Another alternative is Müller's method which works with the same idea as secant method but using a quadratic model of the function (which requires 3 points as opposed to 2 for linear model in secant method).

**Secant Method**

Function \( p(x) \) is approx. by a linear model: secant passing through points \((x_0, p(x_0)), (x_1, p(x_1))\). Next iterate is root of linear model.

**Müller's Method**

Function \( p(x) \) is approx. by a quadratic model: parabola passing through points \((x_0, p(x_0)), (x_1, p(x_1)), (x_2, p(x_2))\) and \((x_3, p(x_3))\). Next iterate is root of quadratic.

In pseudocode:

Müller's method

Given \( x_0, x_1, x_2 \)

for \( k = 1 \ldots \) maxit

- Find coeff. \( a, b, c \) of quadratic
  \[
  q(x) = a(x - x_2)^2 + b(x - x_1) + c
  \]
  \((x_0, p(x_0)), (x_1, p(x_1)), (x_2, p(x_2))\)

- \( x_3 = \text{root of } q(x) \) closest to \( x_2 \)

- if \( |x_3 - x_2| < \text{tol} \) stop

- \( x_0 = x_1; x_1 = x_2; x_2 = x_3 \); prep for next iteration.

See book. We shall see an easy way of determining \( a, b, c \) when we get to interp.
Advantage of Müller's method is that:

- If used close to real root, it should not use complex arithmetic.
- If used close to complex root, then complex numbers occur naturally since roots of $q(3)$ may be complex.

One word of caution on finding roots of quadratic $q(3) = ax^2 + bx + c$.

**Usual formula:**

$$g_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

... may not be stable numerically because of catastrophic cancellation if $b$ is large and $a, c$ small.

In fact, if $b > 0$ then

$$g_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$ could be unstable (subtracting two numbers of similar magnitude)

$$g_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$ is fine (adding two numbers of some magnitude)

Fortunately, we can find alternate formulas where this catastrophic cancellation does not occur:

$$g_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} = \frac{-b + \sqrt{b^2 - 4ac}}{2a} + \frac{b + \sqrt{b^2 - 4ac}}{2a}$$

$$\frac{b^2 - 4ac - b^2}{(b + \sqrt{b^2 - 4ac})2a} = \frac{-2c}{b + \sqrt{b^2 - 4ac}}$$

$$= \frac{b^2 - 4ac - b^2}{(b + \sqrt{b^2 - 4ac})2a} = \frac{-2c}{b + \sqrt{b^2 - 4ac}}$$

The choice of root in Müller's method is:

$$g_3 - g_2 = \frac{-2c}{b + 5q(3)(b^2 - 4ac)}$$

- The catastrophic cancellation and denominator is largest in magnitude
- $g_3$ is closest root to $g_2$. 

CHAPTER 3  INTERPOLATION AND POLYNOMIAL APPROXIMATION

Idea: interpolation: given some data points, find a polynomial that passes through all data points.

extremely useful to:
- fill-in gaps in data from e.g. measurements
- numerical integration: integrating interpolating polynomials is easy!
- numerical differentiation: differentiating interpolating polynomials is easy!

Why would we expect to have a good match between function and polynomial?

Theorem (Weierstrass): Let \( f \in C^0[a,b] \).

\( \forall \epsilon > 0 \exists \text{ poly } p(x) \text{ s.t. } \forall x \ |f(x) - p(x)| < \epsilon \)

i.e. polynomials uniformly approximate continuous functions.

Remark: Taylor's polynomial does approximate well a function, but approximation is local meaning it quickly degrades far away from point \( x_0 \) where we do the expansion.

Interpolation allows us to get poly that is close to function over some interval.

3.1 Polynomial interpolation

Theorem on polynomial interpolation: If \( x_0, x_1, \ldots, x_m \) are \( m \) distinct real numbers, then for arbitrary values \( y_0, y_1, \ldots, y_m \) there is a unique polynomial \( P_n \) of degree at most \( n \\)

for which:

\[ P_n(x_i) = y_i, \quad i = 0, \ldots, m. \]
Proof: Uniqueness: assume there are two interpolating polynomials \( q_n \) of degree at most \( n \) s.t.

\[
p_n(x_i) = y_i = q_n(y_i), \quad i = 0, \ldots, n.
\]

then \( r(z) = p_n(z) - q_n(z) \) is a poly of degree \( \leq n \)

with \( n+1 > n \) distinct roots \( \Rightarrow r(z) \equiv 0 \) (FTA)

\[
\Rightarrow p_n(z) = q_n(z).
\]

Further, the existence part of the theorem can be work by induction.

