Math 104A - Homework 3

Due 7/7

2.4.6 Show that the following sequences converge linearly to $p = 0$. How large must n be before we have $|p_n - p| \leq 5 \cdot 10^{-2}$?

a $p_n = 1/n$.

Since
$$\frac{|p_{n+1} - 0|}{|p_n - 0|} = \frac{1/(n+1)}{1/n} = \frac{n}{n+1} \to 1$$
as $n \to \infty$, we have that p_n converges linearly to 0. In order for $|p_n - p| < 5 \cdot 10^{-2}$, we need
$$\frac{1}{n} < 5 \cdot 10^{-2}$$
$$n > \frac{10^2}{5} = 20.$$

b $p_n = 1/n^2$.

Since
$$\frac{|p_{n+1} - 0|}{|p_n - 0|} = \frac{1/(n+1)^2}{1/n^2} = \frac{n^2}{(n+1)^2} \to 1$$
as $n \to \infty$, we have that p_n converges linearly to 0. In order for $|p_n - p| < 5 \cdot 10^{-2}$, we need
$$\frac{1}{n^2} < 5 \cdot 10^{-2}$$
$$n^2 > \frac{10^2}{5}$$
$$n > \frac{10}{\sqrt{5}} \approx 4.47.$$

2.4.7a Show that for any positive integer k, the sequence defined by $p_n = 1/n^k$ converges linearly to $p = 0$.

Since
$$\frac{|p_{n+1} - 0|}{|p_n - 0|} = \frac{1/(n+1)^k}{1/n^k} = \frac{n^k}{(n+1)^k} \to 1$$
as $n \to \infty$, we have that p_n converges linearly to 0 for any integer $k > 0$.

2.4.8a Show that the sequence $p_n = 10^{-2^n}$ converges quadratically to 0.

Since
$$\frac{|p_{n+1} - 0|}{|p_n - 0|^2} = \frac{10^{-2^{(n+1)}}}{10^{-2 \cdot 2^n}} = \frac{10^{2^{(n+1)}}}{10^{2^{(n+1)}}} \to 1$$
as $n \to \infty$, we have that p_n converges quadratically to 0.

2.4.8b Show that the sequence $p_n = 10^{-n^k}$ does not converge quadratically, regardless of the size of the exponent k.

Since
$$\frac{|p_{n+1} - 0|}{|p_n - 0|^2} = \frac{10^{-(n+1)^k}}{10^{-2n^k}} = 10^{2n^k - (n+1)^k} \to \infty$$
as $n \to \infty$, we have that p_n does not converge quadratically to 0, for any positive integer k.

2.4.9a Construct a sequence that converges to 0 of order 3.

Let $p_n = 10^{-3^n}$. Then since
$$\frac{|p_{n+1} - 0|}{|p_n - 0|^3} = \frac{10^{-3^{(n+1)}}}{10^{-3\cdot 3^n}} = \frac{10^{3^{(n+1)}}}{10^{3^{(n+1)}}} \to 1,$$
we have that p_n converges to 0 of order 3.

2.4.9b Suppose $\alpha > 1$. Construct a sequence that converges to 0 of order α.

Let $p_n = 10^{-\alpha^n}$. Then since
$$\frac{|p_{n+1} - 0|}{|p_n - 0|^\alpha} = \frac{10^{-\alpha^{(n+1)}}}{10^{-\alpha\cdot\alpha^n}} = \frac{10^{\alpha^{(n+1)}}}{10^{\alpha^{(n+1)}}} \to 1,$$
we have that p_n converges to 0 of order α.

2.4.11 Show that the bisection method gives a sequence with an error bound that converges linearly to 0.

By theorem 2.1, we know that the error bound for the bisection method is $\frac{b-a}{2^n}$. Then since
$$\frac{(b-a)/2^{n+1}}{(b-a)/2^n} = \frac{1}{2}$$
we have that this error bound converges only linearly.

