Computational methods-Lecture 2

1 Some methods to speed up computing the interpolation polynomials

The previous error bound is in the *a priori* type, namely the error involves the information of the true function, which is sometimes hard to control. Hence, we often do not know how many points we need to use. If we increase the number of points, can we use the previously computed values, avoiding starting over?

1.1 Neville's method

The Neville's method is to treat this issue and convenient for evaluating the function value at a given points.

Let $v = \{x_0, x_1, \cdots, x_n\}$ be a list of points. We let L_v be the interpolation at the points in v.

Proposition 1. *It holds that*

$$L_v(x) = \frac{(x - x_j)L_{v\backslash\{x_j\}}(x) - (x - x_i)L_{v\backslash\{x_i\}}(x)}{x_i - x_j}$$

The proof is straightforward. First of all, the degree of the polynomial is at most $n+1$. Secondly, it agrees with f at these $n+1$ points. By the uniqueness of the interpolation polynomial with degree n, the claim follows.

The following table shows how this observation can be developed into an algorithm for increasing the number of points.

x_0	$L_{\{x_0\}} = Q_{0,0}$			
x_1	$L_{\{x_1\}} = Q_{1,0}$	$L_{\{x_0,x_1\}} = Q_{1,1}$		
x_2	$L_{\{x_2\}} = Q_{2,0}$	$L_{\{x_1,x_2\}} = Q_{2,1}$	$L_{\{x_0,x_1,x_2\}} = Q_{2,2}$	
x_3	$L_{\{x_3\}} = Q_{3,0}$	$L_{\{x_2,x_3\}} = Q_{3,1}$	$L_{\{x_1,x_2,x_3\}} = Q_{3,2}$	$L_{\{x_0,x_1,x_2,x_3\}} = Q_{3,3}$

1.2 Divided differences and Newton's interpolation

The motivation of Newton's interpolation is pretty much like the Neville's method, i.e. we want to use the results for the previous low order polynomials, avoiding recomputing the polynomials altogether. However, this is better for computing the coefficients of the polynomial, instead of the values.

The idea is as follows.

First of all, we use one point, only, then we have

$$P_0(x) = f(x_0).$$

The, we add x_1, the polynomial has degree 1, but clearly $P_1(x) - P_0(x)$ is zero at x_0. This means $(x - x_0)|P_1 - P_0$, and thus

$$P_1(x) = f(x_0) + a_1(x - x_0).$$

Similarly, if we add x_2, it is a polynomial with degree 2, but $(x - x_0)(x - x_1)|P_2 - P_1$. Hence,

$$P_2(x) = f(x_0) + a_1(x - x_0) + a_2(x - x_0)(x - x_1).$$

This process is quite good. To go from P_{n-1} to P_n, we only need to add $a_n(x - x_0) \cdot (x - x_{n-1})$. Then, we only need to determine a_n. This will save work if we only add one more point inside.

Let us determined the coefficients. Using $P_1(x_1) = f(x_1)$, one has

$$a_1 = \frac{f(x_1) - f(x_0)}{x_1 - x_0}$$

Similarly,

$$a_2 = \frac{\frac{f(x_2)-f(x_0)}{x_2-x_0} - \frac{f(x_1)-f(x_0)}{x_1-x_0}}{x_2 - x_1}.$$

This seems to suggest that the coefficients can be defined in a recursively way. In fact, this is true.

We define the zeroth divided difference by

$$f[x_i] = f(x_i),$$

and the first divided difference

$$f[x_i, x_{i+1}] = \frac{f[x_{i+1}] - f[x_i]}{x_{i+1} - x_i}.$$

In general, the k**th divided difference** is defined by

$$f[x_i, x_{i+1}, \ldots, x_{i+k}] = \frac{f[x_{i+1}, x_{i+2}, \cdots, x_{i+k}] - f[x_i, x_{i+1}, \cdots, x_{i+k-1}]}{x_{i+k} - x_i}$$

Remark 1. *Note that the kth divided difference is symmetric:*

$$f[x_0, x_1, \cdots, x_k] = \sum_{j=0}^{k} \frac{f(x_j)}{(x_j - x_0)\cdots(x_j - x_{j-1})(x_j - x_{j+1})\cdots(x_j - x_k)}.$$

Hence, the kth divided difference in fact can be defined by choosing any two points. For example,

$$f[x_i, x_{i+1}, \ldots, x_{i+k}] = \frac{f[x_i, \cdots, x_{i+k-2}, x_{i+k}] - f[x_i, x_{i+1}, \cdots, x_{i+k-1}]}{x_{i+k} - x_{i+k-1}}$$

