MATH 4220
Assignment # 4

Jarren Ralf

Due On: November 14, 2018

1 Let $f \in \mathcal{C}^3[a,b]$ be given at equidistant points $x_i = a + ih$, $i = 0, 1, ..., n$ where $nh = b - a$. Assume further that $f'(a)$ is given as well.

(a) Construct an algorithm for $\mathcal{C}^1$ piecewise quadratic interpolation of the given values. Thus, the interpolating function is written as

$$v(x) = s_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2, \quad x_i \leq x \leq x_{i+1}$$

for $i = 0, ..., n-1$ and your job is to specify an algorithm for determining the $3n$ coefficients a_i, b_i and c_i.

A quadratic spline is a great tool to interpolate data when you know nothing about the function, and instead you are given a set of data points. We will define a quadratic spline with the following conditions

$$\begin{cases} s_i(x_i) = f(x_i), & i = 0, 1, ..., n-1 \\ s_i(x_{i+1}) = f(x_{i+1}), & i = 0, 1, ..., n-1 \\ s_i'(x_{i+1}) = s_{i+1}'(x_{i+1}), & i = 0, 1, ..., n-2. \end{cases}$$

Let us take note of two important pieces of information. First, notice that at each data point (excluding the endpoints) we match the derivative of one spline with its adjacent spline. This means that not only would the function be continuous, it would also be differentiable. As the guidelines specify, we can be assured that $v \in \mathcal{C}^1[a,b]$.

Next, if you noticed how many equations the above conditions produced, we have $3n - 1$. This is a problem because we will end up with n polynomials each with 3 unknown coefficients, which implies we need $3n$ equations. Observe from the question that we are given a condition which will yield the final equation. Since, we may assume that we know $f'(a)$, we can say that

$$s_0'(x_0) = f'(a).$$

We will do a little bit of preliminary work in preparation for building the algorithm. First to take a derivative of the interpolating function, we get $s_i'(x) = b_i + 2c_i(x - x_i)$. Next, using the first condition, we end up with

$$s_i(x_i) = f(x_i) \implies s_i(x_i) = a_i + b_i(x_i - x_i) + c_i(x_i - x_i)^2 = a_i = f(x_i).$$

We know that we are dealing with $n+1$ equidistant points, so we can come up with a generalized statement representing the size of an interval with the following $h = x_{i+1} - x_i$. Notice here that h is not dependent on the ith interval. This is because we were told that we have equidistant abscissae, and thus given that $nh = b - a$. Clearly, a, b, and n are constants given at the beginning

of the task, hence solving for h yields a scalar representing the size of any interval. This will allow us to write the following equation more compact. From condition two, we can assert that

$$s_i(x_{i+1}) = f(x_{i+1}) \implies s_i(x_{i+1}) = a_i + b_i(x_{i+1} - x_i) + c_i(x_{i+1} - x_i)^2 = a_i + b_i h + c_i h^2 = f(x_{i+1}).$$

Allow us to simplify and rearrange our expression, we get

$$\begin{aligned} a_i + b_i h + c_i h^2 &= f(x_{i+1}) \\ f(x_i) + b_i h + c_i h^2 &= f(x_{i+1}) \\ b_i h + c_i h^2 &= f(x_{i+1}) - f(x_i) \\ b_i + c_i h &= \frac{f(x_{i+1}) - f(x_i)}{h} \\ b_i + c_i h &= f[x_i, x_{i+1}]. \end{aligned}$$

Finally, our third condition yields

$$s_i'(x_{i+1}) = s_{i+1}'(x_{i+1}) \implies s_i'(x_{i+1}) = b_i + 2c_i(x_{i+1} - x_i) = b_i + 2c_i h = b_{i+1}.$$

Now solving for c_i we get the equation

$$c_i = \frac{b_{i+1} - b_i}{2h}.$$

We will plug this expression in our equation that we derived from condition two. This becomes

$$\begin{aligned} b_i + c_i h &= f[x_i, x_{i+1}] \\ b_i + \left(\frac{b_{i+1} - b_i}{2h}\right) h &= f[x_i, x_{i+1}] \\ b_i + \frac{b_{i+1}}{2} - \frac{b_i}{2} &= f[x_i, x_{i+1}] \\ \frac{3b_{i+1}}{2} + \frac{b_i}{2} &= f[x_i, x_{i+1}] \\ 3b_{i+1} + b_i &= 2f[x_i, x_{i+1}]. \end{aligned}$$

There is one more observation to make. Our additional condition is still in terms of a and $s_0'(x_0)$. We know $a = x_0$ so we can express this as

$$s_0'(x_0) = b_0 + 2c_0(x_0 - x_0) = b_0 = f'(x_0).$$

With all of this information we are now able to devise an algorithm to find the coefficients for our quadratic spline. Algorithm 1 is at the top of the next page.

Algorithm 1: Construct a quadratic spline given equidistant abscissae and with the condition that $f'(a)$ is known.

function QuadracticSpline $(x_i, f(x_i), f'(x_0))$;
Input : A vector of abscissae values, x_i, a vector of function values, $f(x_i)$, each with the same size, namely, $i = 0, ..., n-1$, and the derivative of f at a, or equivalently $f'(x_0)$
Output: A vector of quadratic equations, s_i for $i = 0, ..., n-1$ and a vector of the coefficients; a_i, b_i, and c_i for $i = 0, ..., n-1$
for $i = 0$ **to** $n-1$ **do**
  $a_i = f(x_i)$;
end
/* Below is the implementation of our condition on the left endpoint i.e. the assumption we know $f'(a)$. */
$b_0 = f'(x_0)$;
/* Since we are given equidistant points, we can hardcode our value of h. */
$h = x_1 - x_0$;
for $i = 0$ **to** $n-1$ **do**
  /* Construct the system of equations for the unknowns $\{b_i\}_{i=0}^n$ with the following equation. The unknown coefficients will be stored in a vector called $\vec{b}$. Refer to this system of equations as matrix A and consequently define $\vec{z} = 2f[x_i, x_{i+1}]$ for $i = 0, ..., n-1$. */
  $3b_{i+1} + b_i = 2f[x_i, x_{i+1}]$;
end
/* Solve the system above for the coefficients $\{b_i\}_{i=0}^n$. */
$\vec{b} = A^{-1}\vec{z}$;
for $i = 0$ **to** $n-1$ **do**
  $c_i = \dfrac{b_{i+1} - b_i}{2h}$;
end
for $i = 0$ **to** $n-1$ **do**
  $s_i = a_i + b_i(x - x_i) + c_i(x - x_i)^2$;
end

(b) How accurate do you expect this approximation to be as a function of h? Justify.