3.1.6 Use appropriate Lagrange interpolating polynomials of degrees one, two, and three to approximate each of the following: Note: You can do these by hand, but I highly suggest implementing Neville's iterated interpolation.

a $f(0.43)$ if $f(0) = 1$, $f(0.25) = 1.64872$, $f(0.5) = 2.71828$, $f(0.75) = 4.48169$

Using the attached code (`neville.m`), we get

```
>> x = [0;0.25;0.5;0.75];
>> f = [1;1.64872;2.71828;4.48169];

>> neville(0.43,x(2:3),f(2:3))       % degree one
ans =
   2.418803200000000
```

```
>> neville(0.43,x(2:4),f(2:4))        % degree two
ans =
   2.348863120000000

>> neville(0.43,x,f)                  % degree three
ans =
   2.360604734080000
```

b $f(0)$ if $f(-0.5) = 1.93750$, $f(-0.25) = 1.33203$, $f(0.25) = 0.800781$, $f(0.5) = 0.687500$

Using the attached code (`neville.m`), we get

```
>> x = [-0.5;-0.25;0.25;0.5];
>> f = [1.93750;1.33203;0.800781;0.687500];

>> neville(0,x(2:3),f(2:3))           % degree one
ans =
   1.066405500000000

>> neville(0,x(2:4),f(2:4))           % degree two
ans =
   1.015624333333333

>> neville(0,x,f)                     % degree three
ans =
   0.984374000000000
```

c $f(0.18)$ if $f(0.1) = -0.29004986$, $f(0.2) = -0.56079734$, $f(0.3) = -0.81401972$, $f(0.4) = -1.0526302$

Using the attached code (`neville.m`), we get

```
>> x = [0.1;0.2;0.3;0.4];
>> f = [-0.29004986;-0.56079734;-0.81401972;-1.0526302];

>> neville(0.18,x(1:2),f(1:2))           % degree one
ans =
  -0.506647844000000

>> neville(0.18,x(1:3),f(1:3))           % degree two
ans =
  -0.508049852000000

>> neville(0.18,x,f)                     % degree three
ans =
  -0.508143074400000
```

d $f(0.25)$ if $f(-1) = 0.86199480$, $f(-0.5) = 0.95802009$, $f(0) = 1.0986123$, $f(0.5) = 1.2943767$

Using the attached code (`neville.m`), we get

```
>> x = [-1;-0.5;0;0.5];
>> f = [0.86199480;0.95802009;1.0986123;1.2943767];

>> neville(0.25,x(3:4),f(3:4))        % degree one
ans =
   1.196494500000000

>> neville(0.25,x(2:4),f(2:4))        % degree two
ans =
   1.189597976250000

>> neville(0.25,x,f)                  % degree three
ans =
   1.188935146875000
```

3.1.11 Use Neville's method to approximate $\sqrt{3}$ with the following functions and values.

a $f(x) = 3^x$ and the nodes $x_0 = -2$, $x_1 = -1$, $x_2 = 0$, $x_3 = 1$, and $x_4 = 2$.

Using the attached code (`neville.m`) and letting $x = 0.5$, we get

```
>> x = [-2;-1;0;1;2];
>> f = 3.^x;

>> neville(0.5,x,f)
ans =
   1.708333333333333
```

b $f(x) = \sqrt{x}$ and the nodes $x_0 = 0$, $x_1 = 1$, $x_2 = 2$, $x_3 = 4$, $x_4 = 5$.

Using the attached code (`neville.m`) and letting $x = 3$, we get

```
>> x = [0;1;2;4;5];
>> f = sqrt(x);

>> neville(3,x,f)
ans =
   1.690606764623116
```

c Compare the accuracy of the approximation in parts (a) and (b).

The actual value of $\sqrt{3}$ is 1.732050807568877. In part (a), the absolute error was $|1.708333333333333 - \sqrt{3}| = 0.023717$, and in part (b), the absolute error was $|1.690606764623116 - \sqrt{3}| = 0.041444$, so part (a) was more accurate.

