Statistics and Numerical Methods, Tsinghua University

Numerical differentiation, interpolation and integration

Xuening Bai (白雪宁)

Institute for Advanced Study (IASTU) & Department of Astronomy (DoA)

清華大學

Tsinghua University

Sep. 24, 2024

Truncation error

In any numerical algorithm, we *approximate* the desired solution with a finite number arithmetic operations. Examples include:

- Approximate a function / derivative / integrals with discrete data
- Series expansion truncated after finite number of terms
- Iterative methods that are stopped after desired error is reached

The difference between true and approximate (numerical) solution is called the truncation error.

$$f_E = f_A + \varepsilon_T$$

Exact = Approximation + Truncation error

Convergence

For a valid numerical solution, the truncation error must decrease towards zero as more information is used to construct the solution (i.e., the algorithm converges).

This is usually given by some number "N", e.g., # of discrete points, # of iterations, # of terms in a series, etc.

As the algorithm converges, the truncation error usually diminishes at a rate of this number to some power:

$$\varepsilon_T \sim N^{-\alpha}$$

The power α is related to the order of the algorithm.

Make sure your implementation is correct

i.e., your code is bug-free

Two elements of verification:

Convergence:

Run your code with increasing resolution, and check its order of convergence (before reaching round off error).

Make sure it matches the expected rate!

Consistency:

Make sure it converges to the correct solution!

Validate the results against some analytical solutions.

Summary of errors

Roundoff error:

You have to live with round-off error, but you can modify your algorithms to be less sensitive to it.

Again, use double-precision to minimize its potential influences, so that you can generally focus on truncation errors.

Truncation error:

You can control/reduce truncation error by either increasing the resolution, or by using higher-resolution method to make convergence faster.

But be cautious that your solution may not be smooth!

Outline

- Numerical differentiation
- Numerical interpolation
- Numerical integration

Numerical differentiation: overview

Given a set of points $x_i, f(x_i)$, $(i=0,\ldots,N)$, approximate the derivative $f'(x_i)$.

Two methods:

- Analytically differentiating the interpolating function.
- Approximate the derivatives with by finite differencing.

Usually, they are equivalent on a regular grid. We proceed with the latter.

Sometimes, accuracy is limited not by truncation error, but round-off error.

Many numerical methods rely on certain ways to numerically approximate derivatives, as we shall see later in the course (e.g., solving nonlinear equations, ODEs and PDEs).

Forward/backward differencing

Forward differencing: $f'(x) \approx \dfrac{f(x+h) - f(x)}{h}$

How accurate is this estimate?

$$f(x+h) = f(x) + hf'(x) + \frac{1}{2}h^2 f''(x) + \frac{1}{6}h^3 f'''(x) + \cdots$$

$$\frac{f(x+h) - f(x)}{h} = f' + \boxed{\frac{1}{2}hf'' + \cdots} \leftarrow \text{truncation error}$$

The method is 1st order accurate because truncation error is $\propto h$.

Similarly, there is backward differencing: $f'(x) \approx \dfrac{f(x) - f(x-h)}{h}$

Central differencing

We can improve the accuracy by central differencing: $f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}$

With Taylor expansion, one can easily show that

$$f(x \pm h) = f(x) \pm hf'(x) + \frac{1}{2}h^2 f''(x) \pm \frac{1}{6}h^3 f'''(x) + \cdots$$

$$\frac{f(x+h) - f(x-h)}{2h} = f' + \boxed{\frac{1}{6}h^2 f''' + \ldots}$$ ← truncation error

The method is 2nd order accurate because truncation error is $\propto h^2$. This is the highest order of convergence we can expect from a single divided difference.

It can also be considered as combining two low-order approximations to a higher-order one, a process called Richardson extrapolation (we'll revisit later).

Discussion: round off vs truncation errors

Suppose you want to compute this derivative around x=10.1, and you take h=0.0001, using single precision.

First consider forward/backward differencing.

Round off error of the order $e_r \sim \epsilon_f |f(x)/h|$

Note: $\epsilon_f \gtrsim 10^{-7}$ depending on the form of f.

