CS416: INTRODUCTION TO SCIENTIFIC COMPUTING 2/2/07

Unit 3: Polynomial Interpolation

Notes prepared by: Amos Ron, Yunpeng Li, Mark Cowlishaw, Steve Wright
Instructor: Steve Wright

1 Introduction

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

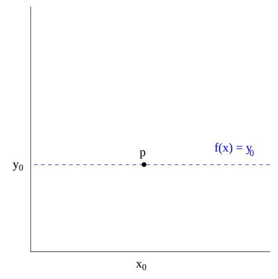


Figure 1: Polynomial Interpolation of One Point

Suppose we are given a point (x_0, y_0) from the graph of some function f. We know nothing else about f but would like to make a good guess at the complete graph of f based on this one data point. Clearly, there are an infinite number of functions p whose graphs contain (x_0, y_0). The simplest one is the constant function:

$$p(x) = y_0$$

shown in Figure 1. 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, the constant function $p(x) = y_0$ is as good a guess as any of the unknown function f. It is the only polynomial of degree 0 whose graph goes through f.

Now, suppose that we are given two points, (x_0, y_0) and (x_1, y_0) from the graph of a function f. Once again, the constant function

$$p(x) = y_0$$

is a good guess of the unknown function f, and is the only constant function whose graph goes through the points (x_0, y_0) and (x_1, y_0).

Finally, suppose that we are given two points (x_0, y_0) and (x_1, y_1) from the graph of some function f as shown in Figure 2. We can estimate f by the function p, which we define to be the line passing through both points. Note that, it is also possible to define some higher-order function p that also passes through the given points, but there are infinitely many such functions, and we

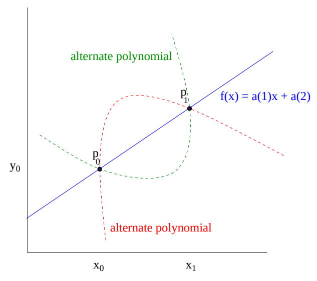


Figure 2: Polynomial Interpolation of Two Points

have no reason to choose one over another. Hence, we prefer to stick with the simplest function p that "explains" the data, which is the linear function. A formula for this function is as follows:

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

(Check that $y_0 = p(x_0)$ and $y_1 = p(x_1)$.)

1.1 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 1.1. $\Pi_n = \{$*all polynomials in one variable of degree* $\leq n\}$

Using this definition, we can define the polynomial interpolation problem formally as follows:

Definition 1.2 (Polynomial Interpolation). *Given a set of* $n+1$ *points:*

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

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

Alternatively, we can think of p as an approximation to a function $f(x)$ whose values are known at the points x_i, $i = 0, 1, 2, \ldots, n$, such that $x_i \neq x_j$ whenever $i \neq j$. We require

$$p(x_i) = f(x_i), \;\; i = 0, 1, 2, \ldots, n.$$

1.2 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'll show that there is always a unique solution to any polynomial interpolation problem as we have defined it.

2 Polynomial Interpolation as a Linear Algebra Problem

2.1 A Linear System of Equations

Note that we can write any polynomial as a linear combination of monomials. For example, any linear (or constant) polynomial can be represented as:

$$p(x) = a(1) \cdot x + a(2) \qquad a = [a(1), a(2)] \in \mathbb{R}^2 \tag{1}$$

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

$$p(x) = a(1) \cdot x^2 + a(2) \cdot x + a(3) \qquad a = [a(1), a(2), a(3)] \in \mathbb{R}^3 \tag{2}$$

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

Example 2.1. *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:

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

or

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

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 algebra problem.

2.2 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

$$V \cdot \vec{a} = \vec{y} \tag{3}$$

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* $y_0, y_1, \ldots, y_n$, then the matrix V cannot be singular and therefore, the solution is also unique.

2.3 General Solution to the Polynomial Interpolation Problem

We can generalize our solution for example 2.1. 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:

$$\underbrace{\begin{pmatrix} x_0^n & x_0^{n-1} & \ldots & x_0^1 & x_0^0 \\ x_1^n & x_1^{n-1} & \ldots & x_1^1 & x_1^0 \\ \vdots & \vdots & & \vdots & \vdots \\ x_{n-1}^n & x_{n-1}^{n-1} & \ldots & x_{n-1}^1 & x_{n-1}^0 \\ x_n^n & x_n^{n-1} & \ldots & x_n^1 & x_n^0 \end{pmatrix}}_{V} \cdot \begin{pmatrix} a(1) \\ a(2) \\ \vdots \\ a(n) \\ a(n+1) \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \\ y_n \end{pmatrix} \tag{4}$$

We call the matrix V the *Vandermonde matrix*.

If the ordinates y_i are the values of a given function f at the abscissae x_i, we can rewrite this system as follows, where $\vec{a} = (a(1), a(2), \ldots, a(n+1))^T$ and $F = (f(x_0), \ldots, f(x_n))^T$:

$$V \cdot \vec{a} = \vec{F}. \tag{5}$$

Transforming the problem of finding a polynomial p into an equivalent linear problem has helped us to understand polynomial interpolation. However, it has some significant drawbacks as a method for solving polynomial interpolation problems.

1. Solving a linear system of equations takes a significant amount of time
2. This system of linear equations is often ill-conditioned and prone to round-off errors

This equivalence between solving a linear problem and finding a polynomial p rests on our representation of p as a linear combination of monomials. To find a better method, we should consider alternate representations. Two methods of polynomial interpolation we shall talk about involve different polynomial representations. These are *Lagrange*[1] *polynomials* and *Newton polynomials*.

[1] Joseph-Louis Comte de Lagrange. 1736-1813. French mathematician; published works on celestial mechanics, differential and variable calculus, theory of numbers, etc.

3 Interpolation by Lagrange Polynomials

Given the $n+1$ interpolation points $x_0, x_1, \ldots, x_n$, suppose we can find $n+1$ polynomials $\ell_0(t), \ell_1(t), \ldots, \ell_n(t)$ of degree $\leq n$. These polynomials are fixed and independent of the function values $f(x_0), f(x_1), \ldots, f(x_n)$, and are defined as follows:

$$\ell_i(x_j) = \begin{cases} 1 & (\text{ if } i = j) \\ 0 & \text{Otherwise} \end{cases} \tag{6}$$

So, how can we find such polynomials, and, more importantly, how can we use them for interpolation?

