Chapter 4 Interpolation Schemes

Numerical Analysis 1. Winter Semester 2018-19

1 Polynomial interpolations

1.1 Langrange interpolation

Suppose that we wish to interpolate arbitrary functions at a set of fixed nodes $x_0, x_1, \ldots, x_n$. We first define a system of $n+1$ special polynomials of degree n known as **cardinal polynomials** in interpolation theory. These are denoted by $\ell_0, \ell_1, \ldots, \ell_n$ and have the property

$$\ell_i(x_j) = \delta_{ij} = \begin{cases} 0 & \text{if } i \neq j \\ 1 & \text{if } i = j \end{cases}$$

Once these are available, we can interpolate *any* function f by the **Lagrange form of the interpolation polynomial**:

$$p_n(x) = \sum_{i=0}^{n} \ell_i(x) f(x_i) \tag{1}$$

This function p_n, being a linear combination of the polynomials ℓ_i, is itself a polynomial of degree at most n. Furthermore, when we evaluate p_n at x_j, we get $f(x_j)$:

$$p_n(x_j) = \sum_{i=0}^{n} \ell_i(x_j) f(x_i) = \ell_j(x_j) f(x_j) = f(x_j)$$

Thus, p_n is the interpolating polynomial for the function f at nodes $x_0, x_1, \ldots, x_n$. It remains now only to write the formula for the **cardinal polynomial** ℓ_i, which is

$$\ell_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^{n} \left(\frac{x - x_j}{x_i - x_j} \right) \qquad (0 \leqq i \leqq n) \tag{2}$$

Notice Example with first and second order Lagrange polynomial.

1.2 Newton interpolation

Observation: The Lagrange formula is well-suited for many theoretical uses of interpolation, but it is less desirable when actually computing the value of an interpolating polynomial. As an example, knowing $P_2(x)$ does not lead to a less expensive way to evaluate $P_3(x)$, at least not in a simple manner. For this reason, we introduce an alternative and more easily calculable formulation for the interpolation polynomials $P_1(x), P_2(x), \ldots, P_n(x)$.

Idea: Newton interpolation polynomial

$$p(x) = c_0 + c_1(x - x_0) + c_2(x - x_0)(x - x_1) + \cdots + c_n(x - x_0)(x - x_1)...(x - x_n).$$

It can not be overemphasized that the Newton and Lagrange forms are just two different derivations for precisely the same polynomial. The Newton form has the advantage of easy extensibility to accommodate additional data points.

Example 1. *Using the Newton algorithm, find the interpolating polynomial of least degree for this table:*

x	0	1	-1	2	-2
y	-5	-3	-15	39	-9

The solution is $p(x) = -5 + 2x - 4x(x-1) + 8x(x-1)(x+1) + 3x(x-1)(x+1)(x-2)$.

1.3 Error in polynomial interpolation

See [**?**], page 138-140. For the behavior of the error, see page 142-143 in the same book.

Dirichlet Function

As a pathological example, consider the so-called **Dirichlet function** f, defined to be 1 at each irrational point and 0 at each rational point. If we choose nodes that are rational numbers, then $p(x) \equiv 0$ and $f(x) - p(x) = 0$ for all rational values of x, but $f(x) - p(x) = 1$ for all irrational values of x.

However, if the function f is well-behaved, can we not assume that the differences $|f(x) - p(x)|$ will be small when the number of interpolating nodes is large? The answer is still *no*, even for functions that possess continuous derivatives of all orders on the interval!

Runge Function

A specific example of this remarkable phenomenon is provided by the **Runge function**:

$$f(x) = \left(1 + x^2\right)^{-1} \tag{1}$$

on the interval $[-5, 5]$. Let p_n be the polynomial that interpolates this function at $n + 1$ equally spaced points on the interval $[-5, 5]$, including the endpoints. Then

$$\lim_{n\to\infty} \max_{-5 \leqq x \leqq 5} |f(x) - p_n(x)| = \infty$$

Figure 1: Page 154, [**?**]

2 Divided difference and its applications

We introduce a discrete version of the derivative of a function

INTERPOLATION ERRORS I

If p is the polynomial of degree at most n that interpolates f at the $n+1$ distinct nodes $x_0, x_1, \ldots, x_n$ belonging to an interval $[a, b]$ and if $f^{(n+1)}$ is continuous, then for each x in $[a, b]$, there is a ξ in (a, b) for which

$$f(x) - p(x) = \frac{1}{(n+1)!} f^{(n+1)}(\xi) \prod_{i=0}^{n} (x - x_i) \tag{2}$$

UPPER BOUND LEMMA

Suppose that $x_i = a + ih$ for $i = 0, 1, \ldots, n$ and that $h = (b-a)/n$. Then for any $x \in [a, b]$

$$\prod_{i=0}^{n} |x - x_i| \leqq \frac{1}{4} h^{n+1} n! \tag{4}$$

INTERPOLATION ERRORS II

Let f be a function such that $f^{(n+1)}$ is continuous on $[a, b]$ and satisfies $|f^{(n+1)}(x)| \leqq M$. Let p be the polynomial of degree $\leqq n$ that interpolates f at $n+1$ equally spaced nodes in $[a, b]$, including the endpoints. Then on $[a, b]$,

$$|f(x) - p(x)| \leqq \frac{1}{4(n+1)} M h^{n+1} \tag{6}$$

where $h = (b-a)/n$ is the spacing between nodes.