To determine this we will use the theorem on polynomial interpolation error which says that if $f \in \mathcal{C}[a, b]$ is a function with $n+1$ bounded derivatives on $[a, b]$ and $p_n(x)$ is the polynomial interpolating f at the $n+1$ points $x_0, x_1, \ldots, x_n$, then for every $x \in [a, b]$, there exists a point $\xi = \xi(a) \in [a, b]$ such that

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

For our case, since we are interpolating with a degree two polynomial, $n = 2$. Thus we want a function with three bounded derivatives. We know that $f \in \mathcal{C}^3[a, b]$, so we know the function is three times differentiable, so let's just assume that its derivatives are bounded on $[a, b]$. We can now begin with

$$f(x) - p_2(x) = \frac{f'''(\xi)}{3!} \prod_{i=0}^{2} (x - x_i).$$

To start, we will take the absolute value of both sides, then expand the product,

$$|f(x) - p_2(x)| = \left| \frac{f'''(\xi)}{3!} \prod_{i=0}^{2} (x - x_i) \right| = \frac{|f'''(\xi)|}{6} |(x - x_0)(x - x_1)(x - x_2)| .$$

Next, let $g(x) = (x-x_0)(x-x_1)(x-x_2) = x^3 - x^2x_2 - x^2x_1 + xx_1x_2 - x^2x_0 + xx_0x_2 + xx_0x_1 - x_0x_1x_2$. Now collecting some terms, $g(x) = x^3 - (x_2 + x_1 + x_0)x^2 + (x_0x_1 + x_0x_2 + x_1x_2)x - x_0x_1x_2$. We want to bound this function so let's take a derivative and solve for 0. We get,

$$g'(x) = 3x^2 - 2(x_0 + x_1 + x_2)x + (x_0x_1 + x_0x_2 + x_1x_2) = 0.$$

Now to solve using the quadratic equation,

$$\begin{aligned}
x &= \frac{-(-2(x_0 + x_1 + x_2)) \pm \sqrt{(-2(x_0 + x_1 + x_2))^2 - 4(3)(x_0x_1 + x_0x_2 + x_1x_2)}}{2(3)} \\
&= \frac{2(x_0 + x_1 + x_2) \pm \sqrt{4(x_0 + x_1 + x_2)^2 - 12(x_0x_1 + x_0x_2 + x_1x_2)}}{6} \\
&= \frac{2(x_0 + x_1 + x_2) \pm 2\sqrt{(x_0^2 + 2x_0x_1 + 2x_0x_2 + x_1^2 + 2x_1x_2 + x_2^2) - 3(x_0x_1 + x_0x_2 + x_1x_2)}}{6} \\
&= \frac{(x_0 + x_1 + x_2) \pm \sqrt{x_0^2 + 2x_0x_1 + 2x_0x_2 + x_1^2 + 2x_1x_2 + x_2^2 - 3x_0x_1 - 3x_0x_2 - 3x_1x_2)}}{3} \\
&= \frac{(x_0 + x_1 + x_2) \pm \sqrt{x_0^2 + x_1^2 + x_2^2 - x_0x_1 - x_0x_2 - x_1x_2}}{3} \\
&= \frac{(x_0 + x_1 + x_2) \pm \sqrt{x_0(x_0 - x_1) + x_1(x_1 - x_2) + x_2(x_2 - x_0)}}{3} \\
&= \frac{(x_0 + x_1 + x_2) \pm \sqrt{x_2(x_2 - x_0) - x_0(x_1 - x_0) - x_1(x_2 - x_1)}}{3}.
\end{aligned}$$

Now using that fact that the points are equidistant and we can write them in terms of h, we have

$$\begin{aligned}
x &= \frac{(x_0 + x_1 + x_2) \pm \sqrt{2x_2h - x_0h - x_1h}}{3} \\
&= \frac{(x_0 + x_1 + x_2) \pm \sqrt{h(2x_2 - x_0 - x_1)}}{3} \\
&= \frac{(x_0 + x_1 + x_2) \pm \sqrt{h(x_2 - x_0 + x_2 - x_1)}}{3} \\
&= \frac{(x_0 + x_1 + x_2) \pm \sqrt{h(2h + h)}}{3} \\
&= \frac{(x_0 + x_1 + x_2) \pm \sqrt{3h^2}}{3} \\
&= \frac{(x_0 + x_1 + x_2) \pm \sqrt{3}|h|}{3} \text{ since } h > 0 \\
&= \frac{(x_0 + x_1 + x_2) \pm \sqrt{3}h}{3}.
\end{aligned}$$

We will now choose the larger of the two roots and plug it into our function above

$$
\begin{aligned}
|f(x)-p_2(x)| &\le \frac{|f'''(\xi)|}{6}\left|\left(\frac{(x_0+x_1+x_2)+\sqrt{3}h}{3}-x_0\right)\left(\frac{(x_0+x_1+x_2)+\sqrt{3}h}{3}\right.\right. \\
&\qquad \left.\left.-x_1\right)\left(\frac{(x_0+x_1+x_2)+\sqrt{3}h}{3}-x_2\right)\right| \\
&\le \frac{|f'''(\xi)|}{6}\left|\left(\frac{(x_0+x_1+x_2)+\sqrt{3}h-3x_0}{3}\right)\left(\frac{(x_0+x_1+x_2)+\sqrt{3}h-3x_1}{3}\right)\left(\frac{(x_0+x_1+x_2)+\sqrt{3}h-3x_2}{3}\right)\right| \\
&\le \frac{|f'''(\xi)|}{54}\left|\left(-2x_0+x_1+x_2+\sqrt{3}h\right)\left(x_0-2x_1+x_2+\sqrt{3}h\right)\left(x_0+x_1-2x_2+\sqrt{3}h\right)\right| \\
&\le \frac{|f'''(\xi)|}{54}\left|\left(x_1-x_0+x_2-x_0+\sqrt{3}h\right)\left(x_2-x_1+x_0-x_1+\sqrt{3}h\right)\left(-(x_2-x_0+x_2-x_1+\sqrt{3}h)\right)\right| \\
&\le \frac{|f'''(\xi)|}{54}\left|\left(h+2h+\sqrt{3}h\right)\left(h+(-h)+\sqrt{3}h\right)\left(-(2h+h+\sqrt{3}h)\right)\right| \\
&\le \frac{|f'''(\xi)|}{54}\left|\left((\sqrt{3}+3)h\right)\left(\sqrt{3}h\right)\left(-(\sqrt{3}+3)h\right)\right| \\
&\le \frac{h^3}{54}\left|f'''(\xi)\right|\left|\left((\sqrt{3}+3)\right)\left(\sqrt{3}\right)\left(-(\sqrt{3}+3)\right)\right| \\
&\le \frac{h^3}{54}\left|f'''(\xi)\right|\left|-12\sqrt{3}-18\right| \\
&\le \frac{h^3\left(12\sqrt{3}+18\right)}{54}\left|f'''(\xi)\right|.
\end{aligned}
$$