Clearly:

- when \( n = 0 \) there is \( p_0(z) \equiv \text{constant} \) s.t. \( x_0 = y_0 \)
- now assume we have a polynomial \( p_k \) of degree \( k \)

\[
\text{s.t.} \quad p_k(x_i) = y_i, \quad i = 0, \ldots, k-1.
\]

Let \( p_{k+1} \) be of the form:

\[
p_{k+1}(x) = p_k(x) + c(x-x_0)(x-x_1) \cdots (x-x_k)
\]

Then clearly:

\[
p_{k+1}(x_i) = p_k(x_i) \quad \text{for } i = 0, \ldots, k
\]

we determine constant \( c \) from:

\[
p_{k+1}(x_{k+1}) = p_k(x_{k+1}) + c(x_{k+1}-x_0)(x_{k+1}-x_1) \cdots (x_{k+1}-x_k)
\]

\[
c = \frac{p_{k+1}(x_{k+1}) - p_k(x_{k+1})}{(x_{k+1}-x_0)(x_{k+1}-x_1) \cdots (x_{k+1}-x_k)}
\]

Note the denominator is non-zero because we assumed all the nodes to be distinct.
Although interpolating polynomial is unique there are several practical ways of writing it. The first one is:

**Newton form of interp poly**

Unravelling the polynomial that is constructed in previous proof we get:

$$P_k(x) = C_0 + C_1(x-x_0) + C_2(x-x_0)(x-x_1) + \ldots + C_k(x-x_0)(x-x_1)\ldots(x-x_{k-1})$$

$$= \sum_{i=0}^{k} C_i \prod_{j=0}^{i-1} (x-x_j)$$

(our convention that $\prod_{j=0}^{i-1} (x-x_j) = 1$)

**Evaluation of Newton's form of interp poly**

This can be done using Horner's algorithm or nested multiplication.

**Idea:** Imagine we would like to evaluate an expression of the kind:

$$u = \sum_{i=0}^{k} C_i \prod_{j=0}^{i-1} d_j = C_0 + C_1 d_0 + C_2 d_0 d_1 + C_3 d_0 d_1 d_2 + \ldots + C_k d_0 d_1 \ldots d_{k-1}$$

$$= \left( (C_k d_{k-1} + C_{k-1}) d_{k-2} + C_{k-2} \right) d_{k-3} + \ldots + C_2 \right) d_0 + C_0$$

To compute we can organize calculations as follows.

- Initialize $\mu_k := C_k$
- $\mu_{k-1} = \mu_k \cdot d_{k-1} + C_{k-1}$
- $\mu_{k-2} = \mu_{k-1} \cdot d_{k-2} + C_{k-2}$
- $\vdots$
- $\mu_0 = \mu_1 \cdot d_0 + C_0$

**Algorithm:**

- $\mu := C_k$
- for $i := k-1$ to 0
  - $\mu := \mu \cdot d_i + c_i$
- return $\mu$
Going back to Newton's form; we would like to evaluate
\[ u = p_k(t) \]

\[
\begin{align*}
    u &= c_k \\
    &\text{for } i = k-1 \ldots 1:0 \\
    &\text{if } i = (t-x_i)u + c_i \\
\end{align*}
\]

New how do we evaluate the coefficients \( c_k \)? From proof we get:

\[
    c_k = \frac{y_k - p_{k-1}(x_k)}{(x_k-x_0)(x_k-x_1)\ldots(x_k-x_{k-1})}
\]

which we can translate into the following algorithm:

\[
\begin{align*}
    c_0 &= y_0 \\
    \text{for } k = 1 \ldots m \text{ do} & \quad \text{evaluates } u = \frac{p_k(x_k)}{ \prod_{i=0}^{k-1} (x_k-x_i) } \\
    & \quad d = \frac{y_k - u}{d(x_k-x_i)} \\
    & \quad c_k = \frac{y_k - u}{d} \\
\end{align*}
\]

Note: this is not a method that is used in practice; it is just a straightforward translation of the proof... More efficient algorithms are divided differences and Neville's algo.
Lagrange form of interpolation polynomial