$$f[x_0, x_1] = \frac{f(x_1) - f(x_0)}{x_1 - x_0} \qquad \text{first-order divided difference,}$$

$$f[x_0, x_1, x_2] = \frac{f[x_1, x_2] - f[x_0, x_1]}{x_2 - x_0} \qquad \text{second-order divided difference,}$$

$$f[x_0, x_1, x_2, x_3] = \frac{f[x_1, x_2, x_3] - f[x_0, x_1, x_2]}{x_3 - x_0} \qquad \text{third-order divided difference,}$$

and so on ...

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

Example 2. *Computing the divided difference from the following table*

x	*1*	*3/2*	*0*	*2*
y	*3*	*13/4*	*3*	*5/3*

The solution is $p(x) = 3 + 1/2(x-1) + 1/3(x-1)(x-3/2) - 2(x-1)(x-3/2)x$.

x	$f[\,]$	$f[\,,\,]$	$f[\,,\,,\,]$	$f[\,,\,,\,,\,]$
x_0	$f[x_0]$			
		$f[x_0, x_1]$		
x_1	$f[x_1]$		$f[x_0, x_1, x_2]$	
		$f[x_1, x_2]$		$f[x_0, x_1, x_2, x_3]$
x_2	$f[x_2]$		$f[x_1, x_2, x_3]$	
		$f[x_2, x_3]$		
x_3	$f[x_3]$			

Figure 2: Scheme to compute divided difference [**?**]

Newton interpolation polynomial can be computed based on divided difference

$$P_n(x) = f(x_0) + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1) + \cdots + f[x_0, x_1, \ldots, x_n](x - x_0) \cdots (x - x_{n-1}).$$

Evaluating $P_n(x)$ using nested multiplication

$$P_n(x) = D_0 + (x - x_0)[D_1 + (x - x_1)[D_2 + \cdots + (x - x_{n-2})[D_{n-l} + (x - x_{n-1})D_n]...$$

Remark 1. *Properties of divided difference.*
i) Divided difference does not depend on the permutation order of the points $x_0, \ldots, x_n$.
ii) The divided difference can be extended to the case where some or all of the node points x_i *are coincident, provided that* $f(x)$ *is sufficiently differentiable.*

The proof of (4.27) is nontrivial, and we will consider only the cases $n = 1$ and $n = 2$.
For $n = 1$,

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

For $n = 2$, we can expand (4.21) to obtain

$$f[x_0, x_1, x_2] = \frac{f(x_0)}{(x_0 - x_1)(x_0 - x_2)} + \frac{f(x_1)}{(x_1 - x_0)(x_1 - x_2)} + \frac{f(x_2)}{(x_2 - x_0)(x_2 - x_1)} \tag{4.28}$$

Figure 3: Proof of property i)

$$f[x_0, x_0] = \lim_{x_1 \to x_0} f[x_0, x_1] = \lim_{x_1 \to x_0} \frac{f(x_1) - f(x_0)}{x_1 - x_0}$$
$$f[x_0, x_0] = f'(x_0)$$

For an arbitrary $n \geq 1$, let all of the nodes in (4.24) approach x_0. This leads to the definition

$$f[x_0, x_0, \ldots, x_0] = \frac{1}{n!} f^{(n)}(x_0) \tag{4.29}$$

where the left-hand side denotes an order n divided difference, all of whose nodes are x_0.

Figure 4: Property ii) is continued here.

Summary

(1) The Runge function $f(x) = 1/(1 + x^2)$ on the interval $[-5, 5]$ shows that high-degree polynomial interpolation and uniform spacing of nodes may not be satisfactory. The Chebyshev nodes for the interval $[a, b]$ are given by

$$x_i = \frac{1}{2}(a + b) + \frac{1}{2}(b - a) \cos\left[\left(\frac{2i + 1}{2n + 2}\right)\pi\right]$$

(2) There is a relationship between differences and derivatives:

$$f[x_0, x_1, \ldots, x_n] = \frac{1}{n!} f^{(n)}(\xi)$$

(3) Expressions for errors in polynomial interpolation are

$$f(x) - p(x) = \frac{1}{(n + 1)!} f^{(n+1)}(\xi) \prod_{i=0}^{n} (x - x_i)$$
$$f(x) - p(x) = f[x_0, x_1, \ldots, x_n, x] \prod_{i=0}^{n} (x - x_i)$$

(4) For $n + 1$ equally spaced nodes, an upper bound on the error is given by

$$|f(x) - p(x)| \leqq \frac{M}{4(n + 1)} \left(\frac{b - a}{n}\right)^{n+1}$$

Here M is an upper bound on $\left|f^{(n+1)}(x)\right|$ when $a \leqq x \leqq b$.

