

Class business

- Topics due Thursday.

Materials used:

- Gerald and Wheatley; Chapter 3.
- Golub and Ortega, Section 6.3
- Forsythe, et al. Chapter 4

Objectives:

- Close discussion on numerical differentiation and interpolation by reviewing higher order differences, multi-dimensional integrals.
- Functions, or sets of data, can be interpolated or fit using either global or piece-wise polynomials.
- LaGrange polynomials provide easy-to-compute and unique coefficients for interpolating polynomials.
- A common method for interpolation is a cubic spline.

Numerical Differentiation

Last week we derived the forward, backward and centered difference approximations for df/dx and d^2f/dx^2 . These used two or three point values of the function $f(x)$; $(x_{i-1}, f(x_{i-1}))$, $(x_i, f(x_i))$, $(x_{i+1}, f(x_{i+1}))$. The error terms associated with these went as $O(\Delta x)$ and $O(\Delta x^2)$. By using more than three points, higher-order differences achieve a lower error term. They are harder to employ near boundaries, where x_{i-1} or x_{i+1} may not be defined. In the marine sciences, higher order differences are often employed in numerical solutions to partial differential equations; and the boundaries are handled as special cases.

As an example of a higher order difference consider using five points, centered on x_i . The following can be derived by adding the Taylor Series expansions of x_{i-2} , x_{i-1} , x_{i+1} , x_{i+2} , multiplied by -1, 16, 16, -1, respectively.

$$\frac{d^2 f(x_i)}{dx^2} \approx \frac{-f(x_{i+2}) + 16f(x_{i+1}) + 30f(x_i) + 16f(x_{i-1}) - f(x_{i-2}))}{12\Delta x^2} \quad (1)$$
$$+ \frac{2}{15}\Delta x^4 f^{(4)}(x_i);$$

$$Error \sim O(\Delta x^4).$$

Gerald and Wheatley provide several difference formulas in section 5.15.

Another thing to remember in applying these difference equations is that while the lecture assumed Δx to be constant, in practice it is often advantageous to use variable $\Delta x_i = x_{i+1} - x_i$. This enables the algorithm to use higher resolution in areas where the differencing area would be large. For the case of a forward or backward difference approximation for df/dx , this would mean using small Δx where d^2f/dx^2 is largest. The method of finding the difference is the same; but the math is a little messier. Consider finding a centered difference for df/dx under the assumption that Δx_i can vary.

$$\begin{aligned} f(x_2) &= f(x_1 + \Delta x_1) & (2) \\ &= f(x_1) + \Delta x_1 f'(x_1) + f''(x_1) \frac{\Delta x_1^2}{2} + f'''(x_1) \frac{\Delta x_1^3}{3!} + O(\Delta x^4) \\ f(x_0) &= f(x_1 - \Delta x_0) \\ &= f(x_1) - \Delta x_0 f'(x_1) + f''(x_1) \Delta x_0^2 / 2 - f'''(x_1) \frac{\Delta x_0^3}{3!} + O(\Delta x_0^4). \end{aligned}$$

These can be subtracted as before; and $f'(x_1)$ solved for to yield

$$f'(x_1) = \frac{(f(x_2) - f(x_0))}{\Delta x_0 + \Delta x_1} + O((\Delta x)^2); \quad (3)$$

where $\Delta x \approx (\Delta x_0 + \Delta x_1) / 2$.

Numerical Integration

We also covered numerical integration last week, where we considered the global and composite rectangle, trapezoid, and Simpson's rule. Interpolation methods are often called *quadrature schemes*.

Golub and Ortega call what we called the "rectangle rule", where $p_n(x) = f((a + b) / 2)$, the *midpoint rule*, and use *rectangle rule* to refer to $p_n(x) = f(a)$. Gerald and Wheatley used the terminology that we used last week, but it makes more sense to me to differentiate between the rectangle and midpoint rules. They give different error functions, with the midpoint rule being more accurate. The midpoint rule assumes that the function can be evaluated everywhere, at the midpoints as well as at the endpoints of the grid. This may or may not be true for different functions.