More generally, we can express this result as

$$|f(x)-p_2(x)| \le \frac{h^3}{54}\max_{a\le\xi\le b}|f'''(\xi)|.$$

Therefore we have found an upper bound for the error in terms of h and thus the interpolations accuracy with respect to h. As h gets small, the accuracy will increase. You can control this by using more data points, i.e. as n increases, h decreases.

2 Suppose we wish to interpolate $n+1$ data points ($n>2$) with a piecewise quadratic polynomial. How many continuous derivatives at most can this interpolant be guaranteed to have without becoming one global polynomial? Explain.

This interpolant can be guaranteed to have at most one continuous derivative without becoming one global polynomial. First, in order to create a quadratic, three points are needed. The question states that we are given $n>2$ data points, so this isn't an issue for this problem. If we want to create a quadratic between any two points in the interval, this implies that we need an additional condition to solve for the third polynomial coefficient. In this case, we use a condition that links the derivative of the spline at a point equal for the intervals on either side of this point. Instead, let's suppose that we matched concavity. Take two points and draw the line that these two points belong to. Now take a third point that is distinct from the previous two, but also lies on this line. Imagine that you create some quadratic using the first two points. If the concavity must match for the second interval then it is impossible to find a polynomial that will intersect this third point. Furthermore, suppose the points line up such that you could draw a quadratic through each of them. Then this polynomial interpolant would itself be a single quadratic function. Therefore, this interpolant cannot be guaranteed to have two continuous derivatives or more.

In addition to this argument, let's take an algebraic approach to the question. We will use the same form of the quadratic spline defined in question one. The spline and it's derivatives are,

$$s_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2, \qquad s_i'(x) = b_i + 2c_i(x - x_i), \qquad \text{and} \qquad s_i''(x) = 2c_i.$$

Let's suppose that we match the concavity of our spline at each of the data points. Then we need the following equations to match

$$\begin{aligned} s_i''(x_{i+1}) &= s_{i+1}''(x_{i+1}) \\ 2c_i &= 2c_{i+1} \\ c_i &= c_{i+1} \end{aligned}$$

for $i = 0, 1, \ldots, n-2$. Before we move on, imagine this result geometrically. This equation implies that the function is entirely concave down or concave up on the entire domain. Now we realize that in order for an interpolant to match concavity at all the data points, then the first derivatives must match as well. Put in another way, if a function is $\mathcal{C}^2$, then it must be $\mathcal{C}^1$. So the following condition will also be true,

$$\begin{aligned} s_i'(x_{i+1}) &= s_{i+1}'(x_{i+1}) \\ b_i + 2c_i(x_{i+1} - x_i) &= b_{i+1} + 2c_{i+1}(x_{i+1} - x_i) \\ b_i + 2c_i h_{i+1} &= b_{i+1} + 2c_i h_{i+1} \\ b_i &= b_{i+1} \end{aligned}$$

for $i = 0, 1, \ldots, n-2$. Now for completeness, let's check the function values at the same point,

$$\begin{aligned} s_i(x_{i+1}) &= s_{i+1}(x_{i+1}) \\ a_i + b_i(x_{i+1} - x_i) + c_i(x_{i+1} - x_i)^2 &= a_{i+1} + b_{i+1}(x_{i+1} - x_i) + c_{i+1}(x_{i+1} - x_i)^2 \\ a_i + b_i h_{i+1} + c_i h_{i+1}^2 &= a_{i+1} + b_i h_{i+1} + c_i h_{i+1}^2 \\ a_i &= a_{i+1} \end{aligned}$$

for $i = 0, 1, \ldots, n-1$. Therefore, we have shown that if we require the second derivative to match at the data points the coefficients of each section of the quadratic spline are the same. This implies that the interpolant would actually become one global polynomial. Therefore, for piecewise quadratic polynomial interpolants, there can only be guaranteed at most one continuous derivative on the domain.

3 Derive the matrix problem for cubic spline interpolation with the knot-a-knot condition. Show that the matrix is tridiagonal and strictly diagonally dominant.

As we already mentioned in class, we derived $4n - 2$ conditions to solve for $4n$ coefficients. We therefore need to find two more equations to add to the mix. Instead of using free boundary conditions, $v''(x_0) = v''(x_n) = 0$, which creates a natural spline, we will construct a not-a-knot spline. For this we will use third derivatives near the endpoints as follows

$$s_0'''(x_1) = s_1'''(x_1), \qquad \text{and} \qquad s_{n-2}'''(x_{n-1}) = s_{n-1}(x_{n-1}).$$

We have already differentiated our spline twice in our lecture notes, so we need to take one more derivative, $s_i'''(x) = 6d_i$. From the conditions above, we derive the following

$$\begin{aligned} s_0'''(x_1) &= s_1'''(x_1) \\ 6d_0 &= 6d_1 \\ d_0 &= d_1 \\ \frac{c_1 - c_0}{3h_0} &= \frac{c_2 - c_1}{3h_1} \\ h_1 c_1 - h_1 c_0 &= h_0 c_2 - h_0 c_1 \end{aligned} \qquad \begin{aligned} s_{n-2}'''(x_{n-1}) &= s_{n-1}(x_{n-1}) \\ 6d_{n-2} &= 6d_{n-1} \\ d_{n-2} &= d_{n-1}. \\ \frac{c_{n-1} - c_{n-2}}{3h_{n-2}} &= \frac{c_n - c_{n-1}}{3h_{n-1}} \\ h_{n-1} c_{n-1} - h_{n-1} c_{n-2} &= h_{n-2} c_n - h_{n-2} c_{n-1}. \end{aligned}$$