3.1.19 Construct the Lagrange interpolating polynomails for the following functions, and find a bound for the absolute error on the interval $[x_0, x_n]$.

a $f(x) = e^{2x}\cos(3x)$, $x_0 = 0, x_1 = 0.3, x_2 = 0.6$.

Using the attached code (`divided_diff.m`), we find the polynomial to be

```
>> x = [0;0.3;0.6];
>> f = exp(2*x).*cos(3*x);

>> divided_diff(x,f);
Polynomial is:
1.000+0.442(x-0.000)-11.220(x-0.000)(x-0.300)
```

A bound for the error can be found via the error term in theorem 3.3:

$$\begin{aligned}
|f(x) - P(x)| &= \left|\frac{f^{(3)}(\xi)}{6}x(x-0.3)(x-0.6)\right| \\
&= \left|\frac{-e^{2\xi}(9\sin(3\xi) + 46\cos(3\xi))}{6}x(x-0.3)(x-0.6)\right| \\
&\leq \left|\frac{-e^{2\cdot 0.6}(9\sin(3\cdot 0.6) + 46)}{6}0.0103923\right| \\
&= 0.31493
\end{aligned}$$

b $f(x) = \sin(\ln x)$, $x_0 = 2, x_1 = 2.4, x_2 = 2.6$

Using the attached code (`divided_diff.m`), we find the polynomial to be

```
>> x = [2;2.4;2.6];
>> f = sin(log(x));

>> divided_diff(x,f);
Polynomial is:
0.639+0.322(x-2.000)-0.131(x-2.000)(x-2.400)
```

An error bound is

$$\begin{aligned}
|f(x) - P(x)| &= \left|\frac{f^{(3)}(\xi)}{6}(x-2)(x-2.4)(x-2.6)\right| \\
&= \left|\frac{3\sin(\ln(\xi)) + \cos(\ln(\xi))}{6\xi^3}(x-2)(x-2.4)(x-2.6)\right| \\
&\leq \left|\frac{4}{6\cdot 2^3}0.016901\right| \\
&= 0.0014084
\end{aligned}$$

3.1.26 Inverse Interpolation Suppose $f \in C^1[a,b]$, $f'(x) \neq 0$. Let $x_0, \ldots, x_n$ be $n+1$ distinct numbers in $[a,b]$ with $f(x_k) = y_k$. To approximate the root p of f, construct the interpolating polynomial of degreen n on the nodes

$y_0, \ldots, y_n$ for the function f^{-1}. Since $y_k = f(x_k)$ and $0 = f(p)$, it follows that $f^{-1}(y_k) = x_k$ and $f^{-1}(0) = p$. Using iterated interpolation to approximate $f^{-1}(0)$ is called *iterated inverse interpolation*.

Use iterated inverse interpolation to find an approximation to the solution of $f(x) = x - e^{-x} = 0$, using the data

x	0.3	0.4	0.5	0.6
e^{-x}	0.740181	0.670320	0.606531	0.548812

Using the attached code (`neville.m`) and the data

$y = x - e^{-x}$	-0.440818	-0.270320	-0.106531	0.0511884
x	0.3	0.4	0.5	0.6

we get

```
>> x = [-0.440818;-0.270320;-0.106531;0.0511884];
>> f = [0.3;0.4;0.5;0.6];

>> neville(0,x,f)
ans =
   0.567142492111250
```

3.2.2 Use Algorithm 3.2 (Newton's divided differences) to construct interpolating polynomials of degree one, two, and three for the following data. Approximate the specified value using each of the polynomials.

a $f(0.43)$ if $f(0) = 1$, $f(0.25) = 1.64872$, $f(0.5) = 2.71828$, $f(0.75) = 4.48169$

