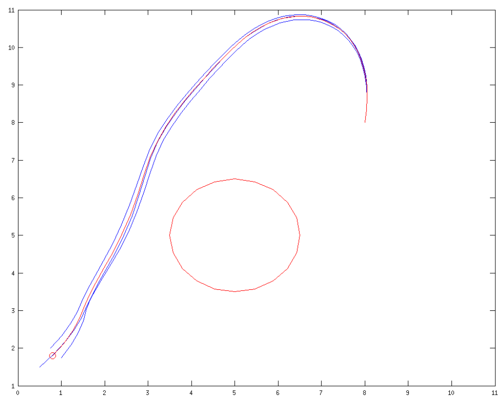
Construct a new path p as a weighted sum of the three paths. So, at each waypoint, $p(t) = \alpha_1 p_1(t) + \alpha_2 p_2(t) + \alpha_3 p_3(t)$.

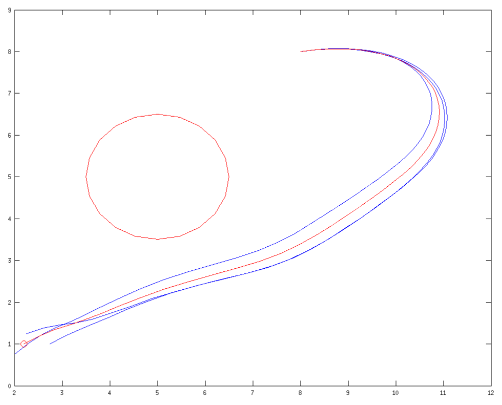
Discuss any decisions you made in your implementation. How did you pick a set of paths? How did you choose the weights α_i?

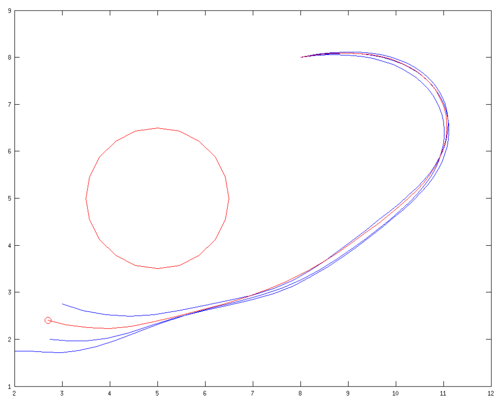
The paths must be in different homotopy classes, i.e., we cannot pick two paths that go on opposite sides of the circle. We can differentiate these paths by their starting position: each starting point leads to a path in the shortest direction for this particular problem. One metric for picking paths could be to minimize the sum of the distances to the starting points. The α_i should be the same ones we found in the first part using barycentric coordinates.

(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.

Here, the paths being interpolated are blue, and the interpolated path is red. The paths should not intersect the ring of fire.







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

We would need to introduce more homotopy classes, so that we didn't attempt to interpolate paths on opposite sides of an obstacle.