Let's now solve the equations above for c_0 and c_n, respectively. We then derive the following

$$\begin{aligned} h_1c_0 &= h_1c_1 + h_0c_1 - h_0c_2 & h_{n-2}c_n &= h_{n-2}c_{n-1} + h_{n-1}c_{n-1} - h_{n-1}c_{n-2} \\ c_0 &= c_1 + \frac{h_0}{h_1}(c_1 - c_2) & c_n &= c_{n-1} + \frac{h_{n-1}}{h_{n-2}}(c_{n-1} - c_{n-2}) \\ c_0 &= \left(1 + \frac{h_0}{h_1}\right)c_1 - \frac{h_0}{h_1}c_2 & c_n &= \left(1 + \frac{h_{n-1}}{h_{n-2}}\right)c_{n-1} - \frac{h_{n-1}}{h_{n-2}}c_{n-2}. \end{aligned}$$

Now since we have expressions for c_0 and c_n, we may now plug these these into our general equation for the system, namely $h_ic_i + (h_i + h_{i+1})c_{i+1} + h_{i+1}c_{i+2} = q_i$. The result is

$$\begin{aligned} h_0c_0 + 2(h_0 + h_1)c_1 + h_1c_2 &= \left(h_0 + \frac{h_0^2}{h_1}\right)c_1 - \frac{h_0^2}{h_1}c_2 + 2(h_0 + h_1)c_1 + h_1c_2 \\ &= h_0c_1 + \frac{h_0^2}{h_1}c_1 - \frac{h_0^2}{h_1}c_2 + 2h_0c_1 + 2h_1c_1 + h_1c_2 \\ &= 3h_0c_1 + 2h_1c_1 + \frac{h_0^2}{h_1}c_1 + h_1c_2 - \frac{h_0^2}{h_1}c_2 \\ &= \left(3h_0 + 2h_1 + \frac{h_0^2}{h_1}\right)c_1 + \left(h_1 - \frac{h_0^2}{h_1}\right)c_2 \end{aligned}$$

and,

$$\begin{aligned} h_{n-2}c_{n-2} + 2(h_{n-2} + h_{n-1})c_{n-1} + h_{n-1}c_n &= h_{n-2}c_{n-2} + 2(h_{n-2} + h_{n-1})c_{n-1} \\ &\quad + \left(h_{n-1} + \frac{h_{n-1}^2}{h_{n-2}}\right)c_{n-1}^2 - \frac{h_{n-1}^2}{h_{n-2}}c_{n-2} \\ = h_{n-2}c_{n-2} + 2h_{n-2}c_{n-1} + 2h_{n-1}c_{n-1} + h_{n-1}c_{n-1} + \frac{h_{n-1}^2}{h_{n-2}}c_{n-1} - \frac{h_{n-1}^2}{h_{n-2}}c_{n-2} \\ = h_{n-2}c_{n-2} - \frac{h_{n-1}^2}{h_{n-2}}c_{n-2} + 3h_{n-1}c_{n-1} + 2h_{n-2}c_{n-1} + \frac{h_{n-1}^2}{h_{n-2}}c_{n-1} \\ = \left(h_{n-2} - \frac{h_{n-1}^2}{h_{n-2}}\right)c_{n-2} + \left(3h_{n-1} + 2h_{n-2} + \frac{h_{n-1}^2}{h_{n-2}}\right)c_{n-1}. \end{aligned}$$

Observe that we would no longer have a tridiagonal matrix if we used the traditional set up of the system. To fix this, we will re-index the vector of parameters. Notice that c_0 is derivable from c_1 and c_2, likewise, so to is c_n derivable from c_{n-1} and c_{n-2}. So in fact, we are able to construct the following tridiagonal matrix,

$$\begin{pmatrix} 3h_0 + 2h_1 + \frac{h_0^2}{h_1} & h_1 - \frac{h_0^2}{h_1} & & & & \\ h_1 & 2(h_1 + h_2) & h_2 & & & \\ & \ddots & \ddots & \ddots & & \\ & & h_{n-3} & 2(h_{n-3} + h_{n-2}) & h_{n-2} & \\ & & & h_{n-2} - \frac{h_{n-1}^2}{h_{n-2}} & 3h_{n-1} + 2h_{n-2} + \frac{h_{n-1}^2}{h_{n-2}} \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_{n-1} \\ c_n \end{pmatrix} = \begin{pmatrix} 0 \\ q_1 \\ \vdots \\ q_{n-2} \\ 0 \end{pmatrix}.$$

Next, notice the diagonal of the matrix has not changed with exception of the first element and last element. So to check that this matrix is diagonally dominant, we need only check these two elements.

Since $A \in \mathcal{M}_n(\mathbb{R})$, matrix A is diagonally dominant if

$$\sum_{\substack{j=1 \\ i \neq j}}^{j=n} |a_{ij}| \leq |a_{ii}| \qquad \forall i \in 0, 1, ..., n-1.$$

For row one, $\left| h_1 - \frac{h_0^2}{h_1} \right| = \left| \frac{h_1^2 - h_0^2}{h_1} \right| = \left| \frac{(h_1 + h_0)(h_1 - h_0)}{h_1} \right| \leq \left| \frac{(h_0 + h_1)(h_0 + 2h_1)}{h_1} \right| = \left| \frac{h_0^2 + 3h_0h_1 + 2h_1^2}{h_1} \right| = \left| 3h_0 + 2h_1 + \frac{h_0^2}{h_1} \right|$. This condition is true, now for the last row. We get $\left| h_{n-2} - \frac{h_{n-1}^2}{h_{n-2}} \right| = \left| \frac{h_{n-2}^2 - h_{n-1}^2}{h_{n-2}} \right| = \left| \frac{(h_{n-2} + h_{n-1})(h_{n-2} - h_{n-1})}{h_{n-2}} \right| \leq \left| \frac{(h_{n-1} + h_{n-2})(h_{n-1} + 2h_{n-2})}{h_{n-2}} \right| = \left| \frac{h_{n-1}^2 + 3h_{n-1}h_{n-2} + 2h_{n-2}^2}{h_1} \right| = \left| 3h_{n-1} + 2h_{n-2} + \frac{h_{n-1}^2}{h_{n-2}} \right|$ which is also true. These two conditions combined with the work we did in class proves our resulting matrix is also diagonally dominant.

