Chapter 2: Polynomial Interpolation

FuSen Lin

Department of Computer Science and Engineering
National Taiwan Ocean University

Scientific Computing, Fall 2011

2 Polynomial Interpolation

2.0 Three Problems for Data Fitting

- Purpose: Given a set of (experimental) data (or points) $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, we try to create a *mathematical model* (formula or function), and then to estimate reasonable values of some points that are not in this set.
- The **first problem**: For the given data, is it possible to find a a simple and convenient formula that represents the given points exactly? —Using Polynomial Interpolation

2.0 Three Problems for Data Fitting

- The **second problem** is similar to the first one, but the given data (usually from experiments) are *possibly contaminated by errors*. Now we ask for a formula that represents the data approximately and, if possible, filters out the errors. —Using the Least Squares method
- The **third problem** is that a function f is given, perhaps in the form of a computer precedure, but it is expensive to evaluate it directly. In this case, we seek for another function g that is simpler (cheaper) to evaluate and produces a reasonable approximation to f. —Using Polynomial or Spline approximation

2.0 Introduction to Interpolation

- The problem of **data approximation:** given some points $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ and are asked to find a function $\phi(x)$ that "captures the trend" of the data.
- If the *trend is one of decay*, the function ϕ may be the form

$$\phi(x) = a_1 e^{-\lambda_1 x} + a_2 e^{-\lambda_2 x}.$$

- If the trend of the data is *oscillatory*, then a trigonometric approximant might be appropriate:

$$\phi(x) = \lambda_1 \sin \alpha_1 x + \lambda_2 \cos \alpha_2 x.$$

- Other settings may require a low-degree **polynomial**

$$\phi(x) = a_1 + a_2 x + a_3 x^2 + \cdots + a_n x^{n-1}$$

Polynomial Interpolation

- If the function $\phi(x)$ actually *"goes through" the data*, this means that
$$\phi(x_i) = y_i, \quad i = 1 : n,$$
we say that ϕ interpolates the data and ϕ is called an **interpolant** of the data.
- The **polynomial interpolation** is simple and particularly important:
- Given the data $x_1, ..., x_n$ (distinct) and $y_1, ..., y_n$, find a polynomial $p_{n-1}(x)$ of degree $n - 1$ (or less) such that $p_{n-1}(x_i) = y_i$ for $i = 1 : n$.

Continuing

- How to *represent* the interpolating polynomial $p_{n-1}(x)$?
- How to determine the associated *coefficients*?
- After we have obtained the coefficients, how can the interpolant be *evaluated* (at other values of x) with efficiency?

Continuing

- In MATLAB these issues can be handled by the build-in functions **polyfit** and **polyval**. The syntax:

$$a = \text{polyfit}(x, y, n-1)$$

and

$$yvalues = \text{polyval}(a, xvalues)$$

- **Example:** To interpolate these points $(-2, -15), (3, -5)$, and $(1, 3)$ and then evaluate the interpolant at the 100 values on $[-3, 2]$.

2.1 The Vandermonde Approach

- The interpolating polynomial is a *linear combination* of the set $\{1, x, x^2, x^3, ...\}$.
- An **example**–A four-point Interpolation Problem:
- let us find a **cubic polynomial**

$$p_3(x) = a_1 + a_2x + a_3x^2 + a_4x^3$$

that interpolates the data $(-2, 10), (-1, 4), (1, 6)$ and $(2, 3)$.

$$\begin{aligned}
P_3(-2) = 10 &\implies a_1 - 2a_2 + 4a_3 - 8a_4 = 10 \\
P_3(-1) = 4 &\implies a_1 - a_2 + a_3 - a_4 = 4 \\
P_3(1) = 6 &\implies a_1 + a_2 + a_3 + a_4 = 6 \\
P_3(2) = 3 &\implies a_1 + 2a_2 + 4a_3 + 8a_4 = 3
\end{aligned}$$

(Continuing)

- Expressing these four equations in *matrix-vector form* gives

$$\begin{bmatrix} 1 & -2 & 4 & -8 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix} = \begin{bmatrix} 10 \\ 4 \\ 6 \\ 3 \end{bmatrix}$$

- The solution $a = [4.5000, 1.9167, 0.5000, -0.9167]$ to this system can be found by a common solver of linear systems $a = V\backslash y$.