3.1 Using Lagrange Polynomials for Interpolation

FACT 3.1. *Any polynomial $p \in \Pi_n$ can be represented as a linear combination of $n+1$ Lagrange polynomials of degree $\leq n$.*

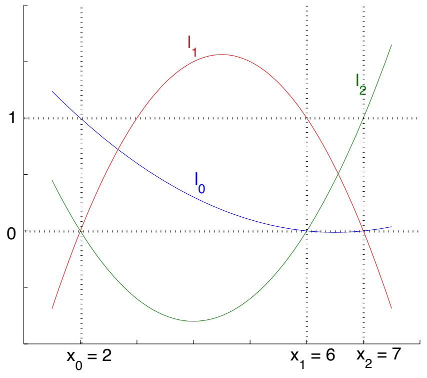


Figure 3: Quadratic Lagrange Polynomials ℓ_0, ℓ_1, ℓ_2 for $\vec{x} = (2\ 6\ 7)^T$

To illustrate the use of Lagrange polynomials, consider the following example.

EXAMPLE 3.1. *Given the points $\vec{x} = (2\ \ 6\ \ 7)^T$, and the corresponding function values $\vec{F} = (-1\ \ 8\ \ -3)^T$, find an interpolating polynomial $p \in \Pi_2$.*

Consider that, if we had three quadratic Lagrange polynomials $\ell_0(t), \ell_1(t), \ell_2(t)$ (as shown in Figure 3) with

$$\ell_0(\vec{x}) = \begin{pmatrix} 1 & 0 & 0 \end{pmatrix}^T$$
$$\ell_1(\vec{x}) = \begin{pmatrix} 0 & 1 & 0 \end{pmatrix}^T$$
$$\ell_2(\vec{x}) = \begin{pmatrix} 0 & 0 & 1 \end{pmatrix}^T$$

If we multiply each polynomial l_i by the corresponding function value $f(x_i)$, then add them together, the resulting polynomial will match f at each of the input points. So, we can define p as:

$$p(t) = -1 \cdot \ell_0(t) + 8 \cdot \ell_1(t) - 3 \cdot \ell_2(t)$$

Evaluating p at the input point $x_0 = 2$, we see:

$$\begin{aligned} p(2) \quad &= -1 \cdot \ell_0(2) + 8 \cdot \ell_1(2) - 3 \cdot \ell_2(2) \\ &= -1 \cdot 1 + 8 \cdot 0 - 3 \cdot 0 \\ &= -1 \end{aligned}$$

In general, given a set of input points $x_0, x_1, \ldots, x_n$, the corresponding points on the graph of the function f, $f(x_0), f(x_1), \ldots, f(x_n)$, and the Lagrange polynomials for our input points $\ell_0(t), \ell_1(t), \ldots, \ell_n(t)$, we can define the following polynomial $p \in \Pi_n$ that interpolates f:

$$p(t) = \sum_{i=0}^{n} f(x_i)\ell_i(t) \tag{7}$$

3.2 Finding Lagrange Polynomials

So how can we find a polynomial ℓ_i given its roots and a point x_i with $\ell_i(x_i) = 1$? Consider that, if a polynomial has roots r_1, r_2, then it must have factors $(t - r_1)$ and $(t - r_2)$. We can get our polynomial by first multiplying the factors together, obtaining a polynomial with the correct roots. We can then multiply by a constant scaling factor so that $\ell_i(x_i) = 1$.

To illustrate, consider Example 3.1. Since ℓ_0 is 0 at 6 and 7, it must have factors $(t - 6)$ and $(t-7)$. If we consider $(t-6)\cdot(t-7)$ evaluated at $x_0 = 2$, we see that its value will be $(2-6)\cdot(2-7)$, so, if we divide by this value, we will have the correct polynomial:

$$\ell_0(t) = \frac{(t-6)\cdot(t-7)}{(2-6)\cdot(2-7)}$$

We can define ℓ_1 and ℓ_2 similarly:

$$\begin{aligned} \ell_1(t) &= \frac{(t-2)\cdot(t-7)}{(6-2)\cdot(6-7)} \\ \ell_2(t) &= \frac{(t-2)\cdot(t-6)}{(7-2)\cdot(7-6)} \end{aligned}$$

In general, we can calculate each Lagrange polynomial ℓ_i for a set of input points $x_0, x_1, \ldots, x_n$ as:

$$\ell_i(t) = \prod_{\substack{j=0 \\ j \neq i}}^{n} \left(\frac{(t - x_j)}{(x_i - x_j)} \right) \tag{8}$$

3.3 Analysis of Interpolation by Lagrange Polynomials

So we have seen that we can solve a polynomial interpolation problem quite easily using a linear combination of Lagrange polynomials. We should ask ourselves how easy it is to use the resulting polynomial for numerical computation. For example:

- How easy is it to evaluate?
- How easy is it to evaluate the derivative?

We might argue that evaluating the polynomial is fairly straightforward. However, it is easy to see that evaluating the derivative could be very complicated (consider using the product rule on n terms with n factors each).

Thus, with Lagrange polynomials, we have struck on a polynomial representation that makes it easy to solve polynomial interpolation problems, but potentially difficult to use once the problem has been solved.

How important is it that the polynomial representation we end up with is easy to use? Consider that we have already used polynomial interpolation without knowing it, when we approximated the derivative of a function.

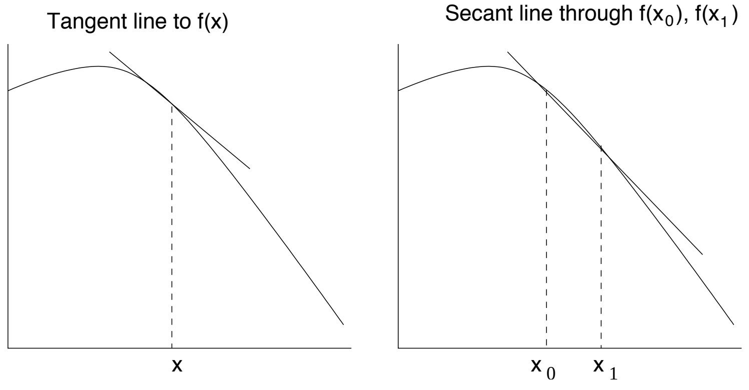


Figure 4: Approximating the Derivative Using a Secant Line

Recall that the derivative of a function f at a point $(x, f(x))$ is simply the slope of the tangent line at that point. We can approximate the tangent line by taking two points $(x_0, f(x_0))$, $(x_1, f(x_1))$, and interpolating a degree ≤ 1 polynomial (a secant line) that approximates the function between these two points (as shown in Figure 4). We can then approximate the slope at x by taking the derivative of the secant line. If we look at this problem as a polynomial interpolation problem, we might get a better estimate by interpolating over a larger number of points. In cases like this, clearly it is important that the representation of the resulting polynomial is easy to use.