Using the divided difference, we have

$$f(x) = f(x_0) + f[x, x_0](x - x_0),$$

and

$$f[x, x_0] = f[x_0, x_1] + f[x_0, x_1, x](x - x_1).$$

Similarly, $f[x_0, x_1, x]$ can be written out. In general, one has

$$\begin{aligned} f(x) &= f(x_0) + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1) + \cdots \\ &\quad + f[x_0, x_1, \cdots, x_n](x - x_0)\cdots(x - x_{n-1}) + f[x, x_0, \cdots, x_n]\omega_{n+1}(x). \end{aligned}$$

Since ω_{n+1} is zero at any $x_0, x_1, \cdots, x_n$, we find

$$\begin{aligned} N_n(x) := f(x_0) + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1) \\ + \cdots + f[x_0, x_1, \cdots, x_n](x - x_0)\cdots(x - x_{n-1}), \end{aligned}$$

is exactly the interpolation polynomial at these points. This is called the **Newton's interpolatory divided difference formula**. By the uniqueness of interpolation polynomial,

$$N_n(x) = L_n(x),$$

i.e. it is another form of the Lagrange interpolation polynomial.

Comparing with the starting formula, we have showed that

$$a_n = f[x_0, x_1, \cdots, x_n].$$

Now consider the function

$$\varphi(t) = f(t) - N_n(t).$$

This is zero at $x_0, x_1, \ldots, x_n$, hence by Rolle's theorem, if f is nth continuously differentiable, we have that

$$\varphi^{(n)}(\xi) = 0.$$

In other words,

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

The remainder term

$$R_n := f(x) - N_n(x) = f[x, x_0, \cdots, x_n]\omega_{n+1}(x). \tag{2}$$

Using (1), we find

$$f[x, x_0, \cdots, x_n] = \frac{f^{(n+1)}(\xi(x))}{(n+1)!}$$

provided that f is $(n+1)$th continuously differentiable. However, the error formula (2) holds even if the function is not differentiable.

Netwon's formula is clearly more convenient when the number of points changes. It is more suitable for programming. In fact, similar to Neville's method, one can also fill in a certain table.

When the grid points are uniform, i.e., arranged consecutively with with equal spacing, the Newton's formula can be simplied. Read the book for more details and we omit it here.

2 Hermite Interpolation

What if we want to approximate both the function values and the derivatives at several points? These will lead to the osculating polynomial. When the function values and first order derivatives are given some points, we then have the **Hermite polynomials**.

The idea is again to construct basis functions $\{h_{n,j}(x), \hat{h}_{n,j}(x)\}$ such that

$$h_{n,j}(x_i) = \delta_{ij}, h'_{n,j}(x_i) = 0,$$
$$\hat{h}_{n,j}(x_i) = 0, \hat{h}'_{n,j}(x_i) = \delta_{ij}$$

Then, the interpolation polynomial is given by

$$H_{2n+1}(x) = \sum_{j=0}^{n} f(x_j)h_{n,j}(x) + \sum_{j=0}^{n} f'(x_j)\hat{h}_{n,j}(x).$$

The Hermite interpolation also has a speedup algorithm like Newton's interpolation. Read the book by Burden and Faires.

3 Runge's phenomenon and cubic splines

If we use polynomials to intepolate some smooth functions on equally spaced grid points, the error will grow fast near the boundary, exponentially fast in the degree (i.e. error a^n for $a > 1$). This is called the Runge's phenomenon.

One example is

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

on $[a, b] = [-5, 5]$. We consider the grid points $x_j = -5 + 10\frac{j}{n}$, $j = 0, \ldots, n$. It can be shown that on $(-3.6, 3.6)$ the interplation will converge, while the error grows fast when $|x| > 4$.

The Runge's phenomenon can be understood using the potential theory in the plane. To resolve this issue, there are essentially two ways:

- Use non-uniform grids, like Chebyshev grid points so that there are more points near the boundary.
- Use piecewise interpolation, with degrees of the polynomial be low on each interval.

Here, we mention the second choice briefly, called the **piecewise-polynomial approximation**.

This idea of approximation is the basic idea for **Finite Element Method**, used to construct certain discrete Sobolev spaces.

1. Use piecewise linear function. In this case, the unknowns one each interval is 2, so there are $2n$ unknowns (we have $n+1$ points, and thus n intervals). How about the number of conditions and thus number of equations we can establish? For the linear function on each interval, we need to specify the values at the two endpoints. Hence, there are also $2n$ equations. The coefficients can be solved uniquely.

2. The piecewise linear interpolation only guarantees the continuity of function values. Sometimes, one desires continuity of derivatives. What if we want the first order derivative to be continuous?