Using the attached code (`divided_diff.m`), we get

```
>> x = [0;0.25;0.5;0.75];
>> f = [1;1.64872;2.71828;4.48169];

>> divided_diff(x(2:3),f(2:3));            % degree one
Polynomial is:
1.649+4.278(x-0.250)

>> divided_diff(x(2:4),f(2:4));            % degree two
Polynomial is:
1.649+4.278(x-0.250)+5.551(x-0.250)(x-0.500)

>> divided_diff(x,f);                      % degree three
Polynomial is:
1.000+2.595(x-0.000)+3.367(x-0.000)(x-0.250)...
   +2.912(x-0.000)(x-0.250)(x-0.500)
```

Substituting $x = 0.43$ into these expressions give

$$P_{1,2}(0.43) = 2.41880$$
$$P_{1,2,3}(0.43) = 2.34886$$
$$P_{0,1,2,3}(0.43) = 2.36060$$

b $f(0)$ if $f(-0.5) = 1.93750$, $f(-0.25) = 1.33203$, $f(0.25) = 0.800781$, $f(0.5) = 0.687500$

Using the attached code (`divided_diff.m`), we get

```
>> x = [-0.5;-0.25;0.25;0.5];
>> f = [1.93750;1.33203;0.800781;0.687500];

>> A = divided_diff(x(2:3),f(2:3));    % degree one
Polynomial is:
1.332-1.062(x+0.250)

>> A = divided_diff(x(2:4),f(2:4));    % degree two
Polynomial is:
1.332-1.062(x+0.250)+0.812(x+0.250)(x-0.250)

>> A = divided_diff(x,f);              % degree three
Polynomial is:
1.938-2.422(x+0.500)+1.813(x+0.500)(x+0.250)...
   -1.000(x+0.500)(x+0.250)(x-0.250)
```

Substituting $x = 0$ into these expressions give

$$P_{1,2}(0.43) = 1.06641$$
$$P_{1,2,3}(0.43) = 1.01562$$
$$P_{0,1,2,3}(0.43) = 0.984374$$

3.2.19 Given

$$P_n(x) = f[x_0] + f[x_0, x_1](x - x_0) + a_2(x - x_0)(x - x_1) + \cdots + a_n(x - x_0)\cdots(x - x_{n-1}),$$

use $P_n(x_2)$ to show that $a_2 = f[x_0, x_1, x_2]$.

Substituting x_2 for x into P_n gives

$$\begin{aligned} f[x_2] = P_n(x_2) &= f[x_0] + f[x_0, x_1](x_2 - x_0) + a_2(x_2 - x_0)(x_2 - x_1) \\ \Rightarrow a_2 &= \frac{f[x_2] - f[x_0]}{(x_2 - x_0)(x_2 - x_1)} - \frac{f[x_0, x_1]}{x_2 - x_1} \\ &= \frac{f[x_2] - f[x_1] + f[x_1] - f[x_0]}{(x_2 - x_0)(x_2 - x_1)} - \frac{f[x_0, x_1]}{x_2 - x_1} \\ &= \frac{f[x_1, x_2]}{x_2 - x_0} + \frac{f[x_1] - f[x_0]}{(x_2 - x_0)(x_2 - x_1)} - \frac{f[x_0, x_1]}{x_2 - x_1} \\ &= \frac{f[x_1, x_2]}{x_2 - x_0} + \frac{f[x_0, x_1]}{x_2 - x_0}\left(\frac{x_1 - x_0}{x_2 - x_1} - \frac{x_2 - x_0}{x_2 - x_1}\right) \\ &= \frac{f[x_1, x_2]}{x_2 - x_0} - \frac{f[x_0, x_1]}{x_2 - x_0} = f[x_0, x_1, x_2] \end{aligned}$$