3.4 Uniqueness of Solutions

The process of interpolation by Lagrange polynomials shows that we can *always* find a solution to a polynomial interpolation problem.

Recall that polynomial interpolation is equivalent to solving the linear problem:

$$V\vec{a} = \vec{F} \tag{9}$$

From linear algebra, we know that the solution to this problem hinges on whether or not the matrix V is *singular*. If V is singular, then whether or not the equation has a solution depends on $\vec{F}$:

$$\begin{aligned} V \text{ is non-singular} \quad &\Rightarrow \quad \forall \vec{F} \text{ equation 9 has a unique solution} \\ V \text{ is singular} \quad &\Rightarrow \quad \begin{cases} \exists \vec{F} \text{ such that equation 9 has no solutions} \\ \exists \vec{F} \text{ such that equation 9 has infinitely many solutions} \end{cases} \end{aligned}$$

Now, since we know that we can *always* solve a polynomial interpolation problem using Lagrange polynomials, we know that equation 9 *always* has a solution, regardless of $\vec{F}$. Since this is true, V cannot be singular, and therefore, we know that not only does equation 9 always have a solution, but that solution is always unique.

4 Motivation for Newton interpolation

Recall the polynomial interpolation problem we have been studying for the last few lectures. Given $\vec{X} = (x_0, x_1, \cdots, x_n)$ and the value of f at the given points $\vec{F} = (f(x_0), f(x_1), \cdots, f(x_n))$, find the polynomial $p \in \Pi_n$ so that $p|_{\vec{X}} = \vec{F}$. We proved in the previous lecture that the solution to this problem always exists and is unique.

We introduced two methods for solving the polynomial interpolation problem, based on two different representations of the polynomial solution:

1. We can represent the solution as a linear combination of monomials.

   $$P(t) = a_1 x^n + a_2 x^{n-1} + \cdots + a_n x + a_{n+1}$$

   Using this representation, the problem is equivalent to solving a system of linear equations

   $$V\vec{a} = \vec{F}$$

   Where V is the Vandermonde matrix, $\vec{a}$ is the column vector of (unknown) polynomial coeffcients, and $\vec{F}$ is the column vector of function values at the interpolation points.

   As we have seen, solving this system is not a good method for solving the polynomial interpolation problem. However, this representation is important, since we were able to use it to conclude that the solution to a polynomial interpolation problem is unique.

2. We can represent the solution using Lagrange polynomials.

   $$P(t) = f(x_0) \cdot \ell_0(t) + f(x_1) \cdot \ell_1(t) + \cdots + f(x_n) \cdot \ell_n(t)$$

   Using this representation, we showed that it is easy to find a solution p, and this led to the important conclusion that the polynomial interpolation problem always has a soluton. However, a solution written in Lagrange form may be difficult to use.

4.1 Problems with the Lagrange Representation

Our goal when solving a polynomial interpolation problem is to find a polynomial function p that approximates the function f, which we may know nothing about, other than its value at a few points. Once we have this polynomial function p, we may want to use it to study f in various ways.

Evaluation We may want to use p to approximate f at points outside the set of interpolation points. The Lagrange representation may be problematic for this purpose if we have interpolation points (x_a, x_b) that are very close together. Recall the formula for a Lagrange polynomial $\ell_i(t)$.

$$\ell_i(t) = \prod_{\substack{j=0 \\ j \neq i}}^{n} \left(\frac{(t - x_j)}{(x_i - x_j)} \right)$$

Those Lagrange polynomials that include $(x_a - x_b)$ or $(x_b - x_a)$ in the denominator will be quite large, and may suffer from loss of significance.

Evaluation of Derivative The symbolic differentiation of a Lagrange polynomial, using the product rule, will have n separate terms, each involving $n-1$ multiplications. Sine the solution to the polynomial interpolation problem p contains $n+1$ different Lagrange poynomials, evaluating the derivative could be very inefficient.

Clearly, the Lagrange representation is not ideal for some uses of the polynomial p.

Additionally, there is another problem with the Lagrange representation, it doesn't take advantage of the inductive structure of a polynomial interpolation problem. Polynomial interpolation is often an iterative process where we add points one at a time until we reach an acceptable solution. Given a particular solution p to a polynomial interpolation problem with n points, we would like to add an additional point and find a new solution p' without too much additional cost. In other words, we would like to reuse p to help us find p'.

It is easy to see that the Lagrange representation will not meet this requirement. Using the Lagrange representation, we can find a solution to a polynomial interpolation problem with n points $X = (x_0, x_1, \ldots, x_{n-2}, x_{n-1})$ easily, but, if we add a single additional point x_n, we have to throw out our previous solution and recalculate the solution to the new problem from scratch. This is because every interpolation point is used in every term of the Lagrange representation.

We shall now discuss a polynomial representation that makes use of the inductive structure of the problem. Note, however, that despite its shortcomings, the Lagrange representation is not a bad way to represent polynomials in many cases.

5 Newton Polynomials

We introduce Newton polynomials with a series of examples.

EXAMPLE 5.1. *Given a set of points $\vec{X} = (1, 3, 0)$ and the corresponding function values $F = (2, 7, -8)$, find a quadratic polynomial that interpolates this data.*

Step 1 The order of the points does not matter (as long as each point is matched to the corresponding function value), but we will consider the first point $x_0 = 1, F_0 = 2$, and find a zero order polynomial p_0 that interpolates this point.

$$\begin{aligned} N_0(t) &= 1 \\ p_0 &= 2 \cdot N_0(t) \end{aligned}$$

We call N_0 the zero order Newton polynomial.