4 The gamma function is defined by

$$\Gamma(x) = \int_0^\infty t^{x-1} e^{-t} \, \mathrm{d}t, \quad x > 0.$$

It is known that for integers numbers the function has the value

$$\Gamma = (n-1)! = 1 \cdot 2 \cdot 3 \cdots (n-1).$$

Note that we define $0! = 1$. Thus for example, $(1,1), (2,1), (3,2)(4,6), (5,24)$ can be used as data points for the interpolating polynomial.

(a) Write a MATLAB script that computes the polynomial interpolant of degree four that passes through the above five data points.

Listing 1: MATLAB code to create a Monomial Interpolantion of the Gamma function

```
% Choose the number of positive integer data points to use for interpolation
n = 5;

% Set the evaluation points
x = linspace(0, n);

% Compute the x and y data points
xCoords = 1:n;
yCoords = factorial(xCoords    1);

% Create the Vandermonde matrix
V = fliplr(vander(xCoords));

% Solve for the coefficients
c = V\yCoords';

% Evaluate the monomial interpolating polynomial using Horner's algorithm
v = HornersAlgorithm(c, x);
```

(b) Write a program that computes a cubic spline to interpolate the same data (You may use MATLAB's *spline* function).

Listing 2: MATLAB code to create a Cubic Spline Interpolantion of the Gamma function

```
% Choose the number of positive integer data points to use for interpolation
n = 5;

% Set the evaluation points
x = linspace(0, n);

% Compute the x and y data points
xCoords = 1:n;
yCoords = factorial(xCoords   1);

% Evaluate the cubic spline
s = spline(xCoords, yCoords, x);
```

(c) Plot the two interpolants you found on the same graph, along with a plot of the gamma function itself, which can be reproduced using the MATLAB command *gamma*.

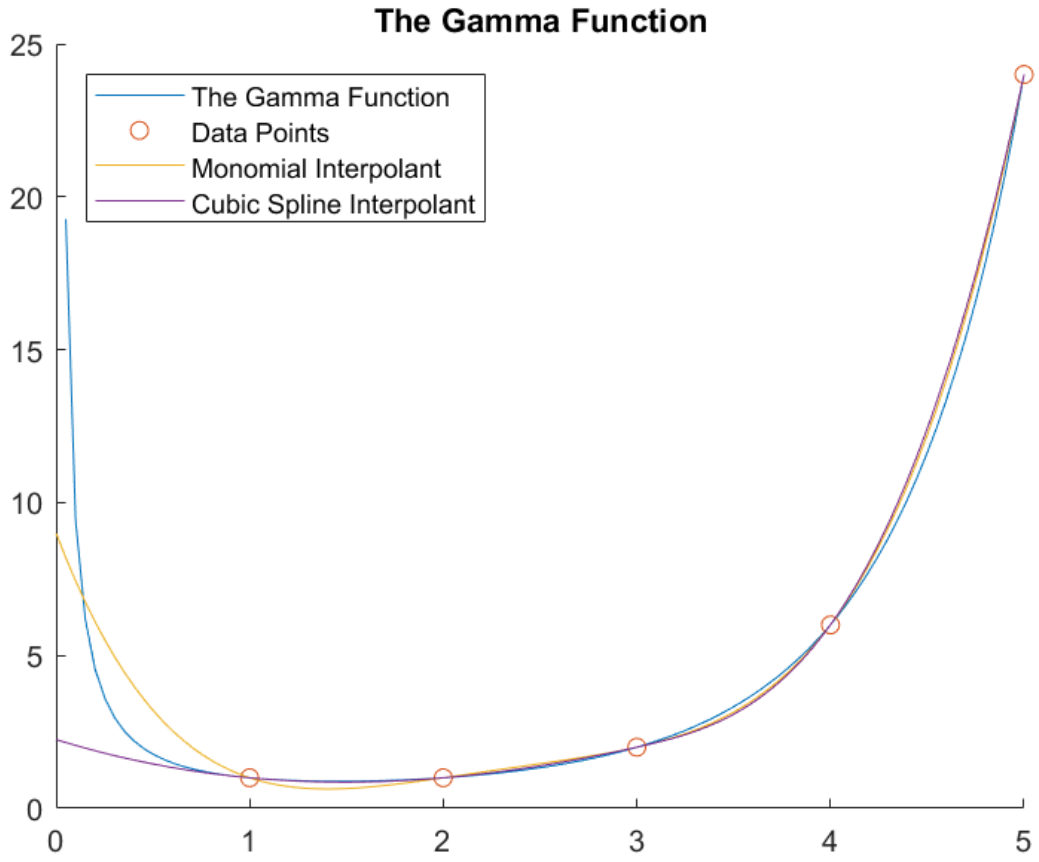
Listing 3: Plotting the Gamma function with Monomial and Cubic Spline Interpolantions in MATLAB

```
% Choose the number of positive integer data points to use for interpolation
n = 5;

% Set the evaluation points and compute the x and y data points
x = linspace(0, n);
xCoords = 1:n;
yCoords = factorial(xCoords   1);

% Evaluate the monomial interpolating polynomial using Horner's algorithm
V = fliplr(vander(xCoords));
c = V\yCoords';
v = HornersAlgorithm(c, x);

% Evaluate the cubic spline
s = spline(xCoords, yCoords, x);

% Graph the Gamma function, and the interpolants
hold on
    plot(x, gamma(x))
    plot(xCoords, yCoords, 'o');
    plot(x, v);
    plot(x, s);
    legend('The Gamma Function', 'Data Points', 'Monomial Interpolant', ...
           'Cubic Spline Interpolant', 'Location', 'Northwest')
    title('The Gamma Function');
hold off
```

(d) Plot the errors in the two interpolants on the same graph. What are your observations?

Both interpolations seem to do a good job in representing the Gamma function. Both interpolations experience larger errors prior to the first data point, between zero and one. You can clearly see in Figure 2, between the data points at four and five, there is a slight error that is a bit more visible than the error in other sections. Compared to each other, if you strictly look between the data range, neither interpolation is a clear winner. In the interval below one, the cubic spline is a fair bit more accurate until one point at approximately $x = 0.3$, where the cubic spline error intersects the monomial error and exceeds it as x gets small.