(5) If f is a polynomial of degree n, then all of the divided differences $f[x_0, x_1, \ldots, x_i]$ are zero for $i \geqq n + 1$.

2.1 Matlab code for Newton interpolation using divided difference

```
function v = div_diff(x,y)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Function div_diff is an implementation of divided difference scheme
%                   in order to compute Newton interpolation polynomial
% Form: y = div_diff(x,y)
% x: interpolation nodes
% y: value of the function at nodes in vector x
% v: Output vector, contains the coefficients of the Newton ...
    interpolation polynomial
% Copyright: Phi Ha, October 2018
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if length(x) ≠ length(y)
  error("Length of nodes mismatch")
end

n = length(x);
A = zeros(n,n);
A(:,1) = y';

for j = 2:n
  for i = j:n
    diff = x(i) - x(i-j+1);
    A(i,j) = (A(i,j-1)-A(i-1,j-1))/diff ;
  end
end

v = diag(A)';
```

```
function value = newton_interp(x,y,w)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Newton Interpolation Using Divided Difference Scheme
% Form: value = newton_interp(x,y,w)
% x: interpolation nodes
% y: value of the function at nodes in vector x
% w : point where we need to compute f(w) using Newton interpolation scheme
% value: Output interpolated value of f(w), which is P(w)
% Supporting function: div_diff
% Copyright: Phi Ha, October 2018
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

v = div_diff(x,y)
n = length(v);

value = v(n);

for i = n-1 : -1 : 1
  value = v(i) + (w-x(i)) * value;
end
```

Newton's divided-difference formula can be expressed in a simplified form when the nodes are arranged consecutively with equal spacing $h = x_{i+1} - x_i$. Using binomial-coefficient notation, we can express $P_n(x)$ compactly as

$$P_n(x) = P_n(x_0 + sh) = f[x_0] + \sum_{k=1}^{s} \binom{s}{k} k! h^k \, f[x_0, x_1, ..., x_k].$$

Then, we can have forward Newton difference, backward Newton difference, center difference (Stirling) formulae. See [**?**], p.129-133 for further details.

3 Neville interpolation

Another method of obtaining a polynomial interpolant from a given table of values was given by Neville. It builds the polynomial in steps, just as the Newton algorithm does. The constituent polynomials have interpolating properties of their own. We start with constant polynomials $P_i(x) = f(x_i)$. Selecting two nodes x_i and x_j with $i > j$, we define recursively Here, each

$$P_{u,\dots,v}(x) = \left(\frac{x - x_j}{x_i - x_j}\right) P_{u,\dots,j-1,j+1,\dots,v}(x) + \left(\frac{x_i - x}{x_i - x_j}\right) P_{u,\dots,i-1,i+1,\dots,v}(x)$$

Using this formula repeatedly, we can create an array of polynomials:

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

successive polynomial can be determined from two adjacent polynomials in the previous column.

Theorem 1. *$P_{u,\dots,v}$ is exactly the Lagrange polynomial that interpolates f at the points $x_u, x_{u+1}, \dots, x_v$.*

We can simplify the notation by letting

$$S_{i,j}(x) = P_{i-j,i-j+1,\dots,i}(x)$$

where $S_{ij}(x)$ for $i \geq j$ denotes the interpolating polynomial of degree j on the $j + 1$ nodes $x_{i-j}, \dots, x_{i-1}, x_i$. Next we can rewrite the recurrence relation above as

$$S_{i,j}(x) = \frac{x - x_{i-j}}{x_i - x_{i-j}} S_{i,j-1}(x) - \frac{x - x_i}{x_i - x_{i-j}} S_{i-1,j-1}(x) \text{ for all } i \geq j$$

then $S_{n,n}$ is exactly the value $P_{0,1,\dots,n}(x)$. The Matlab implementation of this, however, has a small modification, since the indices begin from 0.

```
function value = neville_interp(x,y,w)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Neville Interpolation Using Divided Difference Scheme
% Form: value = neville_interp(x,y,w)
% x: interpolation nodes
% y: value of the function at nodes in vector x
% w : point where we need to compute f(x) using Neville interpolation ...
    scheme
% value: Output value of f(x)
% Copyright: Phi Ha, October 2018
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if length(x) ≠ length(y)
  error("Length of nodes mismatch")
end

n = length(x);
S = zeros(n,n);
S(:,1) = y';

for j = 2:n
  for i = j:n
    diff = x(i) - x(i-j+1);
    S(i,j) = ( (w-x(i-j+1)) * S(i,j-1) - (w-x(i)) * S(i-1,j-1) )/diff;
  end
end

value = S(n,n);
```

4 What have not been covered?

In this script we have not consider the following topics.

i) Spline interpolation, see [**?**], Section 4.3 and [**?**], Chapter 9.

ii) Hermit interpolation, see [**?**], Section 3.4.

iii) Estimating derivatives and Extrapolation, see [**?**], Section 4.2.

iv) Bivariate interpolation, see [**?**], Section 4.2.