The General n case

- The **polynomial interpolation** problem reduces to a **linear equation problem**.
- For general n, the goal is to determine the coefficients $a_1, a_2, \ldots, a_n$ so that $p_{n-1}(x_i) = y_i$ for all $i = 1 : n$, where

$$p_{n-1}(x) = a_1 + a_2 x + a_3 x^2 + \cdots + a_n x^{n-1}$$

- Writing these equations in **matrix-vector form**, we obtain

$$\begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{n-1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^{n-1} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

The coefficient matrix is called Vandermonde matrix.

Designate the Coefficient Matrix

- Designate the matrix of coefficients by V. If V is **nonsingular** then the above system is solvable.
- Setting Up and Solving the System:
  - A conventional **double-loop approach**: algorithm that operate on a 2-D array in row-by-row fashion are *row oriented*.
  - The inner loop can be vectorized because MATLAB supports *point-wise exponentiation*. For example,

$$u = [1,2,3,4].^{\wedge}[3,5,2,3] = [1,32,9,64]$$

  - A **column oriented algorithm**
  - Pointwise vector multiplication: $V(:,j) = x.*V(:,j-1)$.
- *Column-oriented and matrix-vector implementations* will generally be favored in MATLAB.

Nested Multiplication–Horner's Algorithm

- To evaluate the value of $p_{n-1}(x)$ at some points $x = z$ (z may be a vector). It is better to use **Horner's algorithm**.
- an example for the case $n = 4$

$$p_3(x) = a_1 + a_2 x + a_3 x^2 + a_4 x^3 = a_1 + x(a_2 + x(a_3 + x(a_4)))$$

- In general case n,

$$\begin{aligned} p_3(x) &= a_1 + a_2 x + \cdots + a_n x^{n-1} \\ &= a_1 + x(a_2 + \cdots + x(a_{n-1} + x(a_n)) \cdots) \end{aligned}$$

2.2 The Newton Representation

- Consider once again the problem of interpolating the four points (x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4) with a cubic polynomial $p_3(x)$.
- However, instead of expressing the interpolant in terms of the "canonical" basis $\{1, x, x^2, x^3\}$, we use the basis $\{1, (x - x_1), (x - x_1)(x - x_2), (x - x_1)(x - x_2)(x - x_3)\}$ and looking for the coefficients c_1, c_2, c_3, and c_4 so that if

$$p_3(x) = c_1 + c_2(x - x_1) + c_3(x - x_1)(x - x_2) + c_4(x - x_1)(x - x_2)(x - x_3)$$

then $p_3(x_i) = y_i$ for $i = 1 : 4$. This expression is called the **Newton representation of the interpolating polynomial**.

(Continuing)

- In expanded form:

$$
\begin{aligned}
y_1 &= c_1 \\
y_2 &= c_1 + c_2(x_2 - x_1) \\
y_3 &= c_1 + c_2(x_3 - x_1) + c_3(x_3 - x_1)(x_3 - x_2) \\
y_4 &= c_1 + c_2(x_4 - x_1) + c_3(x_4 - x_1)(x_4 - x_2) \\
&\quad + c_4(x_4 - x_1)(x_4 - x_2)(x_4 - x_3)
\end{aligned}
$$

(Continuing)

- It can be expressed these equations in matrix-vector form:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & (x_2 - x_1) & 0 & 0 \\ 1 & (x_3 - x_1) & (x_3 - x_1)(x_3 - x_2) & 0 \\ 1 & (x_4 - x_1) & (x_4 - x_1)(x_4 - x_2) & (x_4 - x_1)(x_4 - x_2)(x_4 - x_3) \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}$$

- This linear system can **be reduced to 3-by-3 system** and solved by **Gaussian Elimination**.