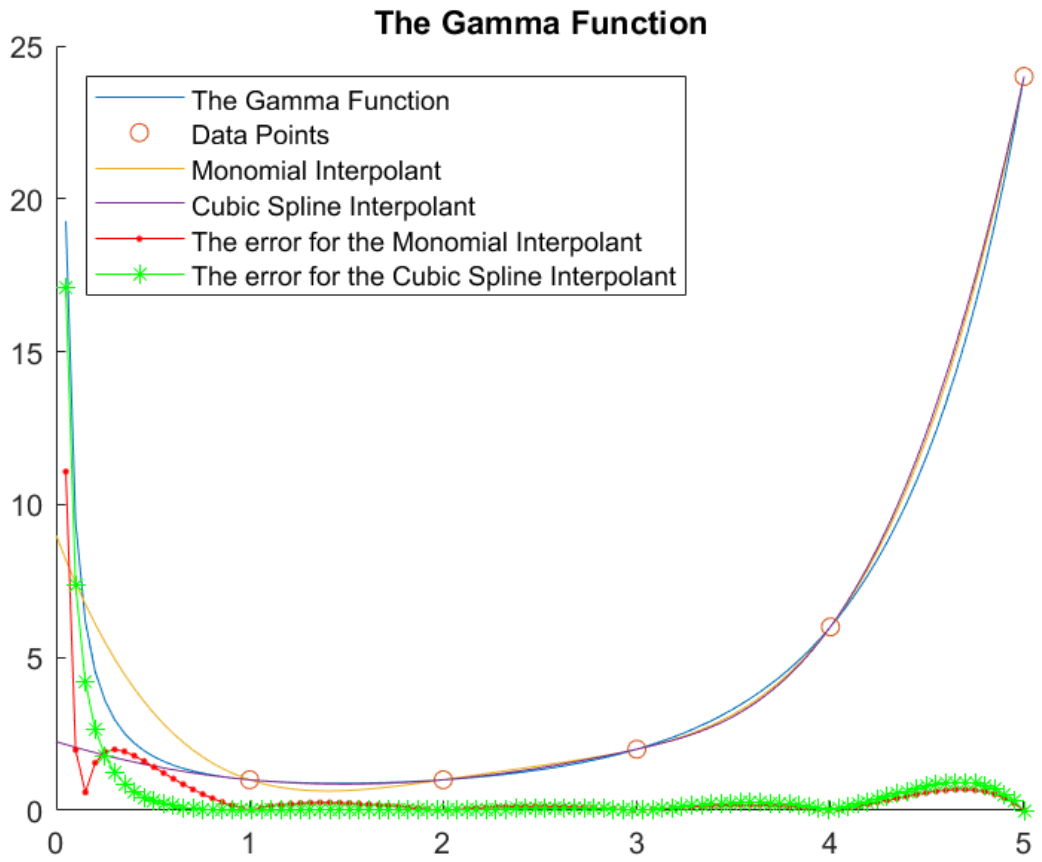
Figure 1: Graph of the Gamma function, along with Monomial and Cubic Spline Interpolations



Listing 4: Plotting the errors in Interpolation of the Gamma function using MATLAB

```
% Choose the number of positive integer data points to use for interpolation
n = 5;

% Set the evaluation points and compute the x and y data points
x = linspace(0, n);
xCoords = 1:n;
yCoords = factorial(xCoords   1);

% Evaluate the monomial interpolating polynomial using Horner's algorithm
V = fliplr(vander(xCoords));
c = V\yCoords';
v = HornersAlgorithm(c, x);

% Evaluate the cubic spline
s = spline(xCoords, yCoords, x);

% Graph the Gamma function, and the interpolants
hold on
    plot(x, gamma(x))
    plot(xCoords, yCoords, 'o');
    plot(x, v);
    plot(x, s);
    plot(x, abs(gamma(x)   v), '. r');
    plot(x, abs(gamma(x)   s), '* g');
    legend('The Gamma Function', 'Data Points', 'Monomial Interpolant', ...
           'Cubic Spline Interpolant', 'The error for the Monomial Interpolant', ...
```

Figure 2: Graph of the Gamma function, along with Monomial and Cubic Spline Interpolations and each of their Associated Errors




```
            'The error for the Cubic Spline Interpolant', 'Location', 'Northwest')
    title('The Gamma Function');
hold off
```

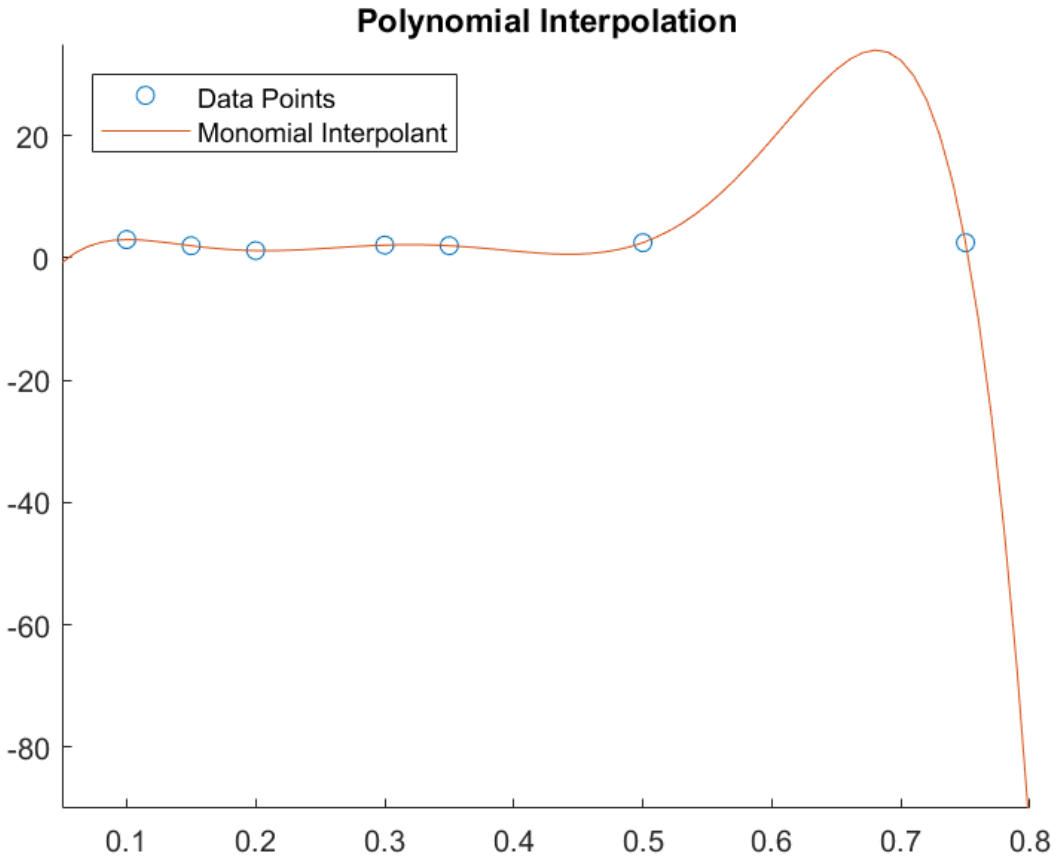
5 Consider interpolating the data $(x_0, y_0), ..., (x_6, y_6)$ given by

x	0.1	0.15	0.2	0.3	0.35	0.5	0.75
y	3.0	2.0	1.2	2.1	2.0	2.5	2.5

Construct the five interpolants specifed below (you may use available software), evaluating them at the points $0.05 : 0.01 : 0.8$, plot, and comment on their respective properties:

(a) a polynomial interpolant;

Figure 3: Monomial Interpolation based on the given Data Points

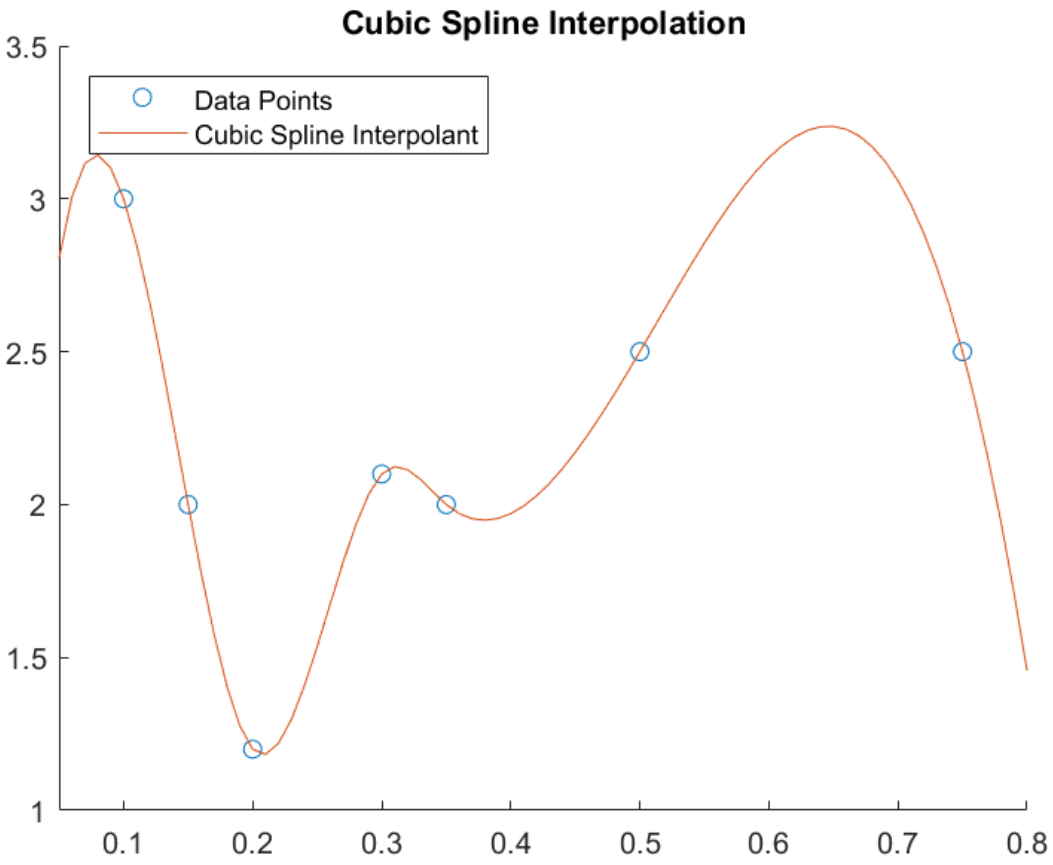


Listing 5: MATLAB code to create a Monomial Interpolantion based on the given Data Points

```
% Set the evaluation points
x = 0.05:0.01:0.8;

% Set the x and y data points
xCoords = [0.1 0.15 0.2 0.3 0.35 0.5 0.75];
yCoords = [3.0 2.00 1.2 2.1 2.00 2.5 2.50];

% Create the Vandermonde matrix
V = fliplr(vander(xCoords));

% Solve for the coefficients
c = V\yCoords';

% Evaluate the monomial interpolating polynomial using Horner's algorithm
v = HornersAlgorithm(c, x);

% Graph the Monomial interpolant and the data points
hold on
    plot(xCoords, yCoords, 'o');
    plot(x, v);
    axis([0.05 0.8 90 35])
    legend('Data Points', 'Monomial Interpolant', 'Location', 'Northwest');
    title('Polynomial Interpolation');
hold off
```

(b) a cubic spline interpolant;

Figure 4: Cubic Spline Interpolation based on the given Data Points



Listing 6: MATLAB code to create a Cubic Spline Interpolantion based on the given Data Points

```
% Set the evaluation points
x = 0.05:0.01:0.8;

% Set the x and y data points
xCoords = [0.1 0.15 0.2 0.3 0.35 0.5 0.75];
yCoords = [3.0 2.00 1.2 2.1 2.00 2.5 2.50];

% Evaluate the cubic spline
s = spline(xCoords, yCoords, x);

% Graph the cubic spline interpolant and the data points
hold on
    plot(xCoords, yCoords, 'o');
    plot(x, s);
    axis([0.05 0.8 1 3.5])
    legend('Data Points', 'Cubic Spline Interpolant', 'Location', 'Northwest');
    title('Cubic Spline Interpolation');
hold off
```

(c) the interpolant

$$v(x) = \sum_{j=0}^{n} c_j \phi_j(x) = c_0\phi_0(x) + \cdots + c_n\phi_n(x),$$

where $n = 7, \phi_0(x) = 1$, and

$$\phi_j(x) = \sqrt{(x - x_{j-1})^2 + \epsilon^2} - \epsilon \quad j = 1, ..., n.$$

In addition to the n interpolation requirements, the condition $c_0 = -\sum_{j=1}^{n} c_j$ is imposed. Construct this interpolant with (i) $\epsilon = 0.1$, (ii) $\epsilon = 0.01$, and $\epsilon = 0.001$. Make as many observations as you can. What will happen if we let $\epsilon \to 0$?