The error functions for the trapezoid and rectangle rule can be evaluated using interpolating polynomials as we'll see in today's lecture. The midpoint rule's error function can be derived using a Taylor Series expansion about $(a + b) / 2$, starting with a definition of the error in the integral:

$$error = \int_{x=a}^b (f(x) - p_n(x)) dx \quad (4)$$

$$= \int_{x=a}^b \left(f(x) - f\left(\frac{a+b}{2}\right) \right) dx$$

. The function $f(x)$ is estimated based on Taylor Series expansion around $(a + b)/2$:

$$f(x) = f\left(\frac{a+b}{2}\right) + \left(x - \frac{a+b}{2}\right) f'\left(\frac{a+b}{2}\right) + \frac{1}{2} \left(x - \frac{a+b}{2}\right)^2 f''\left(\frac{a+b}{2}\right) + \dots \quad (5)$$

Equation 6 is then substituted into equation 5; the integral is evaluated, and the error function derived as

$$error \leq \frac{1}{24} M_2 (b - a)^3; \quad (6)$$

where M_2 represents the highest value possible for $f''(x)$ for $a \leq x \leq b$.

These four methods (rectangle, trapezoid, midpoint, and Simpson's rule) all rely on deriving interpolating polynomials for the function in question, and then integrating the interpolating polynomial. They end up with the format

$$I \approx \sum_{i=0}^N c_i f(x_i) \quad (7)$$

for a set of coefficients (c_i) and function evaluations $f(x_i)$. Another way to approach the problem is to accept the basic form (equation 7) of the integral and choose the coefficients to minimize error for certain types of functions. This is known as the *method of undetermined coefficients*. Using $N=2$; and choosing the coefficients such that cubics, quadratics, linear, and constant functions are integrated exactly yields a type of *Gaussian quadrature*. These are popular when function evaluations are computationally expensive, because they can get high degrees of accuracy with minimal function evaluations.

Like finite differences, it is often advantageous to vary the step size for numerical integration; to use small Δx_i in areas where the error term is big, and larger Δx_i where the error term is not big. This can be done by first using a large step size to estimate I . Next, halve the step size and recalculate I . Continue halving the step size only in areas where the successive estimates of the integral have not converged to within some tolerance.

Finally- integration is often needed across more than one dimension. You might need to both time- and depth-integrate a current velocity, for example, to get a seasonal average value of current.

$$I_{xy} = \int_y \int_x f(x, y) dx dy = \int_x \int_y f(x, y) dy dx. \quad (8)$$

This can be done by employing nested integration.

$$\begin{aligned}
 I_{xy} &\approx \sum_y d_j \left(\sum_x c_i f(x_i, y_j) \right) \\
 &\approx \sum_x c_i \left(\sum_y d_j f(x_i, y_j) \right).
 \end{aligned}
 \tag{9}$$

Standard quadrature rules are employed for each set of summations to derive the coefficients (c and d).

Interpolation

Suppose you have a series of pairs of data points or function evaluations, x_i , and $f(x_i) = y_i$, but need to know something about the values, $f(x)$, between the known values, and suppose that you trust your data so you want the function to match the data points exactly. This is not a *regression* problem- where the data are being fit inexactly by a fitting function; but an interpolation where the data are fit exactly. You could perform a *global* interpolation- that is find a single function that fits all n points. A polynomial of degree n could do this, $p_n(x)$. Another option would be to fit a series of functions (perhaps polynomials) between successive points. Linear interpolation between the data would be an example of a *piece-wise* interpolation.

Global Interpolation: Lagrange Polynomials

For n+1 pairs of data points (or function evaluations), say x_i and y_i , $i = 0 \dots n$ there exists exactly one polynomial of degree n that goes through all the points. One expression for this polynomial is the *Lagrange polynomial*:

$$\begin{aligned}
 p_n(x) &= \sum_{i=0}^n l_i(x) y_i \\
 l_j(x) &= \prod_{j \neq i} \frac{(x - x_j)}{(x_j - x_i)}
 \end{aligned}
 \tag{10}$$

You can see that for $p_n(x_i) = y_i$ for $i = 0 \dots n$, and that $p_n(x)$ is a polynomial with leading term x^n . Because there is only one polynomial of degree n that goes through n+1 points, $p_n(x)$ is unique, though there are more efficient methods for determining the terms of the polynomial than applying equation 10.

The error in the interpolation is estimated by the difference between the polynomial and the fitted value and is approximated by

$$\begin{aligned}
 E(x) &= (x - x_0) (x - x_1) \dots (x - x_n) \frac{f^{(n+1)} \epsilon}{(n + 1)!} \\
 &= [\prod_{i=0 \dots n} (x - x_i)] \frac{f^{(n+1)} \epsilon}{(n + 1)!};
 \end{aligned}
 \tag{11}$$

where ϵ is a value between x_0 and x_n .