Step 2 We use our previous solution p_0 and the first order Newton polynomial N_1 to interpolate the first two points $(1, 2)$ and $(3, 7)$.

$$\begin{aligned} N_1(t) &= (t-1) \\ p_1(t) &= p_0(t) + \Box \cdot N_1(t) \end{aligned}$$

We need to find the value of $\Box$ given the two constraints that $p_1(1) = 2$ and $p_1(3) = 7$.

$$\begin{aligned} p_1(1) \quad &= p_0(1) + \Box \cdot (1-1) = 2 + 0 = 2 \\ p_1(3) \quad &= p_0(3) + \Box \cdot (3-1) \\ 7 \quad &= 2 + \Box \cdot 2 \\ \Box \quad &= 5/2 \end{aligned}$$

Thus, $p_1(t) = p_0(t) + 5/2 \cdot N_1(t)$.

Step 3 We use our previous solution p_1 and the second order Newton polynomial N_2 to interpolate all three points.

$$\begin{aligned} N_2(t) \quad &= (t-1) \cdot (t-3) \\ P_2(t) \quad &= p_1(t) + \Box \cdot N_2(t) \end{aligned}$$

Solving for the three constraints $(1, 2)$, $(3, 7)$, and $(0, -8)$, we obtain:

$$\begin{aligned} p_2(1) \quad &= p_1(1) + \Box \cdot 0 \\ p_2(3) \quad &= p_1(3) + \Box \cdot 0 \\ P_2(0) \quad &= p_1(0) + \Box \cdot 3 \\ -8 \quad &= -1/2 + \Box \cdot 3 \\ \Box \quad &= -5/2 \end{aligned}$$

Yielding the final solution, $P_2(t) = -5/2 \cdot N_2(t)$

To solve polynomial interpolation problems using this iterative method, we need the polynomials $N_i(t)$. We call these *Newton polynomials.*

Definition 5.1 (Newton Polynomials). *Given a set of $n+1$ input points $\vec{X} = (x_0, x_1, \dots, x_n)$, we define the $n+1$ Newton polynomials.*

$$\begin{aligned}
N_0(t) &= 1 \\
N_1(t) &= t - x_0 \\
\vdots \quad & \vdots \\
N_n(t) &= (t - x_0) \cdot (t - x_1) \cdot \dots \cdot (t - x_{n-1})
\end{aligned}$$

The following general formula can be used to define Newton polynomials:

$$N_i(t) = \prod_{j=0}^{i-1} (t - x_j)$$

Note that the Newton polynomials for a polynomial interpolation depend only on the input points $\vec{X} = (x_0, x_1, \dots, x_n)$, and not on the associated function values $\vec{F} = (f(x_0), f(x_1), \dots, f(X(n))$. Given these Newton polynomials, we can define a recurrence for $p_n(t)$:

$$p_n(t) = p_{n-1}(t) + \square \cdot N_n(t) \tag{10}$$

We can find the coefficient $\square$ for each Newton polynomial using the method of *divided differences.*

6 Divided Differences

Definition 6.1 (Divided Differences). *Given a set of $n+1$ input points $\vec{X} = (x_0, x_1, \dots, x_n)$, and the corresponding function values $\vec{F} = (f(x_0), f(x_1), \dots, f(x_n))$, and the Newton representation of the polynomial interpolant $p_n(t) = a_0 N_0(t) + a_1 N_1(t) + \dots + a_n N_n(t)$, the coefficient a_n of the nth Newton polynomial $N_n(t)$ is called the divided difference (or D.D.) of f at $(x_0, x_1, \dots, x_n)$ and is denoted by $f[x_0, x_1, \dots, x_n]$*

Example 6.1. *Given $\vec{X} = (1, 3, 0)$ and $\vec{F} = (2, 7, -8)$, then*

$$\begin{aligned}
p_0(t) &= f[1] \cdot N_0(t) \qquad (11) \\
p_1(t) &= p_0(t) + f[1,3] \cdot N_1(t) \qquad (12) \\
p_2(t) &= p_1(t) + f[1,3,0] \cdot N_2(t) \qquad (13)
\end{aligned}$$

Note that a polynomial interpolation problem is invariant over the ordering of the input points. If we rewrote $P_2(t)$ in equation (13) in monomial form, the coefficient of t^2 would be the divided difference $f[1, 3, 0]$. If we were to reorder the input points as $(3, 0, 1)$ and solve the problem using Newton polynomials, the coefficient of t^2 would be $f[3, 0, 1]$. Since these problems are equivalent, this means that $f[1, 3, 0] = f[3, 0, 1]$. In general, the final coefficient in the Newton representation of a polynomial interpolant is always the same under any ordering of the input points. This means that the divided difference $f[x_0, x_1, \dots, x_n]$ is invariant over the ordering of the points $(x_0, x_1, \dots, x_n)$.

Finally, we define the order of a divided difference.

Definition 6.2. *The* order *of a divided difference $f[x_0, x_1, \ldots, x_n]$ is one less then the number of points in the divided difference.*

A divided difference of order k, $f[x_0, x_1, \ldots, x_k]$ is the coefficient of the final (order k) Newton polynomial in the interpolant of $(x_0, x_1, \ldots, x_k)$.

6.1 Computing Divided Differences

Given a polynomial interpolation problem $\vec{X} = (x_0, x_1, \ldots, x_n)$, $\vec{F} = (f(x_0), f(x_1), \ldots, f(x_n))$, we compute the divided differences by constructing a special half-table, with the rows indexed by the input points $\vec{X}$ and the columns indexed by the order of the divided difference. The values in the first column (column 0) are simply the function values $\vec{F}$. Each successive value $D[i, j]$ (the ith row and jth column of the table) is calculated as a quotient. We define $D[i, j]$ as follows:

$$D[i, j] = f[x_i, x_{i+1}, \ldots, x_{i+j}].$$

The numerator is the difference of the values immediately left ($D[i, j-1]$) and immediately on the downward left diagonal ($D[i+1, j-1]$). The denominator is the difference between the input value x_i labeling the ith row, and the input value x_{i+j} labeling the $i+j$th row. More formally, given a polynomial interpolation problem with $n+1$ input points, we define the following table D:

x_0	$D[0,0]$	$D[0,1]$	$\ldots$	$D[0,N]$
x_1	$D[1,0]$	$\ldots$	$D[1,N-1]$	
		$\vdots$		
x_N	$D[N,0]$			

we can calculate the individual values $D[i, j]$ in the table D as follows.