The first thing to notice about our basis functions is that they are not polynomials. This means that we cannot apply the theorems we have learnt about interpolations because they apply strictly to polynomials. Since the interpolating function can be written in linear form, we can write it as a system of equations. Hence we construct the following matrix,

$$\begin{pmatrix} \phi_0(x_0) & \phi_1(x_0) & \phi_2(x_0) & \cdots & \phi_7(x_0) \\ \phi_0(x_1) & \phi_1(x_1) & \phi_2(x_1) & \cdots & \phi_7(x_1) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \phi_0(x_6) & \phi_1(x_6) & \phi_2(x_6) & \cdots & \phi_7(x_6) \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_7 \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_6 \end{pmatrix}.$$

Notice the dimensions of the matrix in contrast with the column vectors. We have a 7×8 system, while the coefficient vector has length 8, both compared to the 7 y-coordinates we have. One of the conditions we can impose on the system is that c_0 is the negative sum of all the following parameters. Additionally the basis functions that reside in the first column are all equal to one. Let's take one equation as an example and apply the given conditions. We construct the following,

$$\begin{aligned} v(x_0) &= c_0\phi_0(x) + c_1\phi_1(x) + c_2\phi_2(x) + \cdots + c_6\phi_6(x) + c_7\phi_7(x) \quad \text{since } \phi_0(x) = 1, \\ &= c_0 + c_1\phi_1(x) + c_2\phi_2(x) + \cdots + c_6\phi_6(x) + c_7\phi_7(x) \quad \text{since } c_0 = -\sum_{j=1}^{7} c_j, \\ &= -\sum_{j=1}^{7} c_j + c_1\phi_1(x) + c_2\phi_2(x) + \cdots + c_6\phi_6(x) + c_7\phi_7(x) \\ &= -c_1 - c_2 - \cdots - c_6 - c_7 + c_1\phi_1(x) + c_2\phi_2(x) + \cdots + c_6\phi_6(x) + c_7\phi_7(x) \\ &= c_1\phi_1(x) - c_1 + c_2\phi_2(x) - c_2 + \cdots + c_6\phi_6(x) - c_6 + c_7\phi_7(x) - c_7 \\ &= c_1\Big(\phi_1(x) - 1\Big) + c_2\Big(\phi_2(x) - 1\Big) + \cdots + c_6\Big(\phi_6(x) - 1\Big) + c_7\Big(\phi_7(x) - 1\Big). \end{aligned}$$

Consequently, we can rewrite the matrix without the first column. Therefore the system is

$$\begin{pmatrix} \phi_1(x_0) - 1 & \phi_2(x_0) - 1 & \phi_3(x_0) & \cdots & \phi_7(x_0) - 1 \\ \phi_1(x_1) - 1 & \phi_2(x_1) - 1 & \phi_3(x_1) & \cdots & \phi_7(x_1) - 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \phi_1(x_6) - 1 & \phi_2(x_6) - 1 & \phi_3(x_6) & \cdots & \phi_7(x_6) - 1 \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_7 \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_6 \end{pmatrix}.$$

This results in us having a 7×7 system. Notice the first constant, c_0 is missing because it is derivable from the original first column which is no longer part of our new system. Below is the matlab code used to build the interpolating polynomial.

Listing 7: Creating an Interpolating function from the chosen Basis functions and the given Data Points using MATLAB

```
% Set the evaluation points
x = 0.05:0.01:0.8;

% Set your choice/s of epsilon
epsilon = [0.1  0.01  0.001];

% This is the number of epsilon values we want to evaluate with
numEpsilons = length(epsilon);

% Set the x and y data points
xCoords = [0.1  0.15  0.2  0.3  0.35  0.5  0.75];
yCoords = [3.0  2.00  1.2  2.1  2.00  2.5  2.50];

% Set the dimensions of the square Matrix we will create
dims = length(xCoords);

% Initialize the matrix A for speed
A = ones(dims);

for e = 1:numEpsilons % Loop through each epsilon
    for i = 1:dims    % Fill each row of the matrix

        % Construct the matrix one row at a time
        A(i, :) = sqrt( (xCoords(i)   xCoords).^2 + epsilon(e)^2 )   epsilon(e)   1;
    end

    % Solve the system for the coefficients and find c0
    c  = A\yCoords';
    c0 = ( 1)*sum(c);

    % Initialize the interpolation with a value of c0 since phi_0 = 1
    v = c0;

    % Calculate the interpolation
    for j = 1:dims
        v = v + c(j)*(sqrt( (x   xCoords(j)).^2 + epsilon(e)^2 )   epsilon(e));
    end

    % Graph the interpolation for each value of epsilon
    subplot(numEpsilons, 1, e);
    hold on
        plot(xCoords, yCoords, 'o');
        plot(x, v);
        axis([0.05  0.8  1  3.5])
        title(['Interpolation with Basis Functions: $\phi_j(x)'  ...
            '= \sqrt{(x   x_{j   1})^2 + \epsilon^2}   \epsilon$'...
            ' for $\epsilon =$', num2str(epsilon(e))], 'interpreter', 'latex');
    hold off
end
```

On the next page we have three graphs of the interpolating functions for various values of epsilon. There is a clear change in the graphs as epsilon gets smaller. The plot with the smallest epsilon is beginning to look similiar to a piecewise linear interpolation. Is this a reasonable expectation? We can check by taking a limit as epsilon approaches zero of $\phi_j(x)$. The result is,

$$\lim_{\epsilon \to 0} \phi_j(x) = \lim_{\epsilon \to 0} \sqrt{(x - x_{j-1})^2 + \epsilon^2} - \epsilon = \sqrt{(x - x_{j-1})^2} = |x - x_{j-1}|.$$

The absolute value of the difference of two values is a way of calculating distance, or making straight lines. So this will precisely become a broken line interpolation as epsilon approaches zero.

If we compare this plot to the monomial interpolation in Figure 3, you can see the range of the monomial is much more sporadic. This would certainly make the interpolant in part c) more favourable to interpolate this data. However, compare the top graph below to Figure 4, and then you see that they are very similar in terms of shape and range. Both graphs are completely contained in the range of $[1, 3.5]$. The final difference is that for the interpolation in part c) the maximum is at the left endpoint whereas with the cubic spline, it is at about 6.5. But still, either of these two would make for a fairly accurate interpolation, depending on the user's tolerance.

Figure 5: Custom Interpolation from the chosen Basis functions and the given Data Points