   Try using quadratic functions: there are n subintervals, and on each, the qudratic function has three unknowns. Hence, there are $3n$ unknowns.

   How about conditions? For the function values, one can get $2n$ conditions. Moreover, the continuity of first derivatives at the $n-1$ points, yields $n-1$ extra conditions. Hence, there are $3n-1$ required conditions. One needs one more condition. However, you may specify the derivatives at x_0 and x_n, which give two more conditions. This is overdetermined. There is often no solution! Hence, you can only specify one derivative. This is weird.

3. Another practical requirement is that one not only knows the function values at x_j but also the derivatives at x_j. Then, one uses the piecewise Hermit interpolation. On each interval you have a cubic polynomial. For this case, the coefficients can be determined uniquely. The interpolation is continuously differentiable.

4. What if we only know the function values at the grid points but we require the functions to be second order continuously differentiable? This is often desired in designing aircrafts. Then, one needs cubic polynomials on each subinterval. There are $4n$ unknowns. Regarding the conditions, the function values give $2n$ constraints. The continuity of first and second derivatives gives $2(n-1)$ conditions. Hence, in total, there are $4n-2$ constraints.

   We need two more conditions.

3.1 Construction of cubic splines

The interpolation with cubic polynomial on each subinterval that makes the function $C^2[x_0, x_n]$ is called the **cubic spline interpolation**.

We need two more conditions.

- Specifying the first derivatives at the two ends.
- Specifying the second derivatives at the two ends to be zero (the second derivatives often mean the "moments" in beam theory).
- Peridic conditions: need the continuity of function values, first and second derivatives; however, the functions values $f(x_0)$ and $f(x_n)$ should not be specified.

How to find such cubic functions conveniently.

First approach Find the first derivatives b_j at x_j and then use Hermite interpolation. The spline function on $[x_{j-1}, x_j]$ is given by

$$S_j(x) = f(x_{j-1})h^{(j)}_{x_{j-1}}(x) + b_{j-1}\hat{h}^{(j)}_{x_{j-1}}(x) + f(x_j)h^{(j)}_{x_j}(x) + b_j\hat{h}^{(j)}_{x_j}(x),\ j = 1, \cdots, n$$

With this, using the continuity of the second derivatives at x_{j-1} and x_j to establish the equations for b_j. As long as b_j is found. The function $S(\cdot)$ is found.

Second approach A more convenient approach is to solve the moment M_j at x_j first. Then, on $[x_j, x_{j+1}]$, the moment is given by

$$S''(x) = M_j \frac{x_{j+1} - x}{h_j} + M_{j+1} \frac{x - x_j}{h_j}, \ x \in [x_j, x_{j+1}],$$

where $h_j = x_{j+1} - x_j$.

With $S(x_j) = y_j := f(x_j)$ and $S(x_{j+1}) = y_{j+1} := f(x_{j+1})$, one has

$$S'(x_j+) = -\frac{h_j}{3} M_j - \frac{h_j}{6} M_{j+1} + \frac{y_{j+1} - y_j}{h_j},$$

and

$$S'(x_{j+1}-) = \frac{h_j}{6} M_j + \frac{h_j}{3} M_{j+1} + \frac{y_{j+1} - y_j}{h_j}.$$

Hence, at x_j $(j = 1, \cdots, n-1)$, one has

$$\frac{h_{j-1}}{6} M_{j-1} + \frac{h_{j-1}}{3} M_j + \frac{y_j - y_{j-1}}{h_{j-1}} = -\frac{h_j}{3} M_j - \frac{h_j}{6} M_{j+1} + \frac{y_{j+1} - y_j}{h_j}.$$

Hence,

$$\frac{h_{j-1}}{h_{j-1} + h_j} M_{j-1} + 2M_j + \frac{h_j}{h_{j-1} + h_j} M_{j+1} = \frac{6}{h_{j-1} + h_j} (f[x_j, x_{j+1}] - f[x_{j-1}, x_j]) = 6f[x_{j-1}, x_j, x_{j+1}].$$

With the conditions at x_0, x_n, one has a tri-diagonal matrix for M_j. As soon as M_j is found, the function $S(\cdot)$ can be found.

Theorem 1. *Let $h = \max_j h_j$. If $f \in C^4[a,b]$ and $\{x_0, \cdots, x_n\} \subset [a,b]$, $x_0 = a, x_n = b$, then*

$$\|f^{(k)} - S^{(k)}\|_{L^\infty[a,b]} \le C_k \|f^{(4)}\|_{L^\infty[a,b]} h^{4-k}.$$

Though we are not requiring $S'(x_j) = f'(x_j)$, the error of first derivative is small.