$$D[i, j] = \begin{cases} f(x_i) & \text{if } j = 0 \\ \frac{D[i+1, j-1] - D[i, j-1]]}{x_{i+j} - x_i} & \text{Otherwise} \end{cases} \tag{14}$$

We can write this formula in terms of the $f[]$ notation as follows:

$$f[x_i, \ldots, x_{i+j}] = \begin{cases} f(x_i) & \text{if } j = 0 \\ \frac{f[x_{i+1}, \ldots, x_j] - f[x_i, \ldots, x_{i+j-1}]}{x_{i+j} - x_i} & \text{Otherwise} \end{cases} \tag{15}$$

For example, if we are given $\vec{X} = (2, 6, 7, 0)$ and $\vec{F} = (1, -1, 0, 2)$, we generate the divided difference array D as

2	1	$\frac{-1-1}{6-2} = -0.5$	$\frac{3}{10}$	$\frac{3}{70}$
6	-1	$\frac{0-(-1)}{7-6} = 1$	$\frac{3}{14}$	
7	0	$\frac{2-0}{0-7} = \frac{-2}{7}$		
0	2			

Note that the first row is $f[2], f[2, 6], f[2, 6, 7]$ and $f[2, 6, 7, 0]$. In general, we only need the first row to construct the Newton representation of the polynomial interpolant. However, we need the other entries in the table to calculate all the values in the first row.

Using the definitions of divided differences above, in combination with Definition 5.1, we can define the *Newton form of the interpolating polynomial* as follows:

$$p_n(t) = \sum_{i=0}^{n} f[x_0, x_1, \ldots, x_i] N_i(t) = \sum_{i=0}^{n} f[x_0, x_1, \ldots, x_i] \prod_{j=0}^{i-1} (t - x_j). \tag{16}$$

EXAMPLE 6.2. *For $\vec{X} = (1, 0, -1)$ and $\vec{F} = (2, 4, 8)$, we create the divided difference table as*

1	*2*	*-2*	*1*
0	*4*	*-4*	
-1	*8*		

and we have

$$p(t) = 2 \cdot 1 - 2 \cdot (t - 1) + 1 \cdot (t - 1)(t - 0)$$

as the solution.

Now when we add one more point and value, say, $\vec{X} = (1, 0, -1, 2)$ and $\vec{F} = (2, 4, 8, 2)$, we construct a new array, with the above array as the upper left corner.

1	2	-2	1
0	4	-4	
-1	8		

$\Rightarrow$

1	2	-2	1	-2
0	4	-4	-1	
-1	8	-2		
2	-2			

The work we did in calculating the divided differences for the first three points is not wasted when we add a fourth interpolation point.

Note that the diagonal in the divided difference table, corresponding to the last entry of each row ($D[i, n - i]$ for $i = 0, 1, \ldots, n$) computes the divided differences that we would calculate if the interpolation points $\vec{X} = (x_0, x_1, \ldots, x_n)$ were given in *reverse* order (i.e. $(x_n, x_{n-1}, \ldots, x_1, x_0)$). As always, the final divided difference (the coefficient of the nth order Newton polynomial) is the same no matter the order of the points.

$$f[x_0, x_1, \ldots, x_{n-1}, x_n] = f[x_n, x_{n-1}, \ldots, x_1, x_0] = D[0, n]$$

To illustrate, consider the following example.

EXAMPLE 6.3. *Given the input points $\vec{X} = (0, 1, 2)$ and corresponding function values $\vec{F} = (2, -2, 0)$ find the polynomial interpolant $p_2 \in \Pi_2$.*

To solve this problem using Newton polynomials, we build the following divided difference table.

0	2	-4	3
1	-2	2	
2	0		

The first row corresponds to the divided differences $f[0], f[0, 1], f[0, 1, 2]$, which are the divided differences we would calculate when building the solution in the order $\vec{X} = (0, 1, 2)$.

$$p_2(t) \qquad = 2 \cdot 1 - 4 \cdot (t - 0) + 3 \cdot (t - 0)(t - 1)$$

The diagonal (consisting of the last entry in each row) corresponds to the divided differences $f[2], f[1,2], f[0,1,2]$, which are the divided differences we would calculate when building the solution in the order $\vec{X} = (2,1,0)$.

$$p_2(t) \qquad = 0 \cdot 1 + 2 \cdot (t-2) + 3 \cdot (t-2)(t-1)$$

It is easy to verify that these polynomials are equivalent.

7 Access and Manipulation of Polynomials in Newton Form

We would like to show that Newton polynomials have properties desirable for numerical analysis, particularly:

1. Newton polynomials are easy to evaluate. That is, the computational cost of evaluating the polynomial at a point does not grow too quickly with respect to the degree.
2. Newton Polynomials are easy to differentiate. That is, the computational cost of evaluating the derivative at a point does not grow too quickly with respect to the degree of the polynomial.

7.1 Evaluating Newton Polynomials

Consider the following example.

Example 7.1. What is the value of the polynomial

$$P(t) = 2 + 3(t-4) - 5(t-4)(t+7)$$

at the point $t = 11$? The parameters of this problem are:

$$\begin{array}{lr} 2 \textit{ centers}: & 4, -7 \\ \hline \text{and } 3 \textit{ coefficients}: & 2, 3, -5 \end{array}$$

Note that the number of coefficients is always one more than the number of centers. The order in which centers and coefficients appear is important.

It is inefficient to simply plug in the numbers and compute the operations. We now quantify how the number of operations grows as size of the polynomial increases.

The following table shows the number of multiplications required for each term.

polynomial:	$P(t) = 2$	$+ 3(t-4)$	$- 5(t-4)(t+7)$
number of multiplications:	0	1	2

In general, when the degree of the polynomial is n we have

$$0 + 1 + 2 + \ldots + n = \frac{n(n+1)}{2}$$

multiplications. (We are counting only the number of multiplications because the number of additions/subtractions is almost the same; hence we can get the approximate total number of arithmetic operations by simply doubling the multiplication count.) This "brute force" method is inefficient.