3.3.2 Use algorithm 3.3 (Hermite interpolation) to construct an approximating polynomial for the following data.

a

x	$f(x)$	$f'(x)$
0	1	2
0.5	2.71828	5.43656

Using the attached code (`hermite.m`), we get

```
>> x = [0;0.5];
>> f = [1;2.71828];
>> fp = [2;5.43656];

>> hermite(x,f,fp);
Polynomial is:
1.000+2.000(x-0.000)+2.873(x-0.000)(x-0.000)...
   +2.254(x-0.000)(x-0.000)(x-0.500)
```

b

x	$f(x)$	$f'(x)$
-0.25	1.33203	0.437500
0.25	0.800781	-0.625000

Using the attached code (`hermite.m`), we get

```
>> x = [-0.25;0.25];
>> f = [1.33203;0.800781];
>> fp = [0.437500;-0.625000];

>> hermite(x,f,fp);
Polynomial is:
1.332+0.438(x+0.250)-3.000(x+0.250)(x+0.250)...
   +7.750(x+0.250)(x+0.250)(x-0.250)
```

c

x	$f(x)$	$f'(x)$
0.1	-0.29004996	-2.8019975
0.2	-0.56079734	-2.6159201
0.3	-0.81401972	-2.4533949

Using the attached code (`hermite.m`), we get

```
>> x = [0.1;0.2;0.3];
>> f = [-0.29004996;-0.56079734;-0.81401972];
>> fp = [-2.8019975;-2.6159201;-2.4533949];

>> hermite(x,f,fp);
Polynomial is:
-0.290-2.802(x-0.100)+0.945(x-0.100)(x-0.100)...
   -0.297(x-0.100)(x-0.100)(x-0.200)...
   -0.479(x-0.100)(x-0.100)(x-0.200)(x-0.200)...
   +0.050(x-0.100)(x-0.100)(x-0.200)(x-0.200)(x-0.300)
```

d

x	$f(x)$	$f'(x)$
-1	0.86199480	0.15536240
-0.5	0.95802009	0.2326954
0	1.0986123	0.33333333
0.5	1.2943767	0.45186776

Using the attached code (`hermite.m`), we get

```
>> x = [-1;-0.5;0;0.5];
>> f = [0.86199480;0.95802009;1.0986123;1.2943767];
>> fp = [0.15536240;0.2326954;0.33333333;0.45186776];

>> hermite(x,f,fp)
Polynomial is:
0.862+0.155(x+1.000)+0.073(x+1.000)(x+1.000)...
   +0.016(x+1.000)(x+1.000)(x+0.500)...
   -0.000(x+1.000)(x+1.000)(x+0.500)(x+0.500)...
   -0.001(x+1.000)(x+1.000)(x+0.500)(x+0.500)(x-0.000)...
   -0.000(x+1.000)(x+1.000)(x+0.500)(x+0.500)(x-0.000)(x-0.000)...
   +0.000(x+1.000)(x+1.000)(x+0.500)(x+0.500)(x-0.000)(x-0.000)(x-0.500)

ans =
   0.861994800000000
   0.155362400000000
   0.073376360000000
   0.015826559999999
  -0.000138159999999
  -0.000910680000004
  -0.000058479999996
   0.000050399999996
```

We include the coefficients because the last few are so close to zero that they were rounded off in the printout.

3.3.4 The data in 3.3.2 were generated using the following functions. Use the polynomials constructed in 3.3.2 for the given value of x to approximate $f(x)$, and calculate the absolute error.

a $f(x) = e^{2x}$; approximate $f(0.43)$.

Using the polynomial constructed in 3.3.2a, we have that

$$H_3(0.43) = 2.36207,$$

with an absolute error of

$$|f(0.43) - H_3(0.43)| = 0.00109122.$$

b $f(x) = x^4 - x^3 + x^2 - x + 1$; approximate $f(0)$.

Using the polynomial constructed in 3.3.2a, we have that

$$H_3(0) = 1.09453,$$

with an absolute error of

$$|f(0) - H_3(0)| = 0.0945305.$$

c $f(x) = x^2 \cos(x) - 3x$; approximate $f(0.18)$.

