Numerical Analysis and Computing

Lecture Notes #04 — Solutions of Equations in One Variable, Interpolation and Polynomial Approximation — Accelerating Convergence; Zeros of Polynomials; Deflation; Müller's Method; Lagrange Polynomials; Neville's Method

Joe Mahaffy,
⟨mahaffy@math.sdsu.edu⟩

Department of Mathematics
Dynamical Systems Group
Computational Sciences Research Center
San Diego State University
San Diego, CA 92182-7720

http://www-rohan.sdsu.edu/~jmahaffy

Outline

Introduction

"It is rare to have the luxury of quadratic convergence."
(Burden-Faires, p.83)

There are a number of methods for squeezing faster convergence out of an **already computed sequence** of numbers.

We here explore one method which seems the have been around since the beginning of numerical analysis... **Aitken's Δ^2 method**. It can be used to accelerate convergence of a sequence that is linearly convergent, regardless of its origin or application.

A review of modern extrapolation methods can be found in:
"Practical Extrapolation Methods: Theory and Applications," Avram Sidi, Number 10 in Cambridge Monographs on Applied and Computational Mathematics, Cambridge University Press, June 2003. ISBN: 0-521-66159-5

Recall: Convergence of a Sequence

Definition

Suppose the sequence $\{p_n\}_{n=0}^{\infty}$ converges to p, with $p_n \neq p$ for all n. If positive constants λ and α exists with

$$\lim_{n\to\infty} \frac{|p_{n+1} - p|}{|p_n - p|^{\alpha}} = \lambda$$

then $\{p_n\}_{n=0}^{\infty}$ **converges to p of order α, with asymptotic error constant λ.**

Linear convergence means that $\alpha = 1$, and $|\lambda| < 1$.

Aitken's Δ^2 Method — I/II

Assume $\{p_n\}_{n=0}^{\infty}$ is a **linearly convergent sequence** with limit p.
Further, assume we are far out into the tail of the sequence (n large), and the signs of the successive errors agree, *i.e.*

$$\mathrm{sign}(p_n - p) = \mathrm{sign}(p_{n+1} - p) = \mathrm{sign}(p_{n+2} - p) = \dots$$

and that

$$\frac{p_{n+2} - p}{p_{n+1} - p} \approx \frac{p_{n+1} - p}{p_n - p} \approx \lambda \quad \text{(the asymptotic limit)}$$

This would indicate

$$(p_{n+1} - p)^2 \approx (p_{n+2} - p)(p_n - p)$$

$$p_{n+1}^2 - 2p_{n+1}\mathbf{p} + \mathbf{p^2} \approx p_{n+2}p_n - (p_{n+2} + p_n)\mathbf{p} + \mathbf{p^2}$$

We solve for p and get...

Aitken's Δ^2 Method — II/II

We solve for p and get...

$$p \approx \frac{p_{n+2}p_n - p_{n+1}^2}{p_{n+2} - 2p_{n+1} + p_n}$$

A little bit of algebraic manipulation put this into the classical Aitken form:

$$\hat{p}_n = p = p_n - \frac{(p_{n+1} - p_n)^2}{p_{n+2} - 2p_{n+1} + p_n}$$

Aitken's Δ^2 Method is based on the assumption that the $\hat{p}_n$ we compute from p_{n+2}, p_{n+1} and p_n is a better approximation to the real limit p.

The analysis needed to **prove** this is beyond the scope of this class, see *e.g.* Sidi's book.

Aitken's Δ^2 Method — The Recipe

Given a sequence finite $\{p_n\}_{n=0}^{N}$ or infinite $\{q_n\}_{n=0}^{\infty}$ sequence which converges linearly to some limit.
Define the new sequences

$$\hat{p}_n = p_n - \frac{(p_{n+1} - p_n)^2}{p_{n+2} - 2p_{n+1} + p_n}, \quad n = 0, 1, \dots, N-2$$

or

$$\hat{q}_n = q_n - \frac{(q_{n+1} - q_n)^2}{q_{n+2} - 2q_{n+1} + q_n}, \quad n = 0, 1, \dots, \infty$$

The numerator is a *forward difference* squared, while the denominator is a second order *central difference*.

Aitken's Δ^2 Method — Example

Consider the sequence $\{p_n\}_{n=0}^{\infty}$, where the sequence is generated by the fixed point iteration $p_{n+1} = \cos(p_n)$, $p_0 = 0$.

Iteration	$\mathbf{p_n}$	$\hat{\mathbf{p}}_\mathbf{n}$
0	0.000000000000000	**0**.685073357326045
1	1.000000000000000	**0.7**28010361467617
2	**0**.540302305868140	**0.73**3665164585231
3	**0**.857553215846393	**0.73**6906294340474
4	**0**.654289790497779	**0.73**8050421371664
5	**0.7**93480358742566	**0.73**8636096881655
6	**0.7**01368773622757	**0.73**8876582817136
7	**0.7**63959682900654	**0.73**8992243027034
8	**0.7**22102425026708	**0.7390**42511328159
9	**0.7**50417761763761	**0.7390**65949599941
10	**0.73**1404042422510	**0.7390**76383318956
11	**0.7**44237354900557	**0.73908**1177259563*
12	**0.73**5604740436347	**0.73908**3333909684*

Note: Bold digits are correct; $\hat{p}_{11}$ needs p_{13}, and $\hat{p}_{12}$ additionally needs p_{14}.