Observing that $(t-4)$ appears as a factor in two terms in most of the terms, we can rewrite the polynomial as

$$P(t) = 2 + (t-4)[3 - 5(t+7)].$$

Notice that $P_1(t) = 3 - 5(t+7)$ is a Newton polynomial itself. The parameters of $P_1(t)$ are the same as before, but crossing out the leftmost ones

$$\begin{array}{r} \cancel{4},\ -7 \\ \hline \cancel{2},\ 3,\ -5 \end{array}$$

We can rewrite the original polynomial as $P_2(t) = 2+(t-4)P_1(t)$. This did not happen by accident. In fact suppose we have one more term in the polynomial, expressed in the center/coefficient table as follows:

$$\begin{array}{r} 4,\ -7,\ 8 \\ \hline 2,\ 3,\ -5,\ 6 \end{array}$$

Now the whole polynomial would be $P_3(t) = 2 + (t-4)P_2(t)$ (i.e. only the index of the polynomial is shifted). This indicates that each additional term requires just one more multiplication. So the number of multiplications is linear in the number of terms (and, at worst, linear in the degree of the polynomial). *The operation count grows like n rather than n^2.*

$\square$

Now we are ready to consider the general case. The Newton polynomial of degree n is

$$P_n(t) = c_0 + c_1(t-x_0) + \ldots + c_n(t-x_0)\cdots(t-x_n)$$

and its parameters are

$$\begin{array}{lr} n \textit{ centers}: & x_0,\ x_1,\ \ldots,\ x_{n-1} \\ \cline{2-2} \text{and } n+1 \textit{ coefficients}: & c_0,\ c_1,\ c_2,\ \ldots,\ c_n \end{array}$$

If we delete the first two parameters we have that the parameters of

$$P_{n-1}(t) = c_1 + c_2(t-x_1) + \ldots + c_n(t-x_1)\cdots(t-x_{n-1})$$

are

$$\begin{array}{r} x_1,\ \ldots,\ x_{n-1} \\ \hline c_1,\ c_2,\ \ldots,\ c_n \end{array}$$

In general

$$P_{n-j}(t) = c_j + c_{j+1}(t-x_j) + \ldots + c_n(t-x_j)\cdots(t-x_{n-1})$$

has parameters

$$\begin{array}{r} x_j,\quad \ldots,\ x_{n-1} \\ \hline c_j,\ c_{j+1},\ \ldots,\ c_n \end{array}$$

The algorithm to evaluate the polynomial is

```
1.    $P_0(t) = c_n$
2.    for $j = n-1, \ldots, 0$
```

```
3.        $P_{n-j}(t) = c_j + (t - x_j)P_{n-j-1}(t)$
```

We call this technique "*nested multiplication*".

EXAMPLE 7.2. *Consider the Newton polynomial* $P(t) = 3 - 2(t-5) + 7(t-5)(t+11)$ *whose parameters are*

$$\begin{array}{c} \qquad\qquad x_0 = 5, \quad x_1 = -11 \\ \hline c_0 = 3,\, c_1 = -2,\, c_2 = 7 \end{array}$$

We want to evaluate $P(t)$ *at* $t = 4$, *which means we want to compute* $P_2(4)$. *The steps are*

1. $P_0(4) = 7$

2. $P_1(4) = -2 + (4+11)7 = 103$

3. $P_2(4) = 3 + (4-5)103 = -100$

7.2 Differentiating Newton Polynomials

We are going to show how to evaluate the derivative of a Newton polynomial efficiently. Consider the following example.

EXAMPLE 7.3. *Find the derivative of the following cubic polynomial*

$$P(t) = 3 - 2(t-5) + 7(t-5)(t+11) + 9(t-5)(t+11)(t-1)$$

and evaluate it at a certain point.

Just finding the derivative symbolically could be difficult if we use the product rule. In fact, for the last term we would have to take the derivative of $(t-5)$ and multiply by the other 2 terms, then add the derivative of $(t+11)$ times the other 2 terms, and so on... You can imagine that as the degree of the polynomial grows, this task quickly becomes quite difficult (or at least time-consuming).

Fortunately, there is a method that makes this task easier. The idea is to look at the recurrence relation defining the Newton representation and look at what happens when we differentiate it. Recall that

$$P_n(t) = c_0 + (t - x_0)P_{n-1}(t)$$

When c_0 is the first coefficient, and x_0 the first center. Differentiating on both sides we get

$$P_n'(t) = P_{n-1}(t) + (t - x_0)P_{n-1}'(t)$$

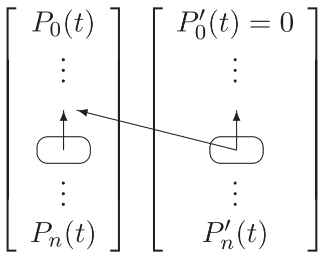
We can easily compute $P_n'(t)$ if we already know $P_{n-1}(t)$ and $P_{n-1}'(t)$.

The algorithm is

```
1.    $P_0(t) = c_n$
2.    $P_0'(t) = 0$
3.    for $j = n-1, \ldots, 0$
4.        $P_{n-j}(t) = c_j + (t - x_j)P_{n-j-1}(t)$
```

5. $\qquad P'_{n-j}(t) = P_{n-j-1}(t) + (t - x_j)P'_{n-j-1}(t)$

In other words, we are building the following two vectors starting from the top. The arrows indicate the dependences.



Note that the coefficients do not appear explicitly in the formula of the derivatives (vector on the right). However, the coefficients are used in the computation of the polynomials (vector on the left), and the polynomials are used in the computation of the derivatives. So the derivatives still depend on the coefficients.

Note that this method is very efficient. Once we have calculated the values of $P_i(t)$ (at a cost of n multiplications), we need only one multiplication per term to evaluate the derivative, for a total of $2n$ multiplications.

On the other hand, it is not easy to integrate Newton polynomials, but later in the course we will see some efficient methods for integrating functions and these methods will work for Newton polynomials.