Recursive Algorithm for Newton Interpolation

- For general n, we see that if $c_1 = y_1$ and

$$q(x) = c_2 + c_3(x - x_2) + \cdots + c_n(x - x_2)\cdots(x - x_{n-1})$$

interpolates the data

$$\left(x_i, \frac{y_i - y_1}{x_i - x_1}\right) \quad i = 2 : n,$$

then

$$p(x) = c_1 + (x - x_1)q(x)$$

interpolates (x_1, y_1),..., (x_n, y_n).

Nonrecursive Algorithm for Newton Interpolation

```
for k=1:n-1
   c(k) = y(k);
   for j = K+1:n
      Substract Eq. k from Eq. j and divide the result
      by (x(j) - x(k))
   end
end
c(n) = y(n);
```

Nonrecursive Algorithm (Continuing)

- Notes: When updating the equations we need only *keep track of the changes* in the y-vector.

$$y(k+1:n) = (y(k+1:n) - y(k))./(x(k+1:n) - x(k)).$$

2.3 Properties

- In scientific computing, **efficiency** and **accuracy** are the main concerns.
- **Efficiency** includes the execution time efficiency (speed) and memory efficiency (how much memory space needed).
- The execution time can be estimated by running time, but it *depends on what kind of the machine* and has no formulas to express.
- Alternatively, to estimate the execution time of a program can count its total **flops** or the order of flops versus the number of n.

Comparison of Efficiency

- The **Vandermonde approach** involves solving an n-by-n linear system and requires $2n^3/3$ **flops**, which is $O(n^3)$.
- The **Newton method** requires $3n^2/2$ **flops** which is of quadratic order (i.e., $O(n^2)$).
- The recursive algorithm of the Newton interpolation is *faster* than the nonrecursive one if n is *not so large*.
- However, if n is getting larger it is getting slower due to the access time (it needs more memory space: it requires *a couple of n-vectors with each level of the recursion*.

Accuracy

- The polynomial interpolant exists and unique, but *how well* does it approximate? It depends on the derivatives of the function that is being interpolated.
- Theorem 2: Suppose $p_{n-1}(x)$ interpolates the function $f(x)$ a t the distinct points x_1,..., x_n. If f has n-th continuous derivatives on an interval I containing the x_i, then for any $x \in I$

$$f(x) = p_{n-1}(x) + \frac{f^{(n)}(\eta)}{n!}(x - x_1) \cdots (x - x_n),$$

where $a \leq \eta \leq b$.
- Suppose $|f^{(n)}(x)| \leq M_n$ for all $x \in [a, b]$. Then for any $z \in [a, b]$, we have

$$|f(z) - p_{n-1}(z)| \leq \frac{M_n}{n!} \max_{a \leq x \leq b} |(x - x_1) \cdots (x - x_n)|.$$

Accuracy (continuing)

- If the interpolation is based on the equally spaced points

$$x_i = a + \frac{b-a}{n-1}(i-1) := a + (i-1)h, \quad h = \frac{b-a}{n-1}$$

(h is called the step size) for $i = 1 : n$ then

$$|f(z) - p_{n-1}(z)| \le M_n \cdot h^n \max_{0 \le s \le n-1} \left| \frac{s(s-1)\cdots(s-n+1)}{n!} \right| .$$

- It can be shown that the **max** $\le 1/(4n)$ and

$$|f(z) - p_{n-1}(z)| \le \frac{M_n}{4n} h^n = \frac{M_n}{4n} \left(\frac{b-a}{n-1} \right)^n = O\left(\frac{1}{n^{n+1}} \right)$$

Accuracy (continuing)

- A failure example of interpolating the function

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

with *equally spaced points* across the interval $[-1, 1]$ (see figure).

- The polynomial interpolant captures the trend of the function in the *middle part* of the interval, but it *blow up near the endpoints*. This is the so-called Gibb's phenomenon.