Faster Convergence for "Aitken-Sequences"

Theorem (Convergence of Aitken-Δ^2-Sequences)

Suppose $\{p_n\}_{n=0}^{\infty}$ is a sequence that converges linearly to the limit p, and for n large enough we have $(p_n - p)(p_{n+1} - p) > 0$. Then the Aitken-accelerated sequence $\{\hat{p}_n\}_{n=0}^{\infty}$ converges fast to p in the sense that

$$\lim_{n\to\infty} \left[\frac{\hat{p}_n - p}{p_n - p} \right] = 0.$$

We can combine Aitken's method with fixed-point iteration in order to get a "fixed-point iteration on steroids."

Steffensen's Method: Fixed-Point Iteration on Steroids

Suppose we have a fixed point iteration:

$$p_0, \quad p_1 = g(p_0), \quad p_2 = g(p_1), \quad \ldots$$

Once we have p_0, p_1 and p_2, we can compute

$$\hat{p}_0 = p_0 - \frac{(p_1 - p_0)^2}{p_2 - 2p_1 + p_0}$$

At this point we "restart" the fixed point iteration with $p_0 = \hat{p}_0$, *e.g.*

$$p_3 = \hat{p}_0, \quad p_4 = g(p_3), \quad p_5 = g(p_4),$$

and compute

$$\hat{p}_3 = p_3 - \frac{(p_4 - p_3)^2}{p_5 - 2p_4 + p_3}$$

Steffensen's Method: The Quadratic, g-g-A, Waltz! Quadratic Convergence

Algorithm: Steffensen's Method

`Input:` Initial approximation p_0; tolerance TOL; maximum number of iterations N_0.
`Output:` Approximate solution p, or failure message.

`1.` Set $i = 1$
`2.` While $i \leq N_0$ do 3--6
`3*` Set $p_1 = g(p_0)$, $p_2 = g(p_1)$,
  $p = p_0 - (p_1 - p_0)^2/(p_2 - 2p_1 + p_0)$
`4.` If $|p - p_0| < TOL$ then
`4a.` output p
`4b.` stop program
`5.` Set $i = i + 1$
`6.` Set $p_0 = p$
`7.` Output: "Failure after N_0 iterations."

Steffensen's Method: Potential Breakage

3* If at some point $p_2 - 2p_1 + p_0 = 0$ (which appears in the denominator), then we stop and select the current value of p_2 as our approximate answer.

Both Newton's and Steffensen's methods give quadratic convergence. In Newton's method we compute one function value and one derivative in each iteration. In Steffensen's method we have two function evaluations and a more complicated algebraic expression in each iteration, **but no derivative**. It looks like we got something for (almost) nothing. **However,** in order the guarantee quadratic convergence for Steffensen's method, the fixed point function g must be 3 times continuously differentiable, *e.g.* $f \in C^3[a, b]$, (*see* theorem-2.14 in Burden-Faires). Newton's method "only" requires $f \in C^2[a, b]$ (BF Theorem-2.5).

Steffensen's Method: Example

Below we compare a Fixed Point iteration, Newton's Method, and Steffensen's Method for solving:

$$f(x) = x^3 + 4x^2 - 10 = 0$$

or alternately,

$$p_{n+1} = g(p_n) = \sqrt{\frac{10}{p_n + 4}}$$

Steffensen's Method: Example

Fixed Point Iteration

i	p_n	$g(p_n)$
0	1.50000	1.34840
1	1.34840	1.36738
2	1.36738	1.36496
3	1.36496	1.3652
4	1.36526	1.36523
5	1.36523	1.36523

Newton's Method

i	x_n	$f(x_n)$
0	1.50000	1.51600e-01
1	1.36495	-3.11226e-04
2	1.36523	-1.35587e-09

Steffensen's Method

i	p_n	p_1	p_2	p	$\lvert p - p_2 \rvert$
0	1.50000	1.34840	1.36738	1.36527	3.96903e-05
1	1.36527	1.36523	1.36523	1.36523	2.80531e-12

Zeros of Polynomials

Definition: Degree of a Polynomial

A **polynomial of degree** n has the form

$$P(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0, \quad a_n \neq 0$$

where the a_i's are constants (either real, or complex) called the **coefficients** of P.

Why look at polynomials? — We'll be looking at the problem $P(x) = 0$ (*i.e.* $f(x) = 0$ for a special class of functions.)

Polynomials are the basis for many approximation methods, hence being able to solve polynomial equations fast is valuable.

We'd like to use Newton's method, so we need to compute $P(x)$ and $P'(x)$ as efficiently as possible.

Fundamentals

Theorem (The Fundamental Theorem of Algebra)

If $P(x)$ is a polynomial of degree $n \geq 1$ with real or complex coefficients, then $P(x) = 0$ has at least one (possibly complex) root.

The proof is surprisingly(?) difficult and requires understanding of complex analysis... We leave it as an exercise for the motivated student!

Key Consequences of the Fundamental Theorem of Algebra

Corollary

If $P(x)$ is a polynomial of degree $n \geq 1$ with real or complex coefficients then there exists unique constants x_1, x_2, $\ldots$, x_k (possibly complex) and unique positive integers m_1, m_2, $\ldots$, m_k such that $\sum_{i=1}^{k} m_i = n$ and

$$P(x) = a_n(x - x_1)^{m_1}(x - x_2)^{m_2} \cdots (x - x_k)^{m_k}$$

— The collection of zeros is unique.

— m_i are the multiplicities of the individual zeros.

— A polynomial of degree n has exactly n zeros, counting multiplicity.

Key Consequences of the Fundamental Theorem of Algebra

Corollary

Let $P(x)$ and $Q(x)$ be polynomials of degree at most n. If x_1, x_2, $\ldots$, x_k, with $k > n$ are **distinct** numbers with $P(x_i) = Q(x_i)$ for $i = 1, 2, \ldots, k$, then $P(x) = Q(x)$ for all values of x.

— If two polynomials of degree n agree at at least $(n + 1)$ points, then they must be the same.

Horner's Method: Evaluating Polynomials Quickly

Let
$$P(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$$
We want an efficient method to solve

$$P(x) = 0.$$

In 1819, William Horner developed an efficient algorithm with n multiplications and n additions to evaluate $P(x_0)$.
Technique is called Horner's Method or Synthetic Division.

Horner's Method: Evaluating Polynomials Quickly

Assume
$$P(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$$
If we are looking to evaluate $P(x_0)$ for any x_0, let
$$b_n = a_n, \quad b_k = a_k + b_{k+1}x_0, \quad k = (n-1), (n-2), \ldots, 1, 0,$$
then $b_0 = P(x_0)$. We have only used n multiplications and n additions.
At the same time we have computed the decomposition
$$P(x) = (x - x_0)Q(x) + b_0,$$
where
$$Q(x) = \sum_{k=1}^{n-1} b_{k+1}x^k.$$
If $b_0 = 0$, then x_0 is a root of $P(x)$.

Horner's Method: Evaluating Polynomials Quickly

Huh?!? Where did the expression come from? — Consider

$$\begin{aligned} P(x) &= a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0 \\ &= (a_n x^{n-1} + a_{n-1}x^{n-2} + \cdots + a_1)x + a_0 \\ &= ((a_n x^{n-2} + a_{n-1}x^{n-3} + \cdots)x + a_1)x + a_0 \\ &= \underbrace{(\ldots((}_{n-1}\underbrace{a_n x + a_{n-1}}_{b_{n-1}})x + \cdots)x + a_1)x + a_0 \end{aligned}$$

Horner's method is "simply" the computation of this parenthesized expression from the inside-out...

Horner's Method: Evaluating Polynomials Quickly

Now, if we need to compute $P'(x_0)$ we have

$$P'(x)\Big|_{x=x_0} = (x-x_0)Q'(x) + Q(x)\Big|_{x=x_0} = Q(x_0)$$

Which we can compute (again using Horner's method) in $(n-1)$ multiplications and $(n-1)$ additions.

Proof? We really ought to prove that Horner's method works. It basically boils down to lots of algebra which shows that the coefficients of $P(x)$ and $(x-x_0)Q(x)+b_0$ are the same...

A couple of examples may be more instructive...

Example#1: Horner's Method

For $P(x) = x^4 - x^3 + x^2 + x - 1$, compute $P(5)$:

$x_0 = 5$	$a_4 = 1$	$a_3 = -1$	$a_2 = 1$	$a_1 = 1$	$a_0 = -1$
		$b_4x_0 = 5$	$b_3x_0 = 20$	$b_2x_0 = 105$	$\mathbf{b_1x_0 = 530}$
	$b_4 = 1$	$b_3 = 4$	$b_2 = 21$	$b_1 = 106$	$\mathbf{b_0 = 529}$

Hence, $\mathbf{P(5) = 529}$, and

$$P(x) = (x-5)(x^3 + 4x^2 + 21x + 106) + 529$$

Similarly we get $P'(5) = Q(5) = 436$

$x_0 = 5$	$a_3 = 1$	$a_2 = 4$	$a_1 = 21$	$a_0 = 106$
		$b_3x_0 = 5$	$b_2x_0 = 45$	$b_1x_0 = 330$
	$b_3 = 1$	$b_2 = 9$	$b_1 = 66$	$b_0 = \mathbf{436}$

Algorithm: Horner's Method

Algorithm: Horner's Method

`Input:` Degree n; coefficients $a_0, a_1, \ldots, a_n$; x_0

`Output:` $y = P(x_0)$, $z = P'(x_0)$.

1. Set $y = a_n$, $z = a_n$
2. For $j = (n-1), (n-2), \ldots, 1$

   Set $y = x_0 y + a_j$, $z = x_0 z + y$
3. Set $y = x_0 y + a_0$
4. Output (y, z)
5. End program

Deflation — Finding All the Zeros of a Polynomial

If we are solving our current favorite problem

$$P(x) = 0, \quad P(x) \text{ a polynomial of degree } n$$

and we are using Horner's method of computing $P(x_i)$ and $P'(x_i)$, then after N iterations, x_N is an approximation to one of the roots of $P(x) = 0$.

We have

$$P(x) = (x - x_N)Q(x) + b_0, \quad b_0 \approx 0$$

Let $\hat{r}_1 = x_N$ be the first root, and $Q_1(x) = Q(x)$.

We can now find a second root by applying Newton's method to $Q_1(x)$.

Deflation — Finding All the Zeros of a Polynomial

After some number of iterations of Newton's method we have

$$Q_1(x) = (x - \hat{r}_2)Q_2(x) + b_0^{(2)}, \quad b_0^{(2)} \approx 0$$

If $P(x)$ is an n^{th}-degree polynomial with n real roots, we can apply this procedure $(n-2)$ times to find $(n-2)$ approximate zeros of $P(x)$: $\hat{r}_1, \hat{r}_2, \ldots, \hat{r}_{n-2}$, and a quadratic factor $Q_{n-2}(x)$.

At this point we can solve $Q_{n-2}(x) = 0$ using the quadratic formula, and we have n roots of $P(x) = 0$.

This procedure is called **Deflation**.

Quality of Deflation

Now, the big question is **"are the approximate roots $\hat{r}_1, \hat{r}_2, \ldots, \hat{r}_n$ good approximations of the roots of $P(x)$???"**

Unfortunately, sometimes, **no**.