8 Error Analysis of Polynomial Interpolation

It is very important to understand the error in polynomial interpolation. Given a function $f : [a, b] \to \mathbb{R}$ and a set of points $X \subset [a, b]$, a polynomial interpolant has the property that the value of the polynomial at the points X is the same as the function. We denote this as

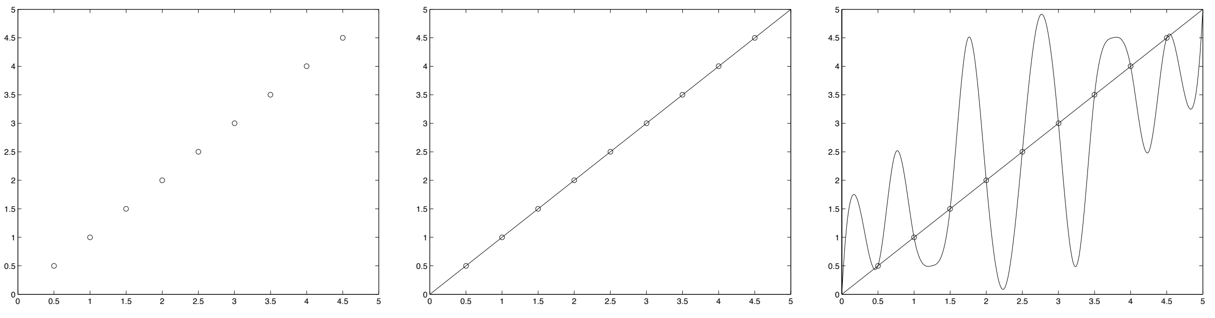
$$P|_X = f|_X.$$

But what happens at the remaining points in $[a, b]$? Error analysis gives an answer to this question.

Example 8.1. *The problem is that we can always "cheat". Suppose that we are given the points of Figure 5a. The points are aligned and the solution is the linear polynomial of Figure 5b (note that for aligned points the polynomial is linear regardless of the number of points). However, knowing the points, we could decide to choose a very bad function for which polynomial interpolation behaves extremely poorly, as in Figure 5c.*

The way to recognize this kind of function is to look at oscillations in the function f. The first derivative of the function of the previous example oscillates rapidly between positive and negative values. Now, consider another example.

Example 8.2. *The first derivative of the function in Figure 6 also oscillates, but the interpolation works pretty well in this case.*



(a) (b) (c)

Figure 5: Example of bad polynomial interpolation

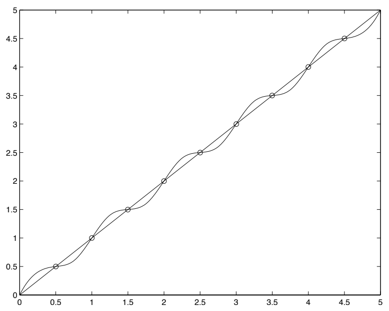


Figure 6: Example of good polynomial interpolation

From Example 8.2 we can see that looking only at the first derivative we might not get enough information. We may have to look at higher-order derivatives. In the "bad" example, the higher derivatives also gyrate wildly, while in the "good" example they do not fluctuate as much.

The general formula for calculating the error of a polynomial interpolant P is given by

$$E(t) = f(t) - p_n(t) = \frac{f^{(n+1)}(c)}{(n+1)!} \cdot \underbrace{\left(\prod_{i=0}^{n}(t - x_i)\right)}_{*} \tag{17}$$

where c lies somewhere in the interval of $X \cup \{t\}$.

Note that to do the error analysis we do not need to know the interpolating polynomial. We just need to know the derivative of the function and the position of the points. Also note that we are not going to compute the exact value of the error (which we can't do anyway, since we don't know the precise value of c), but we are looking for a good bound.

Example 8.3. *Suppose that we are given the function $f(t) = \sin t$ and the points $X = (.5, 1.5, 2.5)$. We want to estimate $f(1) - p(1)$, i.e., we want to estimate the error at the point $t = 1$.*

Using the error formula, we calculate that

$$f(1) - P(1) = \frac{f^{(3)}(c_t)}{3!}(t - .5)(t - 1.5)(t - 2.5)$$

where the point c_t depends on the value of t. In other words we don't know exactly where c_t is, but we know that it lies in the interval containing X. In this case we consider $c_t \in [.5, 2.5]$.

The 3rd derivative of $\sin t$ *is* $-\cos t$*, which is bounded in absolute value by 1, so*

$$|f^{(3)}(c_t)| \leq 1$$

So we can estimate that

$$|f(1) - P(1)| \leq \left| \frac{1}{3!}(1 - .5)(1 - 1.5)(1 - 2.5) \right| = \frac{1}{16}$$

Note that, since we do not know the precise value of c_t, this is almost certainly an overestimate of the exact error. However, without more knowledge, this is the best we can do.

Note that the product (*) in the formula (17) ensures that the error is zero at each of the interpolation points and would be the next Newton polynomial (N_{n+1}), if another interpolation point were added.

Does it make sense that the derivative order ($n+1$) that is involved in the error formula increases with the number of interpolation points? First, with some (quite limited) effort, one can show that in case f itself is a polynomial in Π_n, we have $p_n = f$, and the error in this case is 0. The error formula validates that fact directly, since $f^{(n+1)} = 0$, for every $f \in \Pi_n$.

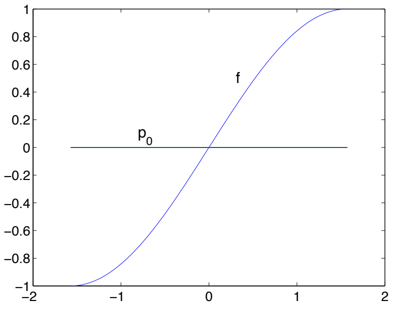


Figure 7: The Error of a Degree Zero Polynomial Interpolant over $[a, b]$

One can investigate this issue from a more geometrical point of view. Consider a degree 0 polynomial interpolating some function f, as shown in Figure 7. The difference between p_0 and f is determined by how quickly f increases (or decreases), and is thus encapsulated in the magnitude of first derivative (as well as the distance from the interpolation point).

Consider now, instead, the error committed when the interpolant is a degree 1 polynomial, as shown in Figure 8. In this case, the first derivative does not capture the difference between p_1 and f, since a large first derivative over $[a, b]$ can be approximated well by a line with large slope. In this case, the concavity of the function is the main source of the error, so that the second derivative describes how well the polynomial interpolant fits the function. Indeed, we have here $n + 1 = 2$, and the error formula, thus, involves in this case the second derivative.

As we shall see, there are two main ways that polynomial interpolation error can become unmanageable.

