Math 361S Lecture Notes
Interpolation I

Jeffrey Wong
March 21, 2018

Figures to be added; will be updated after Friday.

Topics covered

• Interpolation: basics
  ◦ Introduction
  ◦ Lagrange/Newton forms
  ◦ Error analysis

• Divided differences
  ◦ Newton’s formula

• Hermite interpolation
  ◦ ...via divided differences
  ◦ Error formula
1 Basic interpolation

1.1 Introduction

Consider a set of \( n + 1 \) points (or nodes) on an interval,
\[
x_0 < x_1 < \cdots < x_n
\]
and associated function values \( y_i \), presumably coming from some unknown function \( f \) (so \( y_i = f(x_i) \)). The goal of interpolation is to construct a simple function \( p \) that passes through (‘interpolates’) the data
\[
(x_0, y_0), (x_1, y_1), \cdots (x_n, y_n)
\]
and produces a useful approximation to the function \( f(x) \) over the whole interval spanned by the nodes. Note that if the data comes from a function \( f \) say that ‘\( p \) interpolates \( f \)’.

In particular, we will study polynomial interpolation: we seek a polynomial \( p_n(x) \) of degree \( n \) such that
\[
p_n(x_j) = y_j, \quad j = 0, 1, \cdots n.
\]
As will be shown shortly, the following is true:

**Theorem.** A unique polynomial \( p_n \) of degree at most \( n \) interpolates the points \( \{(x_j, y_j)\}_{j=0}^n \).

Interpolation is a fundamental tool in numerical analysis because it lets us construct an approximation from only a finite set of data, filling in the unknown space between the known points. We have already used it in deriving the secant method (interpolating the function with a line), and will use it to estimate derivatives, integrals and solve ODEs.

1.2 Lagrange form of the interpolant

Although the interpolant \( p_n \) is unique, there are several ways to construct it that have different uses. Two are of particular note (and as a byproduct, either construction will prove the theorem stated previously).

One could write the polynomial in the familiar way,
\[
p_n = a_0 + a_1 x + \cdots + a_n x^n
\]
and then solve the resulting system of equations \( p_n(x_j) = y_j \), but this turns out not to be a good idea. A better choice of basis than \( \{1, x, \cdots, x^n\} \) can be made.

One way is to use the **Lagrange basis**. Consider a set of nodes \( x_0, \cdots x_n \) and fix an index \( k \). The \( k \)-th Lagrange basis polynomial \( \ell_k(x) \) is the polynomial of degree \( n \) that takes the value 1 at \( x_k \) and zero at all other \( x_j \)'s:
\[
\ell_k(x_j) = \delta_{jk}.
\]
Here $\delta_{jk}$ is the Kronecker delta:

$$\delta_{jk} = \begin{cases} 1 & j = k \\ 0 & j \neq k. \end{cases}$$

This polynomial is an interpolant for the data $(x_k, \delta_{jk})$. An explicit formula for $\ell_k$ is easy to find by factoring. Since it has degree $n$ and has zeros at $x_j$ for $j \neq k$, we must have

$$\ell_k(x) = c \prod_{j \neq k} (x - x_j).$$

The constant $c$ is determined by the condition $\ell_k(x_k) = 1$:

$$\ell_k(x) = \prod_{j \neq k} \frac{x - x_j}{x_k - x_j}.$$ 

The Lagrange interpolant for the set of data $(x_j, y_j)$ is then

$$P(x) = \sum_{j=0}^{n} y_j \ell_j(x)$$

noting that only one basis function $\ell_j$ is non-zero at a given $x_k$. To check explicitly:

$$P(x_j) = \sum_{k=0}^{n} y_k \ell_k(x_j) = \sum_{k=0}^{n} y_k \delta_{jk} = y_j.$$ 

While sometimes useful in theory, this form is not typically used in computation except in a few cases where the fixed basis functions are convenient.

For example, we can construct a line through two points $(x_0, y_0)$ and $(x_1, y_1)$. The Lagrange basis functions are

$$\ell_0(x) = \frac{x - x_1}{x_0 - x_1}, \quad \ell_1(x) = \frac{x - x_0}{x_1 - x_0},$$

so

$$p_1(x) = y_0 \frac{x - x_1}{x_0 - x_1} + y_1 \frac{x - x_0}{x_1 - x_0}.$$ 

Some effort is required to compute and evaluate the interpolant efficiently in the Lagrange form (not considered in detail here).

### 1.3 Newton form

The idea here is to build up the polynomial inductively, adding one point at a time. Let $p_k$ (with degree $k$) interpolate $(x_0, y_0), \ldots, (x_k, y_k)$. Then

$$p_0(x) = y_0.$$ 

Now suppose $p_{k-1}$ is known. We wish to add the point $(x_k, y_k)$ and adjust $p_{k-1}$ so that it also passes through the new point, adding one to the degree. If $p_k$ is written in the form

$$p_k = p_{k-1} + c_k (x - x_0) \cdots (x - x_{k-1}).$$
for a constant \( c_k \) then \( p_k \) still interpolates the points \((x_0, y_0), \cdots (x_{k-1}, y_{k-1})\) because the added term is zero at these nodes. To get \( p_k \), we need only choose \( c_k \) so that

\[
P_k(x_k) = y_k.
\]

This inductively constructs \( p_k \) in ‘Newton’ form

\[
p_k = c_0 + c_1(x - x_0) + c_2(x - x_1)(x - x_0) + \cdots + c_k(x - x_0) \cdots (x - x_{k-1})
\]

or more succinctly (with \( \prod_{i=0}^{k-1} \) understood to be 1)

\[
p_k = \sum_{j=0}^{n} c_j \prod_{i=0}^{j-1} (x - x_i).
\]

Notice that \( c_k \) depends only on the points up to \( x_k \). So if we need to add more interpolating points, the existing coefficients will remain unchanged. This property is convenient in applications where one might need to construct the interpolant and then add data points later.

Moreover, note that \( p_n \) can be evaluated efficiently using Horner’s method:

\[
p_n = c_0 + (x - x_0) (c_1 + (x - x_1) (c_2 + \cdots)).
\]

The Newton form is good for computation; as we will see shortly, there is an easy algorithm for computing the \( c_k \)’s as well.

### 1.4 Divided differences

(This closely follows the section in K&C, Chapter 6). The Newton form of the interpolating polynomial can be computed efficiently using divided differences. We define divided differences inductively as follows (using square brackets to distinguish from regular function evaluation):

\[
\begin{align*}
    f[x_i] &= y_i \\
    f[x_{i-1}, x_i] &= \frac{f[x_i] - f[x_{i-1}]}{x_i - x_{i-1}} \\
\end{align*}
\]

and in general

\[
    f[x_{i-j}, \cdots, x_i] = \frac{f[x_{i-j+1}, \cdots, x_i] - f[x_{i-j}, \cdots x_{i-1}]}{x_i - x_{i-j}}
\]

whenever the indices make sense (for \( i \geq j \geq 0 \)). Because the notation is unwieldy, let us define some shorthand:

\[
c_{ij} = f[x_{i-j}, x_{i-j+1}, \cdots, x_i].
\]

Then

\[
c_{i0} = y_i, \quad c_{ij} = \frac{c_{i,j-1} - c_{i-1,j-1}}{x_i - x_{i-j}}.
\]

It is not hard (but a little tedious) to show that
**Theorem.** In Newton form, the polynomial interpolating $f$ at $x_0, \ldots, x_n$ is

$$p_n(x) = \sum_{k=0}^{n} f[x_0, x_1, \ldots, x_k] \prod_{j=0}^{k-1} (x - x_0)$$

For example, let

$$f(x) = x^3 - 2x^2 + 1,$$  \quad $x_i = 0, 1, 2, 3.$

The divided differences are best arranged in a table:

<table>
<thead>
<tr>
<th>$i$</th>
<th>$x_i$</th>
<th>$\cdots$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>$f[x_0]$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>$f[x_1]$</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>$f[x_2]$</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>$f[x_3]$</td>
</tr>
</tbody>
</table>

The algorithm is simple: to compute column $j$, take successive differences of elements in the previous column $(j-1)$ and divide by $x_i - x_{i-j}$ (be careful with this, e.g. when computing column 2, the $x$-values are spaced two rows apart). The values used in the Newton polynomial are on the diagonal.

For the example, we have

<table>
<thead>
<tr>
<th>$i$</th>
<th>$x_i$</th>
<th>$c_{i0}$</th>
<th>$c_{i1}$</th>
<th>$c_{i2}$</th>
<th>$c_{i3}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>10</td>
<td>9</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

The result is

$$c_{11} = f[x_0, x_1] = -1, \quad c_{22} = f[x_0, x_1, x_2] = 1, \quad c_{33} = f[x_0, x_1, x_2, x_3] = 1$$

so the Newton form of the interpolating polynomial is

$$p_3(x) = 1 - x + x(x - 1) + x(x - 1)(x - 2).$$

Since the original function is a degree 3 polynomial, $p_3$ and $f$ are the same function (by the uniqueness property of the interpolant).

Note that if we only need to compute the entries for the interpolant, the algorithm can be improved to use only one column of space for the divided differences by overwriting entries into the same column at each step. For the example it would look like

$$\begin{bmatrix} 1 \\ 0 \end{bmatrix} \rightarrow \begin{bmatrix} 1 \\ -1 \end{bmatrix} \rightarrow \begin{bmatrix} 1 \\ -1 \end{bmatrix} \rightarrow \begin{bmatrix} 1 \\ -1 \end{bmatrix}.$$
1.5 Error analysis

Let us suppose that $p_n$ is the interpolant of the function $f$, i.e.

$$p_n(x_j) = f(x_j), \quad j = 0, \ldots, n.$$  

Assume that the nodes are contained in an interval $[a, b]$. It is important to have an estimate for the ‘interpolation error’

$$|f(x) - p_n(x)|$$

for $x \in [a, b]$ (but not one of the nodes, where the error is always zero). The fundamental result is the following:

**Theorem** (Error for Lagrange interpolation). Suppose $f \in C^{n+1}([a, b])$ with all the nodes contained in $[a, b]$. For every $x \in [a, b]$, there is a point $\xi_x \in (a, b)$ (depending on $x$) such that

$$f(x) = p_n(x) + \frac{f^{(n+1)}(\xi_x)}{(n+1)!} \prod_{j=0}^{n} (x - x_j).$$

Note that the error looks similar to Taylor’s theorem, but with $(x - x_0) \cdots (x - x_n)$ instead of $(x - x_0)^{n+1}$. The proof is essentially the same as for Taylor’s theorem (see textbook).

Most of the time, we are concerned with the interval spanned by the nodes, $[x_0, x_n]$ and not points outside this range. However, the theorem does apply when $x$ is to the right/left of all the nodes.

Observe that the error depends on the $(n + 1)$-th derivative of $f$. The usefulness of the bound depends on how much we know about the size of $f^{(n+1)}$ over the interval. If we are interpolating an unknown function, then this theoretical bound is not so useful for a practical error estimate without making some assumptions. This makes sense, since if the interpolant only uses information at the $n + 1$ nodes, it can miss $f$ doing arbitrarily bad things in between the nodes.

**Example:** Consider the parabola $f(x) = e^x$ and two points $x_0 = 1$ and $x_1 = 2$. The linear interpolant of $f$ through these points in Newton form is

$$p_1(x) = e + (e^2 - e)(x - x_0) = e + (e^2 - e)(x - 1).$$

According to the theorem, the error in the interval has the form

$$\frac{f''(\xi_x)}{2} (x - 1)(x - 2).$$

If $x \in [1, 2]$ then $\xi_x \in [1, 2]$ so the second derivative is bounded above by $e^2$:

$$|f''(\xi_x)| = e^{\xi_x} \leq e^2.$$
For the rest, bound above by the maximum over the interval:

\[ |(x - 1)(x - 2)| \leq \max_{x \in [1, 2]} |(x - 1)(x - 2)| = \frac{1}{4}. \]

Thus the maximum error between \( f \) and \( p_1 \) is \( e^2/4 \), i.e.

\[ |p_1(x) - f(x)| \leq e^2/4 \text{ for } x \in [1, 2]. \]

Suppose that we now use 10 equally spaced points in \([1, 2]\) \( (x_0, \ldots, x_9) \). Then the error is

\[ \frac{f^{(10)}(\xi_x)}{10!} \prod_{j=0}^{9} (x - x_j). \]

The derivative factor is bounded above by \( e^2 \) still. The product is too complicated to bound by taking the maximum; a crude estimate can be made by just bounded each factor on its own:

\[ |x - x_j| \leq 1 \]

so the interpolant satisfies

\[ |p_{10}(x) - f(x)| \leq \frac{1}{10!} \approx 3 \times 10^{-7}. \]

It would seem that adding more points leads to better approximations. This turns out to not always be true.

### 1.6 Discussion of error, interpolation schemes

The case above is an idealized one - functions like \( e^x \) and \( \sin x \) whose \( n \)-th derivative stays bounded. An interpolant with only a few points is not a good approximation (and we cannot expect it to be, given how little information it uses). When the function is well-behaved, we can just throw in more nodes. However, in general, it is dangerous to increase the number of nodes to try to improve the interpolant.

To be precise, suppose we have a function \( f(x) \) in \([a, b]\) and wish to construct a sequence of interpolants of higher and higher degree that will converge to \( f \) in the sense that

\[ \|f - p_n\|_\infty := \max_{x \in [a, b]} |f(x) - p_n(x)| \to 0 \text{ as } n \to \infty. \]

The notation \( \|\cdot\|_\infty \) is the max-norm for continuous functions on \([a, b]\):

\[ \|g\|_\infty := \max_{x \in [a, b]} |g(x)|. \]

Let’s suppose that for each \( n \), we pick a set of \( n + 1 \) nodes and construct \( p_n \) as the interpolant for those nodes (the set of nodes can change completely for each \( n \)). For instance, given the
interval $[-1, 1]$ we could pick $n + 1$ equally spaced points:

- $n = 2$: $x_0 = -1$, $x_1 = 1$
- $n = 3$: $x_0 = -1$, $x_1 = 0$, $x_2 = 1$

To see how badly things can go, consider the function

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

which seems reasonably well behaved - certainly smooth, at least, with no singularities in $[-1, 1]$. Let $p_n$ be the interpolant for nodes $x_0, \ldots x_n$ equally spaced in the interval $[-1, 1]$ (with $x_0 = -1$ and $x_n = 1$). As $n$ is increased, a disaster occurs (the Runge phenomenon) - the maximum error

$$\max_{x \in [-1, 1]} |p_n(x) - f(x)|$$

grows exponentially as $n \to \infty$. The polynomial oscillates wildly near the endpoints, getting worse and worse as $n$ increases. It is not too much of a surprise to find this: if the polynomial is increasing through one node, it needs to turn sharply to get back to the next node, causing rapid variation.

How does this relate to the error formula? The error has the form

$$\frac{f^{(n)}(\xi)}{n!} w_n(x), \quad w_n(x) = \prod_{j=0}^{n}(x - x_j).$$

It can be checked that if $n$ is large, $f^{(n)}$ grows quite rapidly with $n$, fast enough that the $w_n$ (decays exponentially) and $1/n!$ are not enough to counteract it.

By choosing the points so that $w_n$ is as small as possible, we can (rather miraculously) get a better set of nodes for which the interpolants work fine (see Chebyshev interpolation).

The analysis is subtle; one can be more precise and determine conditions on $f$ that distinguish the good from bad cases but this requires complex analysis. The result: if $f$ is analytic, then it is nice; if $f$ has a singularity in the complex plane too close too the interval on the real axis, then the 'equally spaced' interpolants may diverge.

So equally spaced points are not a good choice; one may wonder, then, whether there is a way of choosing nodes that will work for any $f$ (i.e. so that $p_n$ always converges to $f$). The answer turns out to be no (proven by Faber in 1914; this is really a standard result in functional analysis in disguise). \(^1\)

However, there is a nearly ideal choice of points (the Chebyshev nodes) for which counterexamples are obscure enough that they are not a cause for concern in practice.

\(^1\)Note: It is also true that given $f$, we can come up with a sequence of $p_n$’s that do converge to $f$, but the choice of nodes will depend on $f$ in some way.
For details, see Cheney’s *Approximation Theory* (these are all functional analysis questions in disguise).

**Key point:** The lesson here is that equally spaced interpolants of high degree can be disastrous (and should be avoided if possible). However, with the right choice of points, high degree interpolants can work fine (we’ll get to this later).

## 2 Hermite interpolation

Interpolants can also be constructed that match the derivatives of the function values at the nodes. To simplify matters, we will consider only one important case, Hermite interpolation, in which we construct a polynomial $p$ such that

$$ p(x_j) = f(x_j), \quad p'(x_j) = f'(x_j), \quad j = 0, 1, 2, \ldots n. \quad (1) $$

It is possible to generalize to allow any number of derivatives at each point. The main theorem is that

**Theorem.** Given a function $f$ and its derivatives at $x_j$ for $j = 0, 1, \ldots n$, there is a unique polynomial $p$ of degree $(at most) 2n + 1$ such that (1) holds.

Note that the degree is $2n + 1$. In general, the degree will one less than the number of given conditions (here there are two per node, for a total of $2n + 2$). The error formula is

$$ p(x) = f(x) + \frac{f^{(2n+2)}(x)}{(2n + 2)!} \prod_{j=0}^{n}(x - x_j)^2 $$

which is the same as the Lagrange formula for a set of $2n + 2$ points but with a duplicate set of $n + 1$ points (it is possible to make sense of this coincidence via some limits).

There is an Hermite basis analogous to the Lagrange basis but it is tedious to construct. Fortunately, Hermite interpolants are easy to compute via divided differences. We define a set of $2n + 2$ points

$$ z_{2i} = z_{2i+1} = x_i, \quad 0 \leq i \leq n $$

i.e. the new $z$-points are $x_0, x_0, x_1, x_1, x_2, x_2, \ldots$. Now we compute the divided differences $f[z_i, \ldots, z_{i+j}]$ the same way as before, except that if there is a division by zero we replace the quotient with a derivative:

$$ f[z_{2i}, z_{2i+1}] = f'(x_i) \text{ instead of } \frac{f[z_{2i+1}] - f[z_{2i}]}{z_{2i+1} - z_{2i}}. $$

This only changes the first (non-trivial) column; the rest of the divided differences work as before. The result is that

$$ p(x) = \sum_{j=0}^{2n+1} f[z_0, \ldots, z_j] \prod_{i=0}^{j-1}(x - z_i). $$
Again, this is the same as before but with a set of $2n + 2$ points that have repeats. For example, let us compute the cubic Hermite interpolant for the data

$$f(-1) = 2, \quad f'(-1) = -1, \quad f(1) = 0, \quad f'(1) = 3.$$ 

The divided difference table is (with the derivatives in boxes and diagonal entries used for $p(x)$ in red as before)

<table>
<thead>
<tr>
<th>$i$</th>
<th>$z_i$</th>
<th>$f[z_i]$</th>
<th>$f[z_i, z_{i-1}]$</th>
<th>$\vdots$</th>
<th>$\vdots$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>2</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>2 1</td>
<td></td>
</tr>
</tbody>
</table>

so the Hermite interpolant is

$$p(x) = 2 - (x + 1) + 0 \cdot (x + 1)^2 + 1 \cdot (x + 1)^2(x - 1).$$

Typically, derivative information greatly improves the quality of the interpolant (the disadvantage: we need to know the derivative). It can be shown that if $p_{2n+1}$ is the Hermite interpolant for the Runge example with $n + 1$ equally spaced points (from earlier), then $p_{2n+1}$ does converge to $f$. However, the high degree of the polynomial is still undesirable. In the next section, we will use the cubic Hermite interpolant to produce excellent approximations to a function by patching together cubic interpolants on small intervals.

3 Brief notes on splines

3.1 Setup

Suppose we are given a set of data $(x_i, y_i)$ for $i = 0, 1, \ldots, n$ from a function $f(x)$. Without access to the function, we do not have the option of changing the interpolation nodes. A high degree interpolant can be a problem, but we can avoid such issues by breaking the interval down into smaller parts and stitching together low degree ‘local’ interpolants on each part to form the full interpolating function.

To be precise, let us define interpolants

$$s_i(x) \text{ in } [x_i, x_{i+1}]$$

for $i = 0, \ldots, n - 1$, where $s_i$ interpolates only the data at the endpoints of the interval:

$$s_i(x_i) = y_i, \quad s_i(x_{i+1}) = y_{i+1}.$$ 

Our ‘global’ interpolant $S(x)$ for the whole set of data is just a piecewise function made up of these parts. Note that the interpolant is continuous since the local interpolants agree at the nodes.

By increasing the degree of the polynomial, we can introduce more degrees of freedom that
allow for better approximations. On popular method is to use **cubic splines**, in which we demand continuity not only of \( S \) but also \( S' \) and \( S'' \):

\[
s_i \text{ and } s_{i+1} \text{ have the same value/first/second derivatives at } x_{i+1}.
\]

Each spline is a cubic:

\[
s_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3.
\]

Requiring continuity of \( S, S' \) and \( S'' \) and that the \( s_i \)'s interpolate the data yields \( 4n - 2 \) conditions for the \( 4n \) unknowns. Two more constraints are needed at the endpoints; they depend on the desired properties of the spline.

The advantage of the spline approximation is that the error depends on the fourth derivative of \( f \), regardless of the number of points. Thus the interpolant will converge nicely to the function as \( n \to \infty \) (adding more points improves the approximation). Moreover, because cubic functions have a relatively simple shape (not too many oscillations), splines tend to be fairly good approximations even without many points.

Splines are commonly used, for example, as the computer’s representation of a complicated curve from some physical problem.