2.4 Special Topics

- (1) Divided Differences
- (2) Inverse Interpolation
- (3) 2-D Linear Interpolation
- (4) Trigonometric Interpolation

Divided Differences

- The coefficients of the Newton form of polynomial interpolation can be expressed by divided differences. For $n = 4$ example:

$$p_3(x) = c_1 + c_2(x - x_1) + c_3(x - x_1)(x - x_2) + c_4(x - x_1)(x - x_2)(x - x_3)$$

$$\begin{aligned}
c_1 &= f(x_1) := f[x_1] \\
c_2 &= \frac{f(x_2) - f(x_1)}{x_2 - x_1} = \frac{f[x_2] - f[x_1]}{x_2 - x_1} := f[x_1, x_2] \\
c_3 &= \frac{\frac{f(x_3)-f(x_2)}{x_3-x_2} - \frac{f(x_2)-f(x_1)}{x_2-x_1}}{x_3 - x_1} = \frac{f[x_2, x_3] - f[x_1, x_2]}{x_3 - x_1} := f[x_1, x_2, x_3] \\
c_4 &= \frac{f[x_2, x_3, x_4] - f[x_1, x_2, x_3]}{x_4 - x_1} := f[x_1, x_2, x_3, x_4]
\end{aligned}$$

- The coefficients are called **divided differences**.

Divided Differences (continuing)

- In general, the coefficients of the Newton interpolating polynomial are denoted as

$$c_k := f[x_1, x_2, \ldots, x_k] = \frac{f[x_2, \ldots, x_k] - f[x_1, \ldots, x_{k-1}]}{x_k - x_1}.$$

which is called the $(k-1)$th order divided differences and has the recurrence form.

- Thus, the **Newton interpolating polynomial** can be expressed as

$$p_{n-1}(x) = \sum_{k=1}^{n} f[x_1, \ldots, x_k] \left(\prod_{j=1}^{k-1} (x - x_j) \right)$$

Inverse Interpolation

- Suppose the function $f(x)$ has an inverse on $[a, b]$. That is, there exists a function g so that $g(f(x)) = x$ for all $x \in [a, b]$.
- If
$$a = x_1 < x_2 < \cdots < x_n = b$$
and $y_i = f(x_i)$, then the polynomial that *interpolates the data (y_i, x_i), $i = 1 : n$* is an Interpolant of the inverse function g. This is called **inverse interpolation**.
- Example: $g(x) = \sqrt{x}$ is the inverse of $f(x) = x^2$ on [0, 1].

2-D Linear Interpolation

- Suppose $(\tilde{x}, \tilde{y})$ is inside the rectangle

$$R = \{(x, y) : a \le x \le b, \quad c \le y \le d\},$$

and $f(x, y)$ is defined on R. The values of its four corners are known:

$$f_{ac} = f(a, c), \quad f_{bc} = f(b, c), \quad f_{ad} = f(a, d), \quad f_{bd} = f(b, d).$$

- Our goal is to use linear interpolation to estimate the value of $f(\tilde{x}, \tilde{y})$: Suppose $\lambda \in [0, 1]$ with the property that $\tilde{x} = (1 - \lambda)a + \lambda b$. It follows that

$$f_{xc} = (1 - \lambda)f_{ac} + \lambda f_{bc}, \quad f_{xd} = (1 - \lambda)f_{ad} + \lambda f_{bd}$$

are linearly interpolated estimates $f(\tilde{x}, c)$ and $f(\tilde{x}, d)$, respectively.

2-D Linear Interpolation (continuing)

- Consequently, if $\mu \in [0,1]$ with $\tilde{y} = (1-\mu)c + \mu d$, then a second interpolation between f_{xc} and f_{xc} gives an estimate of $f(\tilde{x}, \tilde{y})$:
$$z = (1-\mu)f_{xc} + \mu f_{xd} \approx f(\tilde{x}, \tilde{y}).$$
- Putting it all together, we have

$$\begin{aligned} z &= (1-\mu)[(1-\lambda)f_{ac} + \lambda f_{bc}] + \mu[(1-\lambda)f_{ad} + \lambda f_{bd}] \\ &\approx f((1-\lambda)a + \lambda b,\ (1-\mu)c + \mu d) \end{aligned}$$