Using the polynomial constructed in 3.3.2a, we have that

$$H_5(0.18) = -0.508123,$$

with an absolute error of

$$|f(0.18) - H_3(0.18)| = 5.34234 \times 10^{-9}.$$

d $f(x) = \ln(e^x + 2)$; approximate $f(0.25)$.

Using the polynomial constructed in 3.3.2a, we have that

$$H_7(0.25) = 1.18907,$$

with an absolute error of

$$|f(0.25) - H_7(0.25)| = 2.53426 \times 10^{-7}$$

3.3.10 A car traveling along a straight road is clocked at a number of points. The data from the obsercations are given in the following table, where the time is in seconds, the distance is in feet, and the speed is in feet per second.

Time	0	3	5	8	13
Distance	0	225	383	623	993
Speed	75	77	80	74	72

a Use a Hermite polynomial to predict the position of the car and it's speed when $t = 10$ s.

Using the attached code (`hermite.m`), we get

```
>> x = [0;3;5;8;13];
>> f = [0;225;383;623;993];
>> fp = [75;77;80;74;72];

>> hermite(x,f,fp);
```

with Hermite polynomial

$$\begin{aligned} H_9(t) = 75t &+ 0.222t^2(t-3) - 3.11 \cdot 10^{-2}t^2(t-3)^2 \\ &- 6.44 \cdot 10^{-3}t^2(t-3)^2(t-5) \\ &+ 2.26 \cdot 10^{-3}t^2(t-3)^2(t-5)^2 \\ &- 9.13 \cdot 10^{-4}t^2(t-3)^2(t-5)^2(t-8) \\ &+ 1.31 \cdot 10^{-4}t^2(t-3)^2(t-5)^2(t-8)^2 \\ &- 2.02 \cdot 10^{-5}t^2(t-3)^2(t-5)^2(t-8)^2(t-13). \end{aligned}$$

Evaluating at $t = 10$ gives

$$H_9(10) = 742.503.$$

The derivative is given by

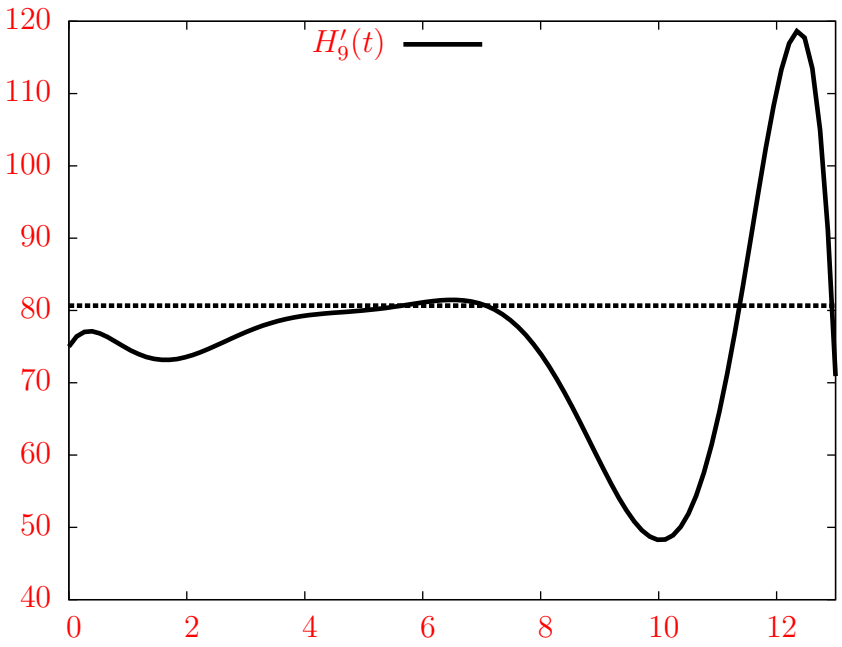
$$\begin{aligned} H_9'(t) = &-0.000182013t^8 + 0.00832472t^7 - 0.15313t^6 \\ &+ 1.45825t^5 - 7.69148t^4 + 22.0325t^3 \\ &- 30.2859t^2 + 14.3238t + 75 \end{aligned}$$

which gives the speed of the car as

$$H_9'(10) = 48$$

b Use the derivative of the Hermite polynomial to determine whether the car ever exceeds a 55 mi/h speed limit on the road. If so, what is the first time the car exceeds this speed?