If you keep increasing the number of data points or function evaluations, n increases. The error term, however, does not necessarily decrease but can grow large. This implies that global interpolating polynomials are not necessarily more accurate as n increases.

Divided Differences

The interpolating polynomial can be found by an alternative method that is more efficient than finding the Lagrange polynomials. It relies on the concept of divided differences; a way of estimating increasingly higher orders of differentials. The notation introduced by divided differences comes up often; so I'll provide it here.

$$f[x_i] = y_i;$$

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

$$f[x_i, x_{i+1}, x_{i+2}] = \frac{f[x_{i+1}, x_{i+2}] - f[x_i, x_{i+1}]}{x_{i+2} - x_i}. \quad (12)$$

$$(13)$$

This notation can be used to define the global interpolating polynomial called the Newtonian Divided Difference Polynomial, $p_n(x)$ for a series of data (x_0, y_0) , $(x_1, y_1), \dots, (x_n, y_n)$:

$$p_n(x) = f[x_0] + (x - x_0) f[x_0, x_1] + (x - x_0)(x - x_1) f[x_0, x_1, x_2] + \dots + (x - x_0)(x - x_1) \dots (x - x_{n-1}) f[x_0, x_1, \dots, x_n]. \quad (14)$$

The advantage of using Newton's divided difference over the Lagrange form (they both give the identical polynomial, disregarding rounding error), is that new data points can be added to Newton's form without having to recalculate all of the terms.

Whether Newton's form or the Lagrange form are used, however, finding an interpolating polynomial quickly becomes cumbersome as n increases, and becomes numerically unreliable by the time the data set contains ten (x_i, y_i) pairs. Another problem is that the higher order polynomials sometimes have large oscillations so that $p_n(x)$ may produce values well outside the range of the data. For these reasons, global interpolating polynomials aren't used very often.

These problems are avoided by choosing several, lower-ordered polynomials to fit data points within a sub-set of all data points. The most common of these are linear interpolations and cubic splines. By keeping the order of the polynomial low, these methods maintain numerical stability, and efficient methods have been developed for finding the interpolation.

Piecewise Interpolation: Linear

For a linear interpolation, you simply assume that the interpolating function is piece-wise linear between ordered pairs of the data set. If you need to estimate $f(x)$, where $x_i \leq x \leq x_{i+1}$, you would use

$$f(x) = \frac{y_{i+1} - y_i}{x_{i+1} - x_i} (x - x_i) + y_i. \quad (15)$$

If you have $n+1$ data points (from $i = 0 \cdots n$), you can interpolate lines between the n sets of adjacent data points. The interpolation will be continuous, because $f(x)$ will approach $f(x_i)$ as x approaches x_i from either the left or right. The slope of the interpolation will not be continuous, however, and that is why people use higher order interpolations like the cubic spline.

Piecewise Interpolation: Cubic Spline

If instead of using a linear fit between pairs of data, we use a cubic polynomial between data points, we can derive an interpolating function that is continuous to its second derivative. This is called a *cubic spline*.

Suppose you again have $n+1$ data points, from $i = 0 \cdots n + 1$, and want to fit a cubic spline to them. There are n intervals between consecutive pairs of data points. To fit cubic polynomials to each of these n intervals, you'd need to specify $4 \times n$ coefficients. To do this requires that you have the same number (i.e. $4n$) of constraints. One set of constraints that you have is that each set of interior data points is satisfied by two of the cubic polynomials, that gives $2 \times (n - 1)$ constraints. The endpoints only have to be satisfied by one cubic each; that gives a total of $2n$ constraints for satisfying the requirement that $f(x_i) = y_i$ for all i . Another set of constraints comes from requiring that the function be continuous in slope. That means that $f'_-(x_i) = f'_+(x_i)$ for the $n-1$ interior points, which supplies another $n-1$ constraints. Cubic splines also have the property that they are continuous in their second slopes, and this supplies another $n-1$ constraints. Therefore, the constraints that the spline be continuous, and be continuous in the first and second derivative supply a total of $2(n - 1) + 2 + (n - 1) + (n - 1) = 4n - 2$ constraints, two less than the needed $4n$. The extra two constraints are supplied by making some assumption about the behavior of the interpolation near the endpoints (x_0 and x_n).