Trigonometric Interpolation

- Let $f(t)$ is a periodic function with period T, $n = 2m$, and we want to interpolate the data (t_0, f_0),..., (t_n, f_n) where $f_k = f(t_k)$ and $t_k = kT/n$ for $k = 0 : n$.
- If the data is periodic, it is better to interpolate with a periodic function rather than with a polynomial.
- We shall use the linear combination of the functions

$$\cos(2\pi jt/T), \quad \sin(2\pi jt/T), \quad j \in \text{integer}$$

(these two functions have the same period).
- We seek real scalars $a_0, \ldots, a_m$ and $b_0, \ldots, b_m$ so that if

$$F(t) = \sum_{j=0}^{m} a_j \cos\left(\frac{2\pi j}{T} t\right) + b_j \sin\left(\frac{2\pi j}{T} t\right),$$

then $F(t_k) = f_k$ for $k = 0 : n$. The function $F(t)$ is called the trigonometric interpolant.

Trigonometric Interpolation (continuing)

- This forms a linear system that consists of $n+1$ equations in $2(m+1)=n+2$ unknowns. However, we do not need b_0 and b_m since $\sin(2\pi jt/T)=0$ if $t=0$ or $t=T$. Moreover, the $k=0$ equation and the $k=n$ equation are identical because of periodicity.
- We really want to determine $a_0,\ldots,a_m$ and $b_1,\ldots,b_{m-1}$ so that if

$$F(t)=a_0+\sum_{j=1}^{m-1} a_j\cos\left(\frac{2\pi j}{T}t\right)+b_j\sin\left(\frac{2\pi j}{T}t\right)+a_m\cos\left(\frac{2\pi m}{T}t\right),$$

then $F(t_k)=f_k$ for $k=0:n-1$.
- This is an n-by-n linear system in n unknowns:

$$f_k=a_0+\sum_{j=1}^{m-1} a_j\cos\left(\frac{kj\pi}{m}\right)+b_j\sin\left(\frac{kj\pi}{m}\right)+(-1)^k a_m,\quad k=0:n-1.$$

Trigonometric Interpolation (Continuing)

- For the $n = 6$ example, these equations form the n-by-n linear system:

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1/2 & -1/2 & -1 & \sqrt{3}/2 & \sqrt{3}/2 \\ 1 & -1/2 & -1/2 & 1 & \sqrt{3}/2 & -\sqrt{3}/2 \\ 1 & -1 & 1 & -1 & 0 & 0 \\ 1 & -1/2 & -1/2 & 1 & -\sqrt{3}/2 & \sqrt{3}/2 \\ 1 & 1/2 & -1/2 & -1 & -\sqrt{3}/2 & -\sqrt{3}/2 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ b_1 \\ b_2 \end{bmatrix} = \begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \end{bmatrix}$$

Trigonometric Interpolation (Continuing)

- Notes: 1. The matrix of coefficients P can be shown to be nonsingular (its inverse matrix P^{-1} exists) so that the interpolation process that we have presented is well-defined.
- 2. Solving the linear system involves $O(n^3)$ flops. In Problem 2.4.7 (on p. 103) we show how to reduce this to $O(n^2)$, since P has the property that $P^T P$ is diagonal (can you prove it?).
- 3. Moreover, if apply the fast Fourier transform, then the flop count can be reduce further to an amazing $O(n\log(n))$ (see Problem 5.4.2, p. 200).

Trigonometric Interpolation (Continuing)

- The test problem is to interpolate the following ascension-declination data

α	0	30	60	90	120	150	180	210	240	270	300	330
d	408	89	-66	10	338	807	1238	1511	1538	1462	1183	804

- With a function of the form

$$d(\alpha) = a_0 + \sum_{j=1}^{5} a_j \cos\left(\frac{2\pi j\alpha}{360}\right) + b_j \sin\left(\frac{2\pi j\alpha}{360}\right) + a_6 \cos\left(\frac{12\pi\alpha}{360}\right).$$