Truncation error of the order $e_t \sim |hf''(x)|$

Optimal scenario: the two errors are comparable => $h \sim \sqrt{\frac{\epsilon_f f}{f''}} \approx \sqrt{\epsilon_f} x_c$

characteristic scale over which f changes

Fractional accuracy of the *optimal estimate*:

$$(e_r + e_t)/|f'| \sim \sqrt{\epsilon_f}(f f''/f'^2)^{1/2} \sim \sqrt{\epsilon_f}$$

Simple finite differencing *at best* gives accuracy of square root of machine precision.

Discussion: round off vs truncation errors

Suppose you want to compute this derivative around x=10.1, and you take h=0.0001, using single precision.

Next consider central differencing.

Round off error of the order $e_r \sim \epsilon_f |f(x)/h|$

Truncation error of the order $e_t \sim |h^2 f'''(x)|$

Note: $\epsilon_f \gtrsim 10^{-7}$ depending on the form of f.

Optimal scenario: the two errors are comparable => $h \sim \left(\frac{\epsilon_f f}{f'''}\right)^{1/3} \sim (\epsilon_f)^{1/3} x_c$

characteristic scale over which f changes

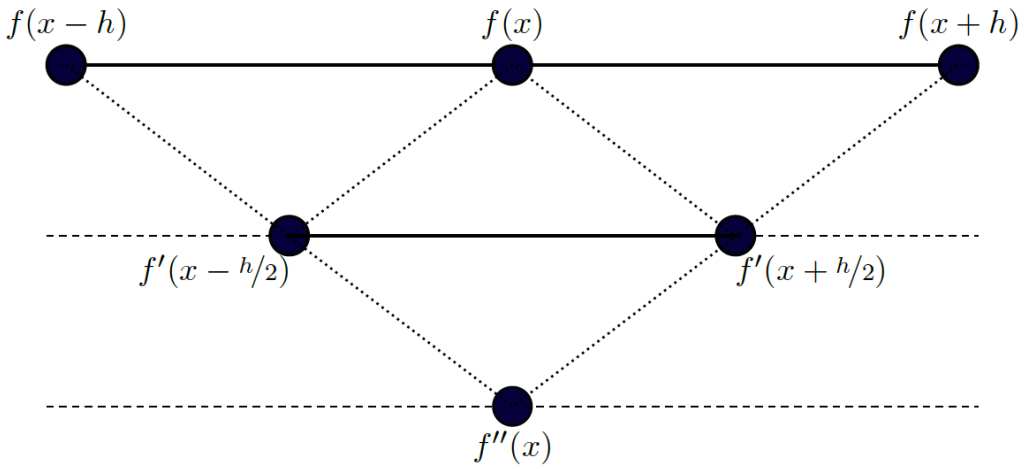
Fractional accuracy of the *optimal estimate*:

$$(e_r + e_t)/|f'| \sim (\epsilon_f)^{2/3} f^{2/3} (f''')^{1/3}/f' \sim (\epsilon_f)^{2/3}$$

Higher-order differencing can improve the accuracy towards machine precision.

Higher-order derivatives

From the same Taylor series: $f(x \pm h) = f(x) \pm hf'(x) + \frac{1}{2}h^2 f''(x) \pm \frac{1}{6}h^3 f'''(x) + \cdots$



Following central differencing, we obtain: $f''(x) = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} - \frac{1}{12}h^2 f^{(4)}(x) + \cdots$

(again, lucky cancellations leads to 2nd order accuracy)

Higher-order finite difference

There are a wide variety of finite differencing schemes, by using wider stencil to cancel more terms in Taylor series.

Caution: they only work for sufficiently smooth functions.

General rule: using a stencil of n+1 points generally gives an method that is n^{th} order accurate.

Example:

$$f'(x) = \frac{-2f(x-h) - 3f(x) + 6f(x+h) - f(x+2h)}{6h} + \frac{1}{12}h^3 f^{(4)}(x) + \cdots$$

4-point stencil, 3rd order accurate

Higher-order finite difference: compact schemes

All formulae so far have utilized only neighboring function values.

Higher-order can also be achieved by using neighboring values of the function derivatives.

For instance, one can estimate 4th order accurate derivatives as

$$\frac{1}{4}f'_{i-1} + f'_i + \frac{1}{4}f'_{i+1} = \frac{3}{4}\frac{f_{i+1} - f_{i-1}}{h} + \frac{1}{120}h^4 f_i^{(5)}$$

$$\frac{1}{10}f''_{i-1} + f''_i + \frac{1}{10}f''_{i+1} = \frac{6}{5}\frac{f_{i-1} - 2f_i + f_{i+1}}{h^2} + \frac{1}{200}h^4 f_i^{(6)}$$

Known as "compact schemes"

Advantage: more accuracy for narrower stencil.

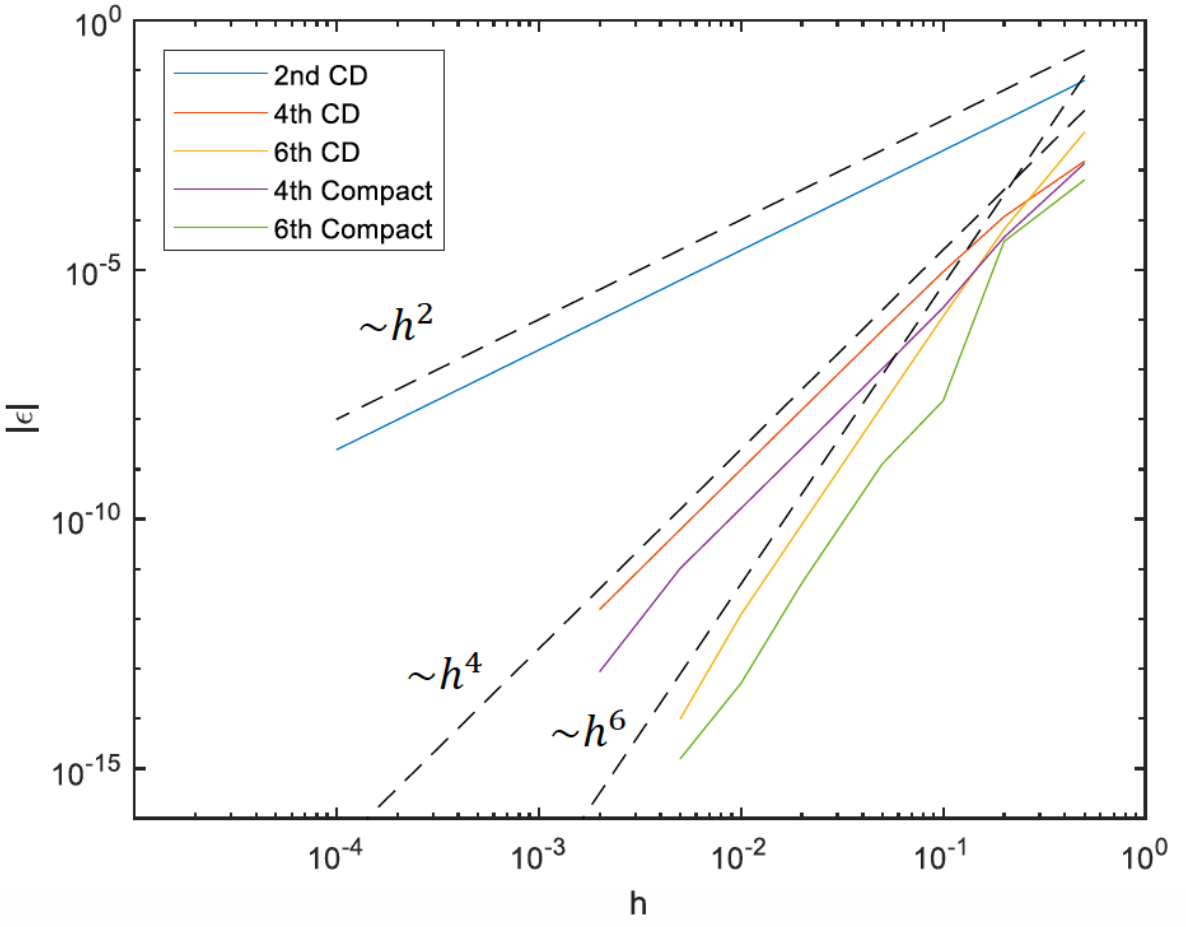
Disadvantage: need to solve a linear system [but not very expensive, in O(N)].

Example:

Consider the Gaussian function:

$$f(x) = e^{-x^2}, \quad f'(x) = -2xe^{-2x^2}$$

Evaluate the derivative at x=1.



Summary on numerical differentiation

- For most applications, central differencing provide cheap and sufficiently accurate approximations.
- Roundoff errors can severely degrade the accuracy. Use double-precision!
- Higher-order schemes generally require wider stencil (may be dangerous). Can use compact scheme involving neighboring derivatives.

We have only considered differentiation in a regular mesh.

More advanced methods required for constructing derivatives from unstructured point data (e.g., in SPH simulations).

Outline

- Numerical differentiation
- Numerical interpolation
- Numerical integration

Interpolation and extrapolation

We know the value of a function $f(x)$ at a set of tabulated points, x_0, x_1, …, x_N, but don't know the values at other points.

We want to estimate the value of f for arbitrary x. It is called interpolation if x is within the smallest and largest values of $\{x_i\}$. Otherwise, it is called extrapolation.

When using a set of $N+1$ points, the method is said to be of N th order.

For our purpose, we do not distinguish interpolation and extrapolation, but we strongly caution using an interpolation function outside the range of tabulated values by more than the typical spacing of tabulated points.

Polynomial interpolation

Most straightforward class of methods.

Task: find a polynomial $f(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_N x^N$

that passes all data points (x_0,y_0), (x_1,y_1), …, (x_N,y_N) , i.e., to solve for $\{a_i\}$.

Naive approach:

$$\begin{pmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^N \\ 1 & x_1 & x_1^2 & \cdots & x_1^N \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_N & x_N^2 & \cdots & x_N^N \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_N \end{pmatrix} = \begin{pmatrix} f(x_0) \\ f(x_1) \\ \vdots \\ f(x_N) \end{pmatrix}$$

It takes $O(N^3)$ to solve (see next lecture), and it is ill-conditioned.

Lagrange interpolation

We can define Lagrange basis polynomial:

$$l_k(x) \equiv \frac{\prod_{j=0, j\neq k}^{N}(x - x_j)}{\prod_{j=0, j\neq k}^{N}(x_k - x_j)} \quad \text{so that} \quad l_k(x_i) = \delta_{ik}$$

With this, we have the Lagrange formula: $L_N(x) = \sum_{k=0}^{N} y_k l_k(x)$

This formula is mathematically unique, and takes $O(N^2)$ to evaluate.

Error estimate: $R_N(x) = \frac{f^{(N+1)}(\xi)}{(N+1)!}\omega_{N+1}(x)\ , \quad \xi \in (x_0, x_N), \quad \omega_{N+1}(x) \equiv \prod_{j=0}^{N}(x - x_j)\ .$

Example: linear and parabolic interpolation

First order (linear) interpolation:

$$l_0(x) = \frac{x - x_1}{x_0 - x_1}, \quad l_1(x) = \frac{x - x_0}{x_1 - x_0}$$

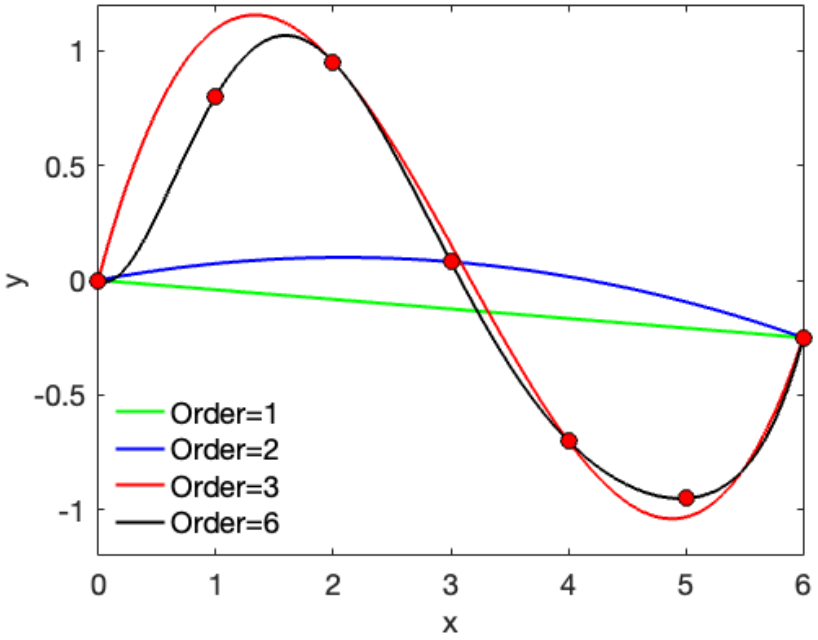
$$\begin{aligned} L_1(x) &= f_0 l_0(x) + f_1 l_1(x) \\ &= f_0 + \frac{f_1 - f_0}{x_1 - x_0}(x - x_0) \end{aligned}$$

Second order (parabolic) interpolation:

$$l_1(x) = \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)}$$

$$l_2(x) = \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)}$$

$$l_3(x) = \frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)}$$



Implementation

The Lagrange formula is somewhat awkward at implementation level:

- Not very flexible: with any change, need to recompute all coefficients.
- The denominator could potentially amplify round-off errors.
- Does not give an estimate of truncation error.

More convenient is to use Neville's algorithm:

$$
\begin{array}{lllll}
x_0: & y_0 = P_0 & & & \\
 & & P_{01} & & \\
x_1: & y_1 = P_1 & & P_{012} & \\
 & & P_{12} & & P_{0123} \\
x_2: & y_2 = P_2 & & P_{123} & \\
 & & P_{23} & & \\
x_3: & y_3 = P_3 & & &
\end{array}
$$

Here, $P_{i(i+1)\ldots(i+m)}$ is the polynomial that connects (x_i,y_i), $(x_{i+1},y_{i+1}),\ldots,$ (x_{i+m},y_{i+m}).

With a recursive relation:

$$P_{i(i+1)\ldots(i+m)} = \frac{(x - x_{i+m})P_{i(i+1)\ldots(i+m-1)} + (x_i - x)P_{(i+1)(i+2)\ldots(i+m)}}{x_i - x_{i+m}}$$

Implementation

More convenient is to use Neville's algorithm:

$$
\begin{array}{lll}
x_0: & y_0 = P_0 & \\
 & & P_{01} \\
x_1: & y_1 = P_1 & P_{012} \\
 & & P_{12} \qquad P_{0123} \\
x_2: & y_2 = P_2 & P_{123} \\
 & & P_{23} \\
x_3: & y_3 = P_3 &
\end{array}
$$

Here, $P_{i(i+1)\ldots(i+m)}$ is the polynomial that connects (x_i,y_i), $(x_{i+1},y_{i+1}),\ldots,$ (x_{i+m},y_{i+m}).

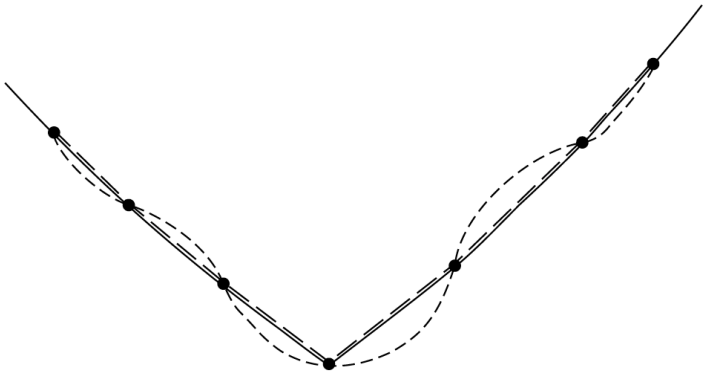
With a recursive relation:

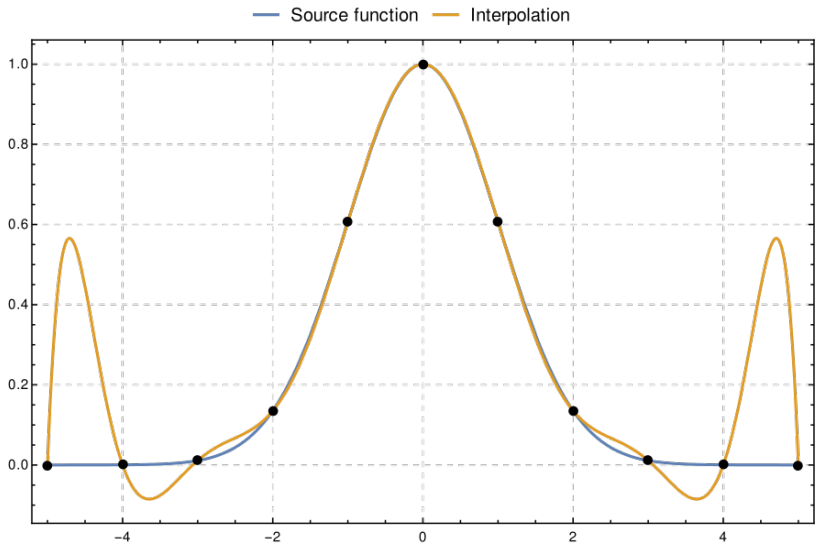
$$P_{i(i+1)\ldots(i+m)} = \frac{(x - x_{i+m})P_{i(i+1)\ldots(i+m-1)} + (x_i - x)P_{(i+1)(i+2)\ldots(i+m)}}{x_i - x_{i+m}}$$

Advantages:

- Adding one new data point x_m: only takes O(M) to evaluate.
- Less prone to round-off errors.
- Can use $P_{0..m}$-$P_{0\ldots m-1}$ or as $P_{0..m}$-$P_{1\ldots m}$ as rough error estimate.

Is higher-order always good?



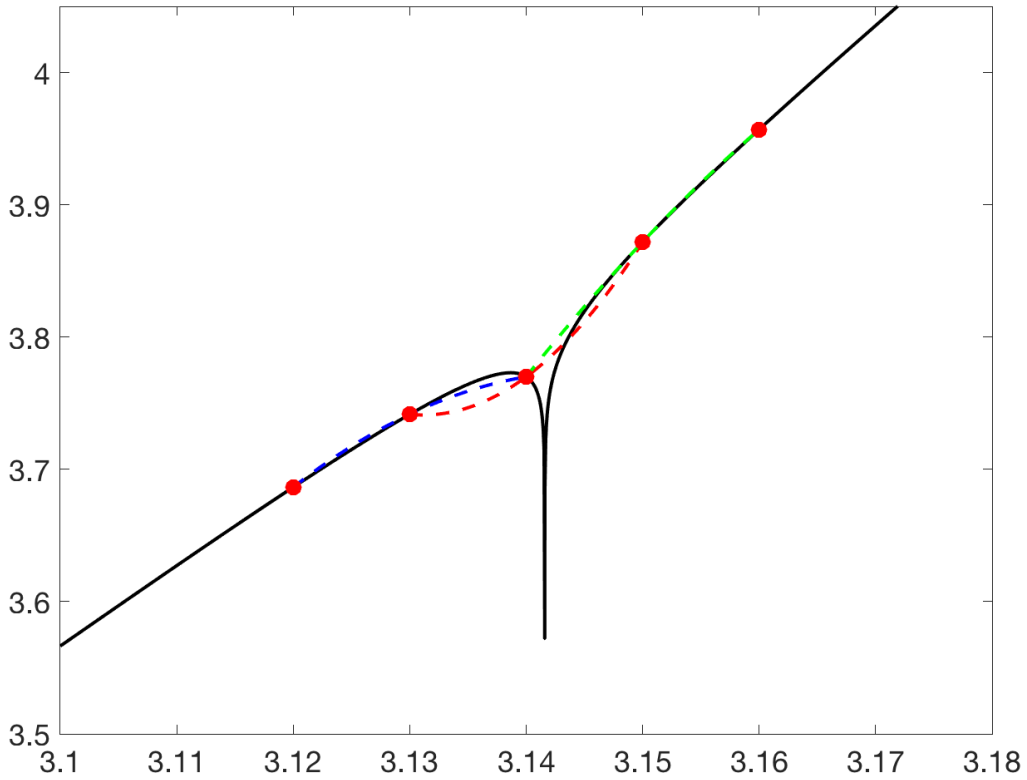


For non-smooth functions, higher-order interpolation yields oscillations.

Runge phenomenon: $f(x) = \dfrac{1}{1 + ax^2}$

Even the function is perfectly smooth, higher-order polynomial interpolations can lead to oscillations, mainly at the edges (due to poles in the complex plane).

Pathological cases



Truncation error usually depends on higher-order derivatives, and can be very large when the function is not well-behaved.

Rational function interpolation

Not all functions are well approximated by polynomials, especially functions with poles. Can use rational functions:

$$R(x) = \frac{P_\mu(x)}{Q_\nu(x)} = \frac{p_0 + p_1 x + \cdots + p_\mu x^\mu}{q_0 + q_1 x + \cdots + q_\nu x^\nu}$$

Need to specify the order for both denominator and numerator (μ+ν+1 unknowns in total). See *Numerical Recipes* for algorithms.

Rational functions can also do well without poles, in this barycentric form:

