Lecture 5: Solving Equations and Polynomial Interpolation

Instructor: Professor Amos Ron Scribes: Yunpeng Li, Mark Cowlishaw, Nathanael Fillmore

1 More on Newton's Method and the Secant Method

In the last lecture, we discussed methods for solving equations in one variable:

$$f(x) = 0$$

Two important methods we discussed were Newton's Method and the Secant Method.

1.1 Review of Newton's Method

Recall that Newton's method is a special case of the method of fixed point iterations. Given an equation $f(x) = 0$, we define a sequence $x_0, x_1, \ldots$ using an initial approximation x_0, and the formula:

$$x_{n+1} = x_n - \frac{f(x)}{f'(x)} \tag{1}$$

1.1.1 Convergence for Newton's Method

As we proved, provided that f is regular enough, Newton's method provides quadratic convergence.

$$e_{n+1} \leq \underbrace{\frac{g''(c_n)}{2}}_{*} \cdot e_n^2 \tag{2}$$

To determine how well Newton's method will converge in a particular case, it is important to look at the value of the constant (*). If this constant is very large, then we won't enjoy good convergence until we are quite close to the root (and $e_n \ll 1$). We can estimate the value of (*) by examining the second derivative of g near the root. If we differentiate g from equation 1, we see that

$$g'(x) = f(x) \cdot \frac{f''(x)}{[f'(x)]^2} \tag{3}$$

differentiating again, we get:

$$g''(x) = f'(x) \cdot \frac{f''(x)}{[f'(x)]^2} + \underbrace{f(x) \cdot [\square]}_{\to 0} \tag{4}$$

$$\approx \frac{f''(x)}{f'(x)} \tag{5}$$

Where $\square$ is a term involving the third derivative of f. Since this term vanishes at r, we may ignore it, except to note that f must be triply differentiable for this analysis to be valid.

Here is an example:

EXAMPLE 1.1. *Solve the equation*

$$x^2 = c$$

Since we have $f(x) = x^2 - c$,

$$\begin{aligned} g''(r) &= \left.\frac{f''(x)}{f'(x)}\right|_{x=r} \\ &= \frac{2}{2r} \\ &= \frac{1}{r} \end{aligned}$$

So, in this case, as long as r is not too small, Newton's method will converge quickly.

1.2 Review of the Secant Method

In the Secant method, we define the sequence $x_0, x_1, x_2, \ldots$ using two initial guesses, x_0 and x_1 and the formula:

$$x_{n+1} = x_n - \frac{f(x_n)[x_n - x_{n-1}]}{f(x_n) - f(x_{n-1})} \tag{6}$$

1.2.1 Convergence

The Secant method converges more quickly than a method with linear convergence, but more slowly than a method with quadratic convergence:

$$|e_{n+1}| \leq c_f \cdot |e_n^\alpha| \qquad (\alpha = \frac{1+\sqrt{5}}{2} \approx 1.6) \tag{7}$$

Where c_f is a constant dependent on the derivative. We call this kind of convergence *super-linear*

1.3 Comparison of Newton's Method and the Secant Method

So, which method is faster? Ignoring constants, it would seem obvious that Newton's method is faster, since it converges more quickly. However, to compare performance, we must consider both cost *and* speed of convergence. An algorithm that converges quickly but takes a few seconds per iteration may take far more time overall than an algorithm that converges more slowly, but takes only a few milliseconds per iteration.

For the purpose of this general analysis, we may assume that the cost of an iteration is dominated by the evaluation of the function - this is likely the case in practice. So, the number of function evaluations per iteration is likely a good measure of cost.

The secant method requires only one function evaluation per iteration, since the value of $f(x_{n-1})$ can be stored from the previous iteration.

Newton's method requires one function evaluation and one evaluation of the derivative per iteration. It is difficult to estimate the cost of evaluating the derivative in general. In some cases, the derivative may be easy to evaluate, in some cases, it may be much harder to evaluate than the function (if it is possible at all). It seems safe, though, to assume that, in most cases, evaluating the derivative is at least as costly as evaluating the function. Thus, we can estimate that Newton's method takes about two function evaluations per iteration.

This disparity in cost means that we can run two iterations of the secant method in the time it will take to run one iteration of Newton's method. So, to compare the performance of the two methods, we must compare the speed of convergence of two iterations of the secant method with one iteration of Newton's method. This is given by:

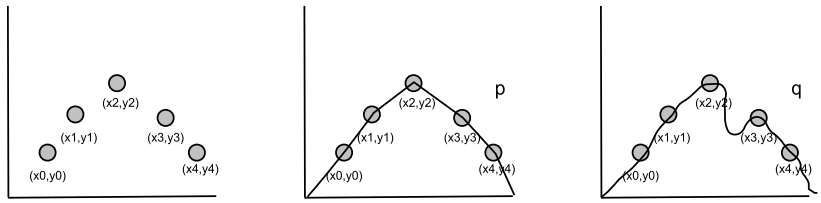
$$\begin{aligned} e_{n+2} &\leq c_f \cdot |e_{n+1}|^\alpha && (8)\\ &\leq c_f \cdot |c_f e_n^\alpha|^\alpha && (9)\\ &\leq c_f^{\alpha+1} \left| e_n^{\alpha^2} \right| && (10) \end{aligned}$$

And, since $\alpha^2 > 2$, we conclude that the secant method has better overall performance than Newton's method.