Here is another form of the interp. poly that is nice for the theory: Again recall we are given \( n+1 \) points \( (x_i, y_i) \), \( i = 0, \ldots, n \), where the \( x_i \) are distinct.

\[
P_n(x) = y_0 \cdot l_0(x) + y_1 \cdot l_1(x) + \ldots + y_n \cdot l_n(x) = \sum_{k=0}^{n} y_k \cdot l_k(x).
\]

Here we want \( l_i(x) \) to be polynomials depending on abscissas \( x_i \) but not on ordinates \( y_i \). The interpolation condition is

\[
P_n(x_i) = y_i = \sum_{k=0}^{n} y_k \cdot l_k(x_i)
\]

⇒ works if we require

\[
l_k(x_i) = \begin{cases} 1 & \text{if } k = i \\ 0 & \text{otherwise} \end{cases}
\]

(Know how)

How can we get such polynomials?

Take for example \( l_0(x) \):

Since \( l_0(x_i) = 0 \) for \( i = 1, \ldots, n \):

\[
l_0(x) = c(x-x_1)(x-x_2)\ldots(x-x_n)
\]

Since \( l_0(x_0) = 1 \) we have:

\[
1 = l_0(x_0) = c(x_0-x_1)(x_0-x_2)\ldots(x_0-x_n)
\]

⇒ \[
l_0(x) = \frac{(x-x_1)(x-x_2)\ldots(x-x_n)}{(x_0-x_1)(x_0-x_2)\ldots(x_0-x_n)}
\]

In general:

\[
l_i(x) = \prod_{\substack{j=0 \ \text{to} \ n \ \text{except} \ i}}^{m} \frac{x-x_j}{x_i-x_j}
\]

\[\Delta \text{ denominator is not zero because modes are different.}\]

"Cardinal polynomials"
Here is yet another (bad!) way of getting interpolation poly:

imagine we wanted the interp poly of the form:

\[ p(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_m x^m \]

interpolation conditions:

\[ p(x_i) = y_i, \quad i = 0, \ldots, m \quad (m+1 \text{ eq}, m+1 \text{ unknown}) \]

can be written in linear system form:

\[
\begin{bmatrix}
1 & x_1 & x_1^2 & x_1^3 & \cdots & x_1^m \\
1 & x_2 & x_2^2 & x_2^3 & \cdots & x_2^m \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
1 & x_m & x_m^2 & x_m^3 & \cdots & x_m^m \\
\end{bmatrix}
\begin{bmatrix}
a_0 \\
a_1 \\
a_2 \\
\vdots \\
a_m \\
\end{bmatrix}
= 
\begin{bmatrix}
y_0 \\
y_1 \\
y_2 \\
\vdots \\
y_m \\
\end{bmatrix}
\]

Vandermonde matrix: by interpolation theorem should be invertible if \( x_i \) are distinct.

However matrix is ill-conditioned. We shall see in chap 6 what this means exactly, but intuitively it means very small errors on the \( y_i \) can lead to huge changes in the \( a_i \).

\( \rightarrow \) not a stable way of computing interp poly!

If we did the same with Lagrange interp poly:

\[
\begin{bmatrix}
l_0(x_0) & l_0(x_1) & \ldots & l_0(x_m) \\
l_1(x_0) & l_1(x_1) & \ldots & l_1(x_m) \\
\vdots & \vdots & \ddots & \vdots \\
l_m(x_0) & l_m(x_1) & \ldots & l_m(x_m) \\
\end{bmatrix}
\begin{bmatrix}
c_0 \\
c_1 \\
\vdots \\
c_m \\
\end{bmatrix}
= 
\begin{bmatrix}
y_0 \\
y_1 \\
y_2 \\
\vdots \\
y_m \\
\end{bmatrix}
\]

Identity matrix: \( \rightarrow \) very stable.
Lagrange interp: good for theory
- good if one has to interpolate many datasets \((x_i, y_i)\) \(i = 0, \ldots, n\), with \(x_i\) the same among datasets (compute \(b_i(x)\) once and reuse)

**Newton form:**
- efficient and easy to evaluate
- accommodate adding new nodes easily
  \(c_0, f_0, \ldots, c_k\) depend on nodes \(x_0, \ldots, x_k\)

Interpolation error (very similar to Taylor's theorem)