1. The function f that we are interpolating may simply be a bad function - it may not be differentiable sufficiently many times (i.e., $n + 1$ times for $n + 1$ interpolation points) over

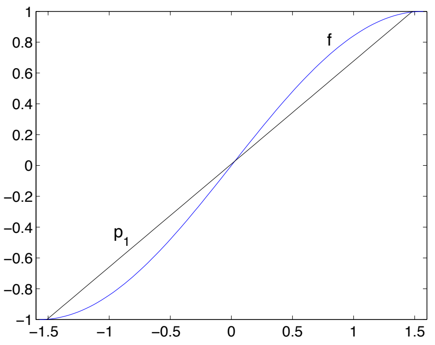
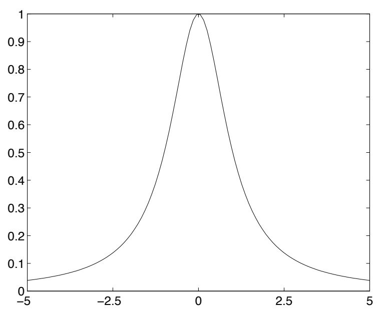
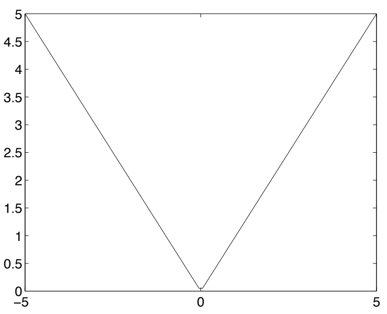


Figure 8: The Error of a Degree One Polynomial Interpolant over $[a, b]$



Runge's Example



The Absolute Value Function

Figure 9: Two Examples of Bad Functions for Polynomial Interpolation

the interval $[a, b]$. Note that, even if this lack of differentiability is due to a single bad point, the error could be large over the entire interval (not just near the "bad" point). An example of this is the absolute value function, $f(t) = |t|$ over any interval containing 0, as shown in Figure 9

We may be able to minimize the global effects of a "bad" point like this through different interpolation methods, or through careful selection of the interpolation points, but we will never be able to fully overcome the error near the "bad" point. You may notice the recurring theme - "bad" functions may fail our numerical algorithms, no matter how sophisticated the methods we use.

2. The maximal magnitude of the derivative $f^{(n)}$ of the function f may grow very fast as we increase n. Then, the more interpolation points we choose for such functions, the worse the error may become. The function f need not be very complex for this to occur, for example Runge's "bell" function (as shown in Figure 9)

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

is quite regular, a simple quotient of polynomials, but the error of the polynomial interpolant of this function over an interval like $[5, -5]$ gets larger as the number of interpolation points goes up. We can control the error somewhat through careful selection of the interpolation points, but for an effective interpolation, the winning solution is to change our methodology altogether.

8.1 Selection of Interpolation Points

If we examine the error formula for polynomial interpolation over an interval $[a, b]$:

$$E(t) = f(t) - p_n(t) = \frac{f^{(n+1)}(c)}{(n+1)!} \cdot \underbrace{\left(\prod_{i=0}^{n}(t - x_i)\right)}_{*} \qquad \text{(for some } c \in [a, b]) \tag{18}$$

we see that we are likely to reduce the error by selecting interpolation points $x_0, x_1, \ldots, x_n$ so as to minimize the product (*) (why only “likely”? as we change the interpolation points, we change also the locations c where the derivative is evaluated; thus that part in the error also changes, and that change is a “black hole” to us: we never know what the correct value of c is, but only that c is *somewhere* in the interval $[a, b]$).

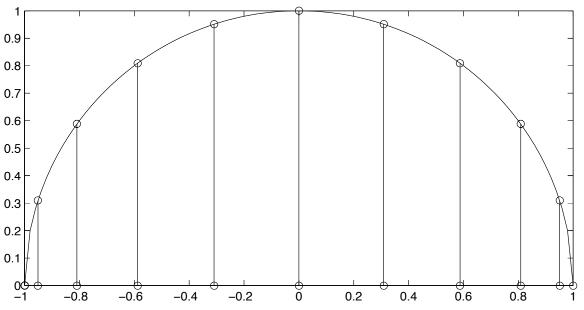


Figure 10: Chebyshev Point Distribution

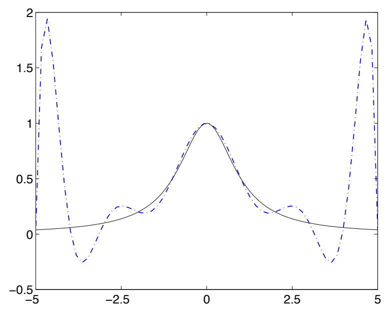
To do this, conceptually, we would like to take many points near the endpoints of the interval and few near the middle. The point distribution that minimizes the maximum value of product (*) is called the *Chebyshev distribution*, as shown in Figure 10. In the Chebyshev distribution, we proceed as follows:

1. Draw the semicircle on $[a, b]$.

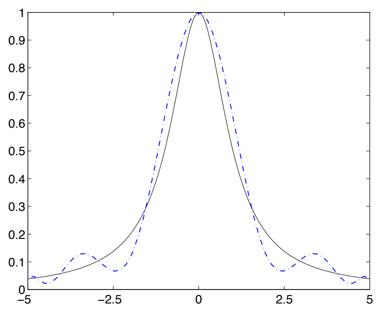
2. To sample $n + 1$ points, place n equidistant partitions on the arc.

3. Project each partition onto the x-axis:

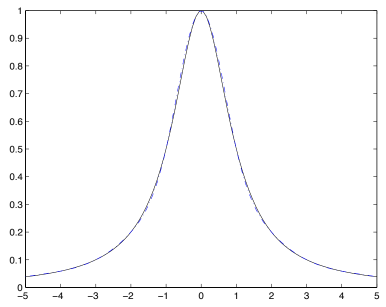
$$x_j = \frac{a+b}{2} + \frac{b-a}{2} \cdot \cos\left(j \cdot \frac{\pi}{n}\right) \qquad \text{(for } j = 0, 1, \ldots, n)$$



(a) Polynomial Interpolation (Equidistant points)



(b) Polynomial Interpolation (Chebyshev points)



(c) Interpolation Using Splines

Figure 11: Comparison of Interpolation Methods on Runge's Example.

8.2 Comparison of Interpolation Methods

How big an effect can the selection of points have? Figure 11 shows Runge's "bell" function interpolated over $[-5, 5]$ using equidistant points, points selected from the Chebyshev distribution, and a new method called *splines*. The polynomial interpolation using Chebyshev points does a much better job than the interpolation using equidistant points, but neither does as well as the use of splines, which we discuss in the next unit.