In each step we solve the equation to some tolerance, *i.e.*

$$|b_0^{(k)}| < tol$$

Even though we may solve to a tight tolerance (10^{-8}), the errors accumulate and the inaccuracies increase iteration-by-iteration...

Question: Is deflation therefore useless???

Improving the Accuracy of Deflation

The problem with deflation is that the zeros of $Q_k(x)$ are not good representatives of the zeros of $P(x)$, especially for high k's.

As k increases, the quality of the root $\hat{r}_k$ decreases.

Maybe there is a way to get all the zeros with the same quality?

The idea is quite simple... in each step of deflation, instead of just accepting $\hat{r}_k$ as a root of $P(x)$, we re-run Newton's method on the full polynomial $P(x)$, with $\hat{r}_k$ as the starting point — a couple of Newton iterations should quickly converge to the root of the full polynomial.

Improved Deflation — Algorithm Outline

Algorithm Outline: Improved Deflation

1. Apply Newton's method to $P(x) \quad \rightarrow \hat{\mathbf{r}}_1,\ \mathbf{Q_1(x)}$.
2. For $k = 2, 3, \dots, (n-2)$ do 3--4
3. Apply Newton's method to $\mathbf{Q_{k-1}} \quad \rightarrow \hat{\mathbf{r}}^*_{\mathbf{k}},\ \mathbf{Q^*_k(x)}$.
4. Apply Newton's method to $\mathbf{P(x)}$ with $\hat{\mathbf{r}}^*_{\mathbf{k}}$ as the initial point $\rightarrow \hat{\mathbf{r}}_{\mathbf{k}}$
   Apply Horner's method to $\mathbf{Q_{k-1}(x)}$ with $\mathbf{x} = \hat{\mathbf{r}}_{\mathbf{k}} \quad \rightarrow \mathbf{Q_k(x)}$
5. Use the quadratic formula on $\mathbf{Q_{n-2}(x)}$ to get the two remaining roots.

Note: "Inside" Newton's method, the evaluations of polynomials and their derivatives are also performed using Horner's method.

Deflation & Improvement — Wilkinson Polynomials

The Wilkinson Polynomials

$$P_n^W(x) = \prod_{k=1}^{n} (x-k)$$

have the roots $\{1, 2, \dots, n\}$, but provide surprisingly tough numerical root-finding problems. *(Additional details in Math 543.)*

In the next few slides we show the results of Deflation and Improved Deflation applied to Wilkinson polynomials of degree 9, 10, 12, and 13.

Deflation & Improvement — $P_9^W(x)$ and $P_{10}^W(x)$

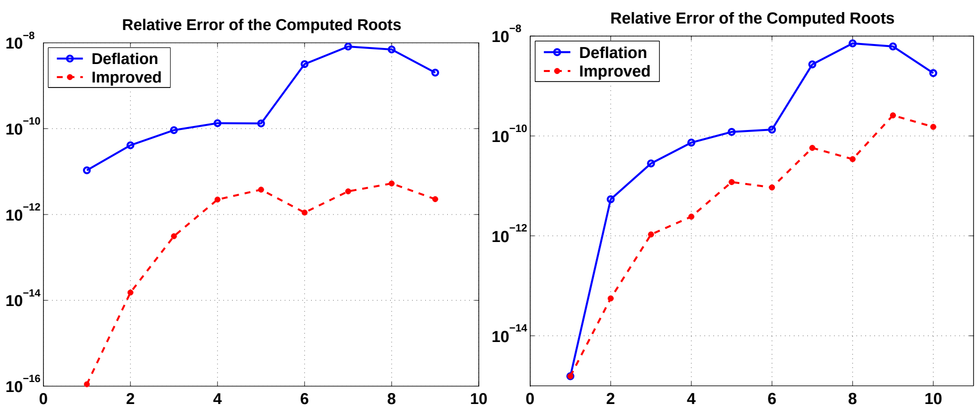


Figure: [LEFT] The result of the two algorithms on the Wilkinson polynomial of degree 9; in this case the roots are computed so that $|b_0^{(k)}| < 10^{-6}$. [RIGHT] The result of the two algorithms on the Wilkinson polynomial of degree 10; in this case the roots are computed so that $|b_0^{(k)}| < 10^{-6}$. In both cases the lower line corresponds to **improved deflation** and we see that we get an improvement in the relative error of several orders of magnitude.

Deflation & Improvement — $P_{12}^W(x)$ and $P_{13}^W(x)$

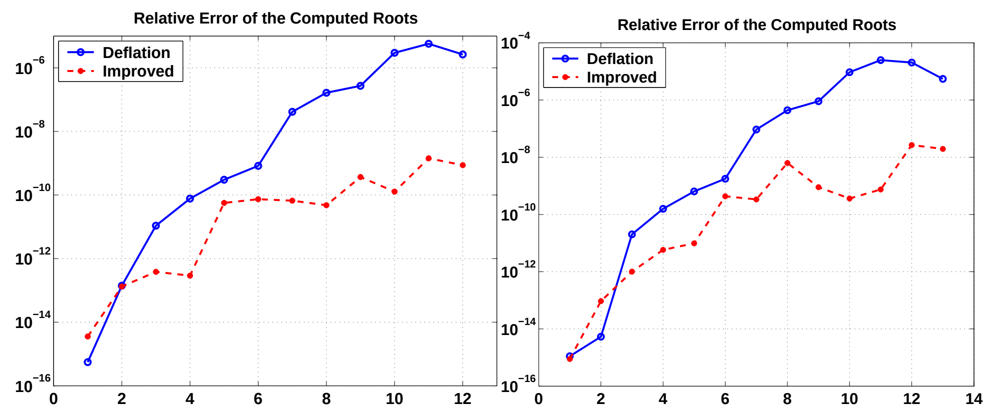


Figure: [LEFT] The result of the two algorithms on the Wilkinson polynomial of degree 12; in this case the roots are computed so that $|b_0^{(k)}| < 10^{-4}$. [RIGHT] The result of the two algorithms on the Wilkinson polynomial of degree 13; in this case the roots are computed so that $|b_0^{(k)}| < 10^{-3}$. In both cases the lower line corresponds to **improved deflation** and we see that we get an improvement in the relative error of several orders of magnitude.

What About Complex Roots???

One interesting / annoying feature of polynomials with real coefficients is that they may have complex roots, *e.g.* $P(x) = x^2 + 1$ has the roots $\{-i, i\}$. Where by definition $i = \sqrt{-1}$.

If the initial approximation given to Newton's method is real, all the successive iterates will be real... which means we will not find complex roots.

One way to overcome this is to start with a complex initial approximation and do all the computations in complex arithmetic.

Another solution is **Müller's Method**...

Müller's Method

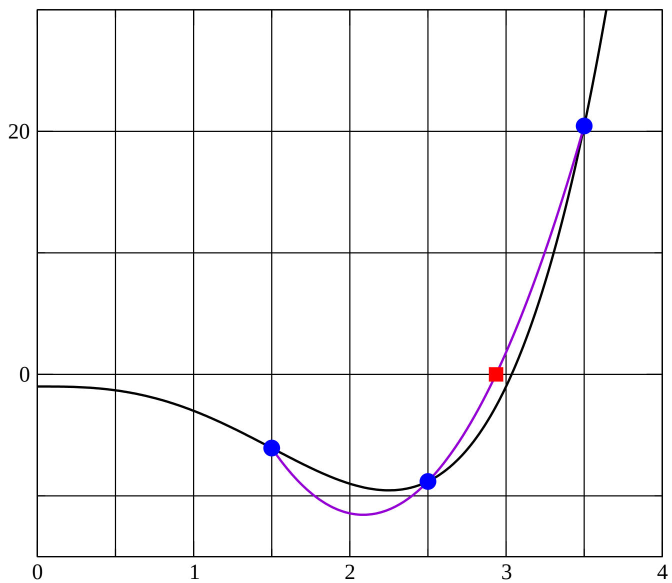
Müller's method is an extension of the Secant method...

Recall that the secant method uses two points x_k and x_{k-1} and the function values in those two points $f(x_k)$ and $f(x_{k-1})$. The zero-crossing of the linear interpolant (the secant line) is used as the next iterate x_{k+1}.

Müller's method takes the next logical step: it uses **three points**: x_k, x_{k-1} and x_{k-2}, the function values in those points $f(x_k)$, $f(x_{k-1})$ and $f(x_{k-2})$; a second degree polynomial fitting these three points is found, and its zero-crossing is the next iterate x_{k+1}.

Next slide: $f(x) = x^4 - 3x^3 - 1$, $x_{k-2} = 1.5$, $x_{k-1} = 2.5$, $x_k = 3.5$.

Müller's Method — Illustration $f(x) = x^4 - 3x^3 - 1$



Müller's Method — Fitting the Quadratic Polynomial

We consider the quadratic polynomial

$$m(x) = a(x - x_k)^2 + b(x - x_k) + c$$

at the three fitting points we get

$$\begin{aligned} f(x_{k-2}) &= a(x_{k-2} - x_k)^2 + b(x_{k-2} - x_k) + c \\ f(x_{k-1}) &= a(x_{k-1} - x_k)^2 + b(x_{k-1} - x_k) + c \\ f(x_k) &= c \end{aligned}$$

We can solve for a, b, and c:

$$\begin{aligned} a &= \frac{(x_{k-1} - x_k)(f(x_{k-2}) - f(x_k)) - (x_{k-2} - x_k)(f(x_{k-1}) - f(x_k))}{(x_{k-2} - x_k)(x_{k-1} - x_k)(x_{k-2} - x_{k-1})} \\ b &= \frac{(x_{k-2} - x_k)^2(f(x_{k-1}) - f(x_k)) - (x_{k-1} - x_k)^2(f(x_{k-2}) - f(x_k))}{(x_{k-2} - x_k)(x_{k-1} - x_k)(x_{k-2} - x_{k-1})} \\ c &= f(x_k) \end{aligned}$$

Müller's Method — Identifying the Zero

We now have a quadratic equation for $(x - x_k)$ which gives us two possibilities for x_{k+1}:

$$x_{k+1} - x_k = \frac{-2c}{b \pm \sqrt{b^2 - 4ac}}$$

In Müller's method we select

$$x_{k+1} = x_k - \frac{2c}{b + \mathrm{sign}(b)\sqrt{b^2 - 4ac}}$$

we are maximizing the (absolute) size of the denominator, hence we select the root closest to x_k.

Note that if $b^2 - 4ac < 0$ then we automatically get complex roots.

Accelerating Convergence
Zeros of Polynomials
Deflation, Müller's Method
Polynomial Approximation
Deflation: Finding All the Zeros of a Polynomial
Müller's Method — Finding Complex Roots

Müller's Method — Algorithm

Algorithm: Müller's Method

`Input:` x_0, x_1, x_2; tolerance tol; max iterations N_0

`Output:` Approximate solution p, or failure message.

1. Set $h_1 = (x_1 - x_0)$, $h_2 = (x_2 - x_1)$, $\delta_1 = [f(x_1) - f(x_0)]/h_1$, $\delta_2 = [f(x_2) - f(x_1)]/h_2$, $d = (\delta_2 - \delta_1)/(h_2 + h_1)$, $j = 3$.
2. While $j \leq N_0$ do 3--7
3. $\quad b = \delta_2 + h_2 d$, $D = \sqrt{b^2 - 4f(x_2)d}$ `complex?`
4. $\quad$ If $|b - D| < |b + D|$ then set $E = b + D$ else set $E = b - D$
5. $\quad$ Set $h = -2f(x_2)/E$, $p = x_2 + h$
6. $\quad$ If $|h| < tol$ then `output p; stop program`
7. $\quad$ Set $x_0 = x_1$, $x_1 = x_2$, $x_2 = p$, $h_1 = (x_1 - x_0)$, $h_2 = (x_2 - x_1)$, $\delta_1 = [f(x_1) - f(x_0)]/h_1$, $\delta_2 = [f(x_2) - f(x_1)]/h_2$, $d = (\delta_2 - \delta_1)/(h_2 + h_1)$, $j = j + 1$
8. `output` — "Müller's Method failed after N_0 iterations."

Accelerating Convergence
Zeros of Polynomials
Deflation, Müller's Method
Polynomial Approximation
Deflation: Finding All the Zeros of a Polynomial
Müller's Method — Finding Complex Roots

Now We Know "Everything" About Solving $f(x) = 0$!?

Let's recap... Things to remember...

The relation between **root finding** ($f(x) = 0$) and **fixed point** ($g(x) = x$).

Key algorithms for root finding: Bisection, Secant Method, and **Newton's Method**. — Know what they are (the updates), how to start (one or two points? bracketing or not bracketing the root?), can the method break, can breakage be fixed? Convergence properties.

Also, know the mechanics of the Regula Falsi method, and understand why it can run into trouble.

Fixed point iteration: Under what conditions do FP-iteration converge for all starting values in the interval?

Accelerating Convergence
Zeros of Polynomials
Deflation, Müller's Method
Polynomial Approximation
Deflation: Finding All the Zeros of a Polynomial
Müller's Method — Finding Complex Roots

Recap, continued...

Basic error analysis: order α, asymptotic error constant λ. — Which one has the most impact on convergence? Convergence rate for general fixed point iterations?

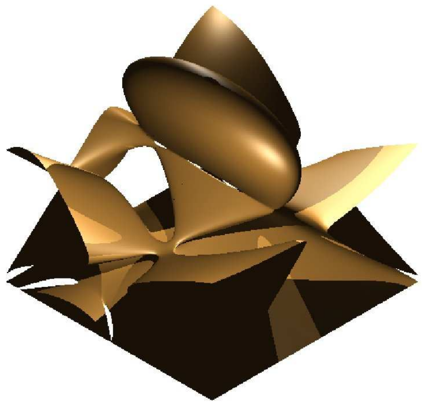
Multiplicity of zeros: What does it mean? How do we use this knowledge to "help" Newton's method when we're looking for a zero of high multiplicity?

Convergence acceleration: Aitken's Δ^2-method. Steffensen's Method.

Zeros of polynomials: Horner's method, Deflation (with improvement), Müller's method.

New Favorite Problem:

Interpolation and Polynomial Approximation



Weierstrass Approximation Theorem

The following theorem is the basis for polynomial approximation:

Theorem (Weierstrass Approximation Theorem)

Suppose $f \in C[a,b]$. *Then* $\forall \epsilon > 0$ $\exists$ *a polynomial* $P(x)$: $|f(x) - P(x)| < \epsilon$, $\forall x \in [a,b]$.

Note: The bound is *uniform*, *i.e.* valid for all x in the interval.

Note: The theorem says nothing about how to find the polynomial, or about its order.

Illustrated: Weierstrass Approximation Theorem

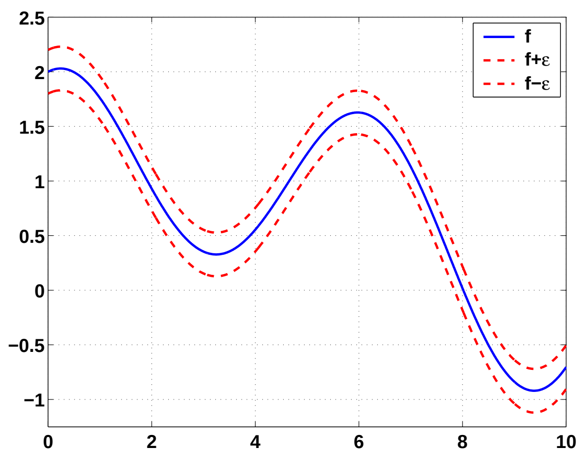


Figure: Weierstrass approximation Theorem guarantees that we (maybe with substantial work) can find a polynomial which fits into the "tube" around the function f, no matter how thin we make the tube.

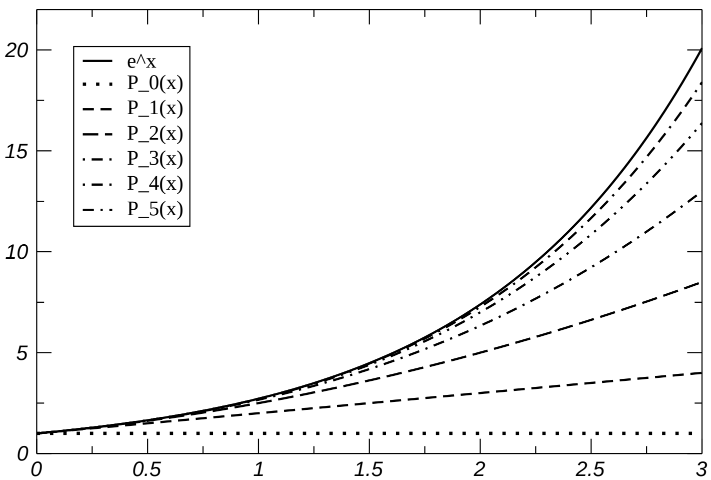
Candidates: the Taylor Polynomials???

Natural Question:
Are our old friends, the Taylor Polynomials, good candidates for polynomial interpolation?

Answer:
No. The Taylor expansion works very hard to be accurate in the neighborhood of *one point*. But we want to fit data at many points (in an extended interval).

[Next slide: The approximation is great near the expansion point $x_0 = 0$, but get progressively worse at we get further away from the point, even for the higher degree approximations.]

Taylor Approximation of e^x on the Interval [0, 3]



Taylor Approximation of $f(x) = 1/x$ about $x = 1$

Let

$$f(x) = \frac{1}{x}$$

The Taylor expansion about $x_0 = 1$ is

$$P_n(x) = \sum_{k=0}^{n} (-1)^k (x-1)^k.$$

Note: $f(3) = 1/3$, but $P_n(3)$ satisfies:

n	0	1	2	3	4	5	6
$P_n(3)$	1	-1	3	-5	11	-21	43

The Taylor's series only converges $|x-1| < 1$ by the ratio test (a geometric series). Thus, not valid for $x = 3$.

Lookahead: Polynomial Approximation

Clearly, Taylor polynomials are not well suited for approximating a function over an **extended** interval.

We are going to look at the following:

- Lagrange polynomials — Neville's Method. [This Lecture]
- Newton's divided differences.
- Hermite interpolation.
- Cubic splines — Piecewise polynomial approximation.
- (Parametric curves)
- (Bézier curves — used in *e.g.* computer graphics)

Interpolation: Lagrange Polynomials

Idea: Instead of working hard at *one point*, we will prescribe a number of points through which the polynomial must pass.

As warm-up we will define a function that passes through the points $(x_0, f(x_0))$ and $(x_1, f(x_1))$. First, lets define

$$L_0(x) = \frac{x - x_1}{x_0 - x_1}, \quad L_1(x) = \frac{x - x_0}{x_1 - x_0},$$

and then define the interpolating polynomial

$$P(x) = L_0(x) f(x_0) + L_1(x) f(x_1),$$

then $P(x_0) = f(x_0)$, and $P(x_1) = f(x_1)$.

– **$P(x)$ is the unique linear polynomial passing through $(x_0, f(x_0))$ and $(x_1, f(x_1))$.**

An n-degree polynomial passing through $n+1$ points

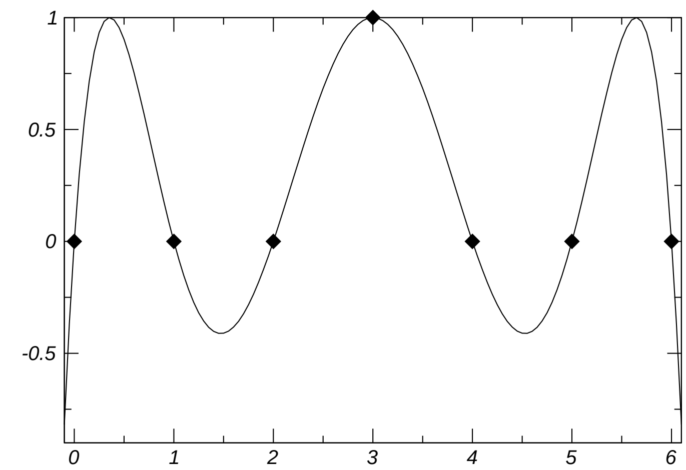
We are going to construct a polynomial passing through the points $(x_0, f(x_0))$, $(x_1, f(x_1))$, $(x_2, f(x_2))$, $\ldots$, $(x_N, f(x_n))$.
We define $L_{n,k}(x)$, the **Lagrange coefficients:**

$$\mathbf{L_{n,k}(x)} = \prod_{\mathbf{i=0,\, i\neq k}}^{\mathbf{n}} \frac{\mathbf{x - x_i}}{\mathbf{x_k - x_i}} = \frac{x - x_0}{x_k - x_0} \cdots \frac{x - x_{k-1}}{x_k - x_{k-1}} \cdot \frac{x - x_{k+1}}{x_k - x_{k+1}} \cdots \frac{x - x_{n1}}{x_k - x_n},$$

which have the properties

$$L_{n,k}(x_k) = 1; \quad L_{n,k}(x_i) = 0, \ \forall i \neq k.$$

Example of $L_{n,k}(x)$



This is $L_{6,3}(x)$, for the points $x_i = i$, $i = 0, \ldots, 6$.

The n^{th} Lagrange Interpolating Polynomial

We use $L_{n,k}(x)$, $k = 0, \ldots, n$ as building blocks for the Lagrange interpolating polynomial:

$$P(x) = \sum_{k=0}^{n} f(x_k) L_{n,k}(x),$$

which has the property

$$P(x_i) = f(x_i), \quad \forall i = 0, \ldots, n.$$

This is the unique polynomial passing through the points $(x_i, f(x_i))$, $i = 0, \ldots, n$.

Error bound for the Lagrange interpolating polynomial

Suppose x_i, $i = 0, \ldots, n$ are distinct numbers in the interval $[a, b]$, and $f \in C^{n+1}[a, b]$. Then $\forall x \in [a, b]$ $\exists\, \xi(x) \in (a, b)$ so that:

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

where $P_{Lagrange}(x)$ is the n^{th} Lagrange interpolating polynomial.

Compare with the error formula for Taylor polynomials

$$f(x) = P_{\text{Taylor}}(x) + \frac{f^{(n+1)}(\xi(x))}{(n+1)!} (x - x_0)^{n+1},$$

Problem: Applying the error term may be difficult...
The error formula is important as Lagrange polynomials are used for numerical differentiation and integration methods.

The Lagrange and Taylor Error Terms

Just to get a feeling for the non-constant part of the error terms in the Lagrange and Taylor approximations, we plot those parts on the interval $[0, 4]$ with interpolation points $x_i = i$, $i = 0, 1, \ldots, 4$:

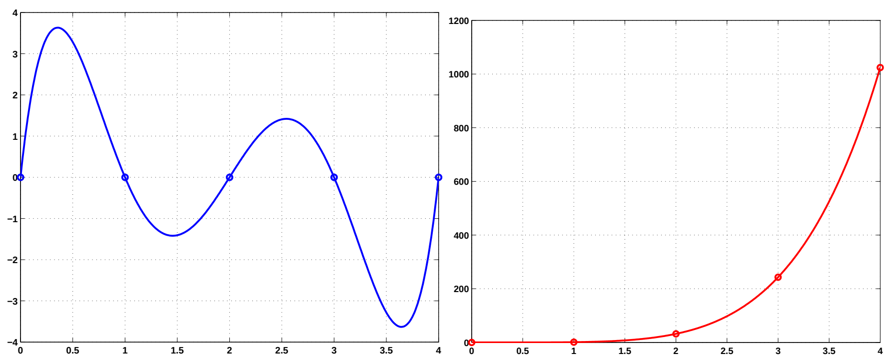


Figure: [LEFT] The non-constant error terms for the Lagrange interpolation oscillates in the interval $[-4, 4]$ (and takes the value *zero* at the node point x_k), and [RIGHT] the non-constant error term for the Taylor extrapolation grows in the interval $[0, 1024]$.

Practical Problems

Applying (estimating) the error term is difficult.

The degree of the polynomial needed for some desired accuracy is not known until after cumbersome calculations — checking the error term.

If we want to increase the degree of the polynomial (to e.g. $n+1$) the previous calculations are not of any help...

Building block for a fix: Let f be a function defined at $x_0, \ldots, x_n$, and suppose that $m_1, m_2, \ldots, m_k$ are k $(< n)$ distinct integers, with $0 \leq m_i \leq n \;\forall i$. The Lagrange polynomial that agrees with $f(x)$ the k points $x_{m_1}, x_{m_2}, \ldots, x_{m_k}$, is denoted $P_{m_1, m_2, \ldots, m_k}(x)$.

Note: $\{m_1, m_2, \ldots, m_k\} \subset \{0, 1, \ldots, n\}$.

Increasing the degree of the Lagrange Interpolation

Theorem

Let f be defined at $x_0, x_1, \ldots, x_k$, and x_i and x_j be two distinct points in this set, then

$$P(x) = \frac{(x - x_j) P_{0,\ldots,j-1,j+1,\ldots,k}(x) - (x - x_i) P_{0,\ldots,i-1,i+1,\ldots,k}(x)}{x_i - x_j}$$

is the k^{th} Lagrange polynomial that interpolates f at the $k+1$ points $x_0, \ldots, x_k$.

Recursive Generation of Higher Degree Lagrange Interpolating Polynomials

x_0	P_0				
x_1	P_1	$P_{0,1}$			
x_2	P_2	$P_{1,2}$	$P_{0,1,2}$		
x_3	P_3	$P_{2,3}$	$P_{1,2,3}$	$P_{0,1,2,3}$	
x_4	P_4	$P_{3,4}$	$P_{2,3,4}$	$P_{1,2,3,4}$	$P_{0,1,2,3,4}$

Neville's Method

The notation in the previous table gets cumbersome... We introduce the notation $Q_{\text{Last},\text{Degree}} = P_{\text{Last}-\text{Degree},\ldots,\text{Last}}$, the table becomes:

x_0	$Q_{0,0}$				
x_1	$Q_{1,0}$	$Q_{1,1}$			
x_2	$Q_{2,0}$	$Q_{2,1}$	$Q_{2,2}$		
x_3	$Q_{3,0}$	$Q_{3,1}$	$Q_{3,2}$	$Q_{3,3}$	
x_4	$Q_{4,0}$	$Q_{4,1}$	$Q_{4,2}$	$Q_{4,3}$	$Q_{4,4}$

Compare with the old notation:

x_0	P_0				
x_1	P_1	$P_{0,1}$			
x_2	P_2	$P_{1,2}$	$P_{0,1,2}$		
x_3	P_3	$P_{2,3}$	$P_{1,2,3}$	$P_{0,1,2,3}$	
x_4	P_4	$P_{3,4}$	$P_{2,3,4}$	$P_{1,2,3,4}$	$P_{0,1,2,3,4}$

Algorithm: Neville's Method — Iterated Interpolation

Algorithm: Neville's Method

To evaluate the polynomial that interpolates the $n+1$ points $(x_i, f(x_i))$, $i = 0, \ldots, n$ at the point x:

1. Initialize $Q_{i,0} = f(x_i)$.
2.

```
FOR i = 1 : n
  FOR j = 1 : i
```

$$Q_{i,j} = \frac{(x - x_{i-j})Q_{i,j-1} - (x - x_i)Q_{i-1,j-1}}{x_i - x_{i-j}}$$

```
  END
END
```

3. Output the Q-table.