This concludes our discussion of solving equations, we now turn to our next major topic in numerical computation.

2 Polynomial Interpolation

To begin our discussion of polynomial interpolation, we'll start with some examples.



(a) (b) (c)

Figure 1: Polynomial Interpolation of One Point

EXAMPLE 2.1. *Suppose we are given the five points $(x_0, y_0), (x_1, y_1), \ldots, (x_4, y_4)$ shown in Figure 1(a). We want to interpolate these points by "some" function p, that is, we want to somehow come up with a function p such that $p(x_i) = y_i$ for $i = 0, \ldots, 4$. Of course, our task is still a little vague: for example, consider the two functions p and q shown in Figures 1(b,c). Both these functions interpolate the points, and depending on what we want to use the interpolated function for, either of them could be acceptable. On the other hand, the interpolating function q looks a little strange, and if we assume the points $(x_0, y_0), \ldots, (x_4, y_4)$ were sampled from the graph of some specific (but unknown) function, then at most one of these interpolating functions can be correct. We might want to formulate the interpolation problem so that only one solution is possible given any collection of points, and we might want to ensure that this solution is in some sense "simplest".*

EXAMPLE 2.2. *Suppose we are given a point $p = (x_0, y_0)$ from the graph of some function f, and we would like to make as good a guess as possible at which function f is. Clearly, there are an infinite number of functions whose graphs contain p. However, as it turns out, the constant function:*

$$f(x) = y_0$$

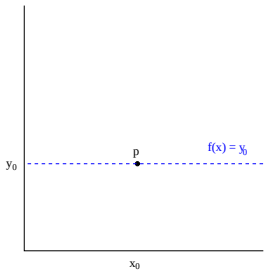


Figure 2: Polynomial Interpolation of One Point

is a good solution (in the absence of more data), as shown in Figure 2. If we consider the value of f at some other x-coordinate x_1, given the information we have, it is just about equally likely that $f(x_1) > y_0$ as it is that $f(x_1) < y_0$. In the absence of more information, $f(x) = y_0$ is just about as well as we can do. Note that it is the only polynomial of degree 0 whose graph goes through p.

EXAMPLE 2.3. *Now, suppose that we are given two points, $p_0 = (x_0, y_0)$ and $p_1 = (x_1, y_0)$ from the graph of a function f. Once again, the constant function*

$$f(x) = y_0$$

is a good guess at f, and the only constant function whose graph goes through p_0 and p_1.

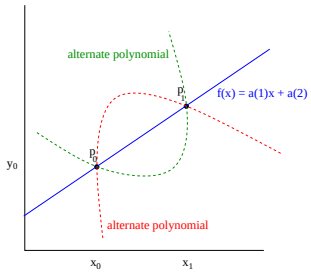


Figure 3: Polynomial Interpolation of Two Points

EXAMPLE 2.4. *Finally, suppose that we are given two points $p_0 = (x_0, y_0)$ and $p_1 = (x_1, y_1)$ from the graph of some function f as shown in Figure 3. We can estimate f as the line passing through both points. Note that, it is also possible to estimate f as some higher-order function, such as a parabola passing through both points, but there are infinitely many such functions, and we have no reason to choose one over another.*

Note from the examples that additional points do not necessarily require a more complex function, as in example 2.3, when a constant function was able to represent two distinct points.

2.1 Formal Definition of Polynomial Interpolation

To define polynomial interpolation more formally, we'll begin by defining the set of polynomials we can draw from:

DEFINITION 2.1. $\Pi_n = \{$*all polynomials in one variable of degree* $\leq n\}$

Using this definition, we can define polynomial interpolation formally:

DEFINITION 2.2 (Polynomial Interpolation). *Given a set of* $n+1$ *points:*

$$(x_0, y_0), (x_1, y_1), (x_2, y_2), \ldots, (x_{n-1}, y_{n-1}), (x_n, y_n) \qquad (\forall_{i,j,\, i \neq j} x_i \neq x_j)$$

Find a polynomial $p \in \Pi_n$ *such that* $p(x_i) = y_i$ *for* $i = 0, 1, 2, \ldots, n$.

2.2 Goals

We would like to demonstrate the following things about polynomial interpolation:

1. Existence - Does a solution always exist?
2. Uniqueness - Is it possible that there is more than one solution?
3. Algorithms - What methods exist for finding the solution?

We will show in the next lecture that there is always a unique solution to any polynomial interpolation problem as we have defined it.

2.3 A Linear System of Equations

Recall (Definitions 2.1 and 2.2) that we have defined polynomial interpolation to be the task of finding a polynomial of degree $\leq n$ that interpolates a given set of $n+1$ points $(x_0, y_0), \ldots, (x_n, y_n)$. What does "find" mean in this definition? Earlier, when we were learning about root-finding, we said that to "have" a function f is to be able to compute $f(x)$ for any x in the domain of f. Here, we want to have an explicit representation of the interpolating polynomial p, and we will use this to compute $p(x)$ for any x.

A natural way to write polynomials is as a linear combination of monomials:

$$\Pi_n = \{a_0 x^n + a_1 x^{n-1} + \cdots + a_{n-1} x + a_n : a_0, \ldots, a_n \in \mathbb{R}\}$$

For example, any linear (or constant) polynomial can be represented as:

$$p(x) = a_0 \cdot x + a_1 \qquad a = (a_0, a_1) \in \mathbb{R}^2 \tag{11}$$

while any quadratic (or lower order) polynomial can be represented as:

$$p(x) = a_0 \cdot x^2 + a_1 \cdot x + a_2 \qquad a = (a_0, a_1, a_2) \in \mathbb{R}^3 \tag{12}$$

So to "find" a polynomial p it is sufficient to find coefficients $a_0, a_1, \ldots, a_n$ of a linear combination of monomials.

Given this representation, we can write the polynomial interpolation problem as a system of linear equations.

EXAMPLE 2.5. *Solve the polynomial interpolation problem for the points* $(1, 7)$, $(-3, 11)$, $(2, 8)$.

Since we are given three points, we know that $n = 2$, and we are looking for a polynomial $p \in \Pi_2$. We can represent this problem as a system of three equations:

$$\begin{aligned} a_0 \cdot 1^2 + a_1 \cdot 1 + a_2 &= 7 \\ a_0 \cdot (-3)^2 + a_1 \cdot (-3) + a_2 &= 11 \\ a_0 \cdot 2^2 + a_1 \cdot 2 + a_2 &= 8 \end{aligned}$$

or

$$\begin{pmatrix} 1^2 & 1^1 & 1^0 \\ (-3)^2 & (-3)^1 & (-3)^0 \\ 2^2 & 2^1 & 2^0 \end{pmatrix} \cdot \begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix} = \begin{pmatrix} 7 \\ 11 \\ 8 \end{pmatrix}$$

Note that, since we could write similar equations for any instance of the polynomial interpolation problem, we can analyze polynomial interpolation just as we would any other linear problem.

2.4 Analyzing Polynomial Interpolation Using Linear Algebra

We have seen that we can transform a polynomial interpolation problem into a set of linear equations of the form

$$V \cdot \vec{a} = \vec{y} \tag{13}$$

Where V is a square matrix of dimension $n+1$, and $\vec{a}$ and $\vec{y}$ are column vectors of size $n+1$. Linear algebra tells us that there are two possible kinds of outcomes when we try to solve this system:

1. There may be a unique solution $\vec{a}$.
2. There may not be a unique solution
   (a) There may be no solutions
   (b) There may be more than one solution

Whether we are in situation (1) or (2) depends on a property of the matrix V called *singularity* (or regularity). If V is non-singular, then we are in situation (1), there is a unique solution. If V is singular then we are in situation (2), and whether there are no solutions or more than one is dependent on the vector $\vec{y}$ on the right hand side.

In later lectures, we will prove that a solution *always* exists, regardless of the points we use. It is not difficult to see that, if there is always a solution regardless of the particular points, then the matrix V cannot be singular and therefore, the solution is also unique.

2.5 General Solution to the Polynomial Interpolation Problem

We can generalize our solution for example 2.5. Given points:

$$(x_0, y_0), (x_1, y_1), \ldots, (x_{n-1}, y_{n-1}), (x_n, y_n) \qquad (\forall_{i,j,i \neq j} x_i \neq x_j)$$

We can find a polynomial of degree $\leq n$ whose graph passes through these points by solving the following linear system:

$$\underbrace{\begin{pmatrix} x_0^n & x_0^{n-1} & \dots & x_0^1 & x_0^0 \\ x_1^n & x_1^{n-1} & \dots & x_1^1 & x_1^0 \\ \vdots & \vdots & & \vdots & \vdots \\ x_{n-1}^n & x_{n-1}^{n-1} & \dots & x_{n-1}^1 & x_{n-1}^0 \\ x_n^n & x_n^{n-1} & \dots & x_n^1 & x_n^0 \end{pmatrix}}_{V} \cdot \underbrace{\begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \\ a_n \end{pmatrix}}_{a} = \underbrace{\begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \\ y_n \end{pmatrix}}_{y} \tag{14}$$

We call the matrix V the Vandermonde matrix.

Transforming a polynomial interpolation problem into a set of linear equations is helpful in understanding the problem, and particularly in proving the uniqueness of a solution. However, as we'll see, solving linear equations is a *terrible* method for polynomial interpolation.