Plotting the derivative,



we see that the speed of the car does exceed 55 mi/h (80.67 ft/s) somewhere between $t = 5$ and $t = 6$. Using the bisection method (see previous homework for code), we find that the car first exceeds 55 mi/h at $t = 5.64492$.

c What is the predicted maximum speed for the car?

From the sketch of the velocity, we see that the maximum speed of the car occurs somewhere between $t = 12$ and $t = 13$. Using the bisection method on the second derivative

$$H_9''(t) = -0.0014561t^7 + 0.0582731t^6 - 0.918778t^5 + 7.29124t^4 - 30.7659t^3 + 66.0974t^2 - 60.5719t + 14.3238,$$

we find that the car reaches is maximum speed of 119.417 ft/s (81.4207 mph) at $t = 12.3737$ s.

3.4.30 The 2004 Kentucky Derby was won by a horse named Smarty Jones in a time of 2:04.06 (2 minutes and 4.06 seconds) for the $1\frac{1}{4}$-mile race. Times at the quarter-mile, half-mile, and mile poles were 0:22.99, 0:46.73, and 1:37.35.

a Use these values together with the starting time to construct a free cubic spline for Smarty Jones' race.

Using the attached code (`natural_cubic_spline.m`), we get

```
>> x = [0;0.25;0.5;1;1.25];
>> f = [0;22.99;46.73;97.35;124.06];

>> [a,b,c,d] = natural_cubic_spline(x,f)

a =
```

```
                         0
    2.299000000000000e+01
    4.673000000000000e+01
    9.734999999999999e+01

b =
    9.139016393442623e+01
    9.309967213114754e+01
    9.697114754098361e+01
    1.054537704918033e+02

c =
                         0
    6.838032786885259e+00
    8.647868852458965e+00
    8.317377049180427e+00

d =
    9.117377049180346e+00
    2.413114754098274e+00
   -2.203278688523582e-01
   -1.108983606557390e+01
```

where the a, b, c, and d give the coefficients in each piece of the spline.

b Use the spline to predict the time at the three-quarter-mile pole, and compare this to the actual time of 1:11.80.

We use the third piece of the spline

$$S_3(x) = 46.73 + 96.97(x - 0.5) + 8.648(x - 0.5)^2 - 0.2203(x - 0.5)^3$$

to obtain the approximation $S_3(0.75) = 71.5098$, which has absolute error

$$|f(0.75) - S_3(0.75)| = |71.80 - 71.5098| = 0.2902.$$

c Use the spline to approximate Smarty Jones' starting speed and speed at the finish line.

We know that $S_1'(0) = b_1 = 91.39$ s/mi, which corresponds to about 39.39 mph. To get his finishing speed, we compute

$$\begin{aligned} S_4'(1.25) &= 105.45 + 2 \cdot 8.3174(1.25 - 1) - 3 \cdot 11.090(1.25 - 1) \\ &= 101.2912 \text{ s/mi} \end{aligned}$$

which corresponds to about 35.54 mph. That's one fast horse!

Code

```
%%% neville.m %%%

function [p,Q] = neville(x0,x,f)

    n = length(x);
    Q = zeros(n,n);
    Q(:,1) = f;

    for j=2:n
        for i=j:n
            Q(i,j) = ( (x0-x(i))*Q(i-1,j-1) - (x0-x(i-j+1))*Q(i,j-1) )/(x(i-j+1)-x
        end
    end

    p = Q(n,n);

end

%%% end of neville.m %%%



%%% divided_diff.m %%%

function F = divided_diff(x,f)

    n = length(x);
    Q = zeros(n,n);
    Q(:,1) = f;

    for i=2:n
        for j=2:i
            Q(i,j) = ( Q(i,j-1)-Q(i-1,j-1) )/(x(i)-x(i-j+1));
        end
    end

    F = diag(Q);

    fprintf('Polynomial is:\n');
    fprintf('%.3f',F(1));
    for i=2:n
        fprintf('%+.3f',F(i));
        for j=1:(i-1)
            fprintf('(x%+.3f)',x(j));
        end
    end
```

```
    fprintf('\n');

end

%%% end of divided_diff.m %%%



%%% hermite.m %%%

function [A,Q] = hermite(x,f,fp)

    n = length(x);
    z = zeros(2*n,1);
    Q = zeros(2*n,2*n);

    for i=1:n
        z(2*i-1) = x(i);
        z(2*i) = x(i);
        Q(2*i-1,1) = f(i);
        Q(2*i,1) = f(i);
        Q(2*i,2) = fp(i);
        if i ~= 1
            Q(2*i-1,2) = ( Q(2*i-1,1)-Q(2*i-2,1) )/( z(2*i-1)-z(2*i-2) );
        end
    end

    % below is a more efficient way to initialize z and Q
    %z(1:2:2*n-1) = x;
    %z(2:2:2*n) = x;
    %Q(1:2:2*n-1,1) = f;
    %Q(2:2:2*n,1) = f;
    %Q(2:2:2*n,2) = fp;
    %for i=3:2:2*n-1
        %Q(i,2) = ( Q(i,1)-Q(i-1,1) )/( z(i)-z(i-1) );
    %end

    for i=3:2*n
        for j=3:i
            Q(i,j) = ( Q(i,j-1)-Q(i-1,j-1) )/( z(i)-z(i-j+1) );
        end
    end

    A = diag(Q);

    fprintf('Polynomial is:\n');
    fprintf('%.3f',A(1));
    for i=2:2*n
```

```
        fprintf('%+.3f',A(i));
        for j=1:(i-1)
            fprintf('(x%+.3f)',-z(j));
        end
    end
    fprintf('\n');

end

%%% end of hermite.m %%%



%%% natural_cubic_spline.m %%%

function [a,b,c,d] = natural_cubic_spline(x,a)

    n = length(x)-1;
    h = zeros(n,1);

    alpha = zeros(n-1,1);
    l = zeros(n,1);
    mu = zeros(n,1);
    z = zeros(n+1,1);

    b = zeros(n+1,1);
    c = zeros(n+1,1);
    d = zeros(n+1,1);

    for i=0:n-1
        h(i+1) = x(i+2)-x(i+1);
    end
    for i=1:n-1
        alpha(i) = 3/h(i+1)*(a(i+2)-a(i+1)) - 3/h(i)*(a(i+1)-a(i));
    end

    l(1) = 1;
    mu(1) = 0;
    z(1) = 0;
    for i=1:n-1
        l(i+1) = 2*(x(i+2)-x(i)) - h(i)*mu(i);
        mu(i+1) = h(i+1)/l(i+1);
        z(i+1) = (alpha(i)-h(i)*z(i))/l(i+1);
    end

    z(n+1) = 0;
    c(n+1) = 0;
```

```
    for j=n-1:-1:0
        c(j+1) = z(j+1)-mu(j+1)*c(j+2);
        b(j+1) = (a(j+2)-a(j+1))/h(j+1)-h(j+1)*(c(j+2)+2*c(j+1))/3;
        d(j+1) = (c(j+2)-c(j+1))/(3*h(j+1));
    end

end

%%% end of natural_cubic_spline.m %%%
```