**Theorem on polynomial interp error**

Let \(f \in C^{n+1}[a,b]\) and let \(p\) be the poly of degree \(\leq m\) that interpolates \(f\) at \(n+1\) distinct points \(x_0, x_1, \ldots, x_n\) in \([a,b]\). Then \(\forall x \in [a,b], \exists \xi \in (a,b)\) such that

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

**Proof:**
- case \(x = x_i\): theorem is trivial to prove \(0 = 0\).
- case \(x \neq x_i\): \(x\) fixed

Let \(\phi(t) = \prod_{i=0}^{n} (t - x_i)\)

\(\phi = f - p - \lambda w\)

where \(\lambda \in \mathbb{R}\) is chosen s.t. \(\phi(x) = 0\) i.e.,

\[
\lambda = \frac{f(x) - p(x)}{w(x)} \quad \Delta \text{ denominator is nonzero because } \ldots?
\]
\( \phi \in C^{m+1}[a,b] \) and \( \phi \) is zero at points:
\( x_0, x_1, \ldots, x_m, x \)

\( \Rightarrow \) \( \phi \) has at least \( m+2 \) zeros in \( [a,b] \)

\( \phi' \) \hspace{1cm} m+1 \hspace{1cm} \text{by Rolle's theorem} \)

\( \phi'' \) \hspace{1cm} m \hspace{1cm} \)

\( \phi^{(n+1)} \) \hspace{1cm} 1 \hspace{1cm} \)

call this zero \( \xi_2 \)

\( \phi^{(n+1)}(\xi_2) = 0 = \left( \frac{f^{(n+1)}(\xi_2)}{(m+1)!} \frac{f(x) - p(x)}{w(x)} \right) \)

This is the proof!

\( f(x) - p(x) = \frac{1}{(m+1)!} f^{(m+1)}(\xi_2) w(x) \)

\( = \frac{1}{(m+1)!} f^{(m+1)}(\xi_2) \prod_{i=0}^{m} (x - x_i) \)
§3.2 Divided Differences

Let $f$ be a function which is known at $n+1$ distinct nodes $x_0, x_1, \ldots, x_n$. We know there is a unique polynomial $p(x)$ of degree $\leq n$ interpolating $f$ at the nodes $x_i$, i.e.

$$p(x_i) = f(x_i), \quad i = 0, 1, \ldots, n.$$ 

Recall that an interpolating polynomial can be written in Newton form:

$$p(x) = \sum_{j=0}^{n} c_j q_j(x)$$

where

$$q_0(x) = 1$$

$$q_1(x) = (x-x_0)$$

$$q_2(x) = (x-x_0)(x-x_1)$$

$$\vdots$$

$$q_n(x) = (x-x_0)(x-x_1) \cdots (x-x_n)$$

Also recall that

$$c_0 \text{ is found s.t. } \sum_{j=0}^{n} c_j q_j(x) \text{ interp. } f \text{ at nodes } x_0,$$

$$c_1 \quad \sum_{j=0}^{n} c_j q_j(x) \quad x_0, x_1$$

$$\vdots$$

$$c_k \quad \sum_{j=0}^{n} c_j q_j(x) \quad x_0, x_1, \ldots, x_k$$

In general $c_k$ depends only on nodes $x_0, \ldots, x_k$ and $f(x_0), \ldots, f(x_k)$. Because of this dependency we denote:

$$C_n = \left[ f(x_0, x_1, x_2, \ldots, x_n) \right]$$
In other words
\[ f(x_0, x_1, \ldots, x_n) = \text{coeff of } g_m \text{ in interp poly} \]

But \( g_m(x) = (x-x_0)(x-x_1)\ldots(x-x_n) = x^m + \text{lower order terms} \)

\[ f(x_0, x_1, \ldots, x_n) = \text{coeff in front of } x^m \text{ in poly interp } f \]

at points \( x_0, x_1, \ldots, x_n \).

\[ f(x_0, \ldots, x_n) \text{ is called a divided difference of } f. \]
The reason comes from figuring out the first few divided differences:

\[ f [x_0] = f(x_0) \] (since \( f[x_0] = \text{coeff in front of constant interpolating } f \) at \( x_0 \))

Now the polynomial interpolating \( f \) at \( x_0, x_1 \), \( \ldots \), \( x_n \) is:

\[ p(x) = f(x_0) + \frac{f(x_1) - f(x_0)}{x_1 - x_0} (x - x_0) \]

\[ = f [x_0, x_1] = \frac{f(x_1) - f(x_0)}{x_1 - x_0} \]

\( = \text{divided difference monomial basis origin} \)

In general:

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

\[ \text{is the interpolating poly of } f \text{ at } x_0, \ldots, x_n \]

\[ \text{Note: if we would like to interpolate only on } x_0, \ldots, x_m, \]
\[ \text{with } m \leq n, \text{ the interp poly will be the sum } (*) \text{ truncated up to } k = m. \]
Why go through this trouble? Because of the following nice relationship we can evaluate divided differences in an efficient manner:

**Theorem (Higher order divided differences)**

\[
f[x_0, \ldots, x_n] = \frac{f[x_1, x_2, \ldots, x_n] - f[x_0, x_1, \ldots, x_{n-1}]}{x_n - x_0}
\]

**Proof:** Let \( p_k = \text{poly of degree } \leq k \) interpolating \( f \) at \( x_0, x_1, \ldots, x_k \).

\( q = \text{poly of degree } \leq n-1 \) interpolating \( f \) at \( x_1, x_2, \ldots, x_n \).

Then:

\[
p_m(x) = q(x) + \frac{x - x_n}{x_n - x_0} \left[ q(x) - p_{m-1}(x) \right]
\]

Why? First both poly have degree \( \leq m \) and:

for \( i = 1, \ldots, n-1 \),

\[
q(x_i) + \frac{x_i - x_n}{x_n - x_0} \left[ q(x_i) - p_{m-1}(x_i) \right] = \frac{f(x_i)}{f(x_i)}
\]

\[
f(x_i) = p_m(x_i)
\]

also:

\[
q(x_0) + \frac{x_0 - x_n}{x_n - x_0} \left[ q(x_0) - p_{m-1}(x_0) \right] = p_{n-1}(x_0)
\]

\[
- p_m(x_0) = f(x_0)
\]

\[
q(x_n) + 0 \left[ \ldots \right] = f(x_n) = p_m(x_n)
\]

Thus the poly on both sides of (4) match at \( m+1 \) points

\( \Rightarrow \) they must be equal!
By matching coeff of degree n we get:

\[ f_n(x) = x^n f[x_0, \ldots, x_n] + \text{l.o.t} \]
\[ = \text{l.o.t.} + \frac{x^n}{x_n - x_0} \left[ f[x_1, x_2, \ldots, x_n] - f[x_0, x_1, \ldots, x_n] \right] \]

Q.E.D.

Since we are free to choose the interpolation points, we get:

\[ f[x_i, x_{i+1}, \ldots, x_{i+j}] = \frac{f[x_{i+1}, x_{i+2}, \ldots, x_{i+j}] - f[x_i, x_{i+1}, \ldots, x_{i+j-1}]}{x_{i+j} - x_i} \]

\[ f[x_i] \equiv \text{div. diff of order 0} \]
\[ f[x_i, x_{i+1}] \equiv \text{div. diff of order 1} \]
\[ f[x_i, x_{i+1}, x_{i+2}] \equiv \text{div. diff of order 2} \]

etc...

The computation of divided differences is usually organized as a table:

<table>
<thead>
<tr>
<th>( x_0 )</th>
<th>( f[x_0] )</th>
<th>( f[x_0, x_1] )</th>
<th>( f[x_0, x_1, x_2] )</th>
<th>( f[x_0, x_1, x_2, x_3] )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_1 )</td>
<td>( f[x_1] )</td>
<td>( f[x_1, x_2] )</td>
<td>( f[x_1, x_2, x_3] )</td>
<td>( f[x_1, x_2, x_3, x_4] )</td>
</tr>
<tr>
<td>( x_2 )</td>
<td>( f[x_2] )</td>
<td>( f[x_2, x_3] )</td>
<td>( f[x_2, x_3, x_4] )</td>
<td>( f[x_2, x_3, x_4, x_5] )</td>
</tr>
<tr>
<td>( x_3 )</td>
<td>( f[x_3] )</td>
<td>( f[x_3, x_4] )</td>
<td>( f[x_3, x_4, x_5] )</td>
<td>( f[x_3, x_4, x_5, x_6] )</td>
</tr>
</tbody>
</table>

\( f \) \text{ from data} \quad 1^{st} \text{order} \quad 2^{nd} \text{order} \quad 3^{rd} \text{order}

Each column corresponds to div diff of the same order and the entries in columns after the dividing line can be computed from entries of preceding column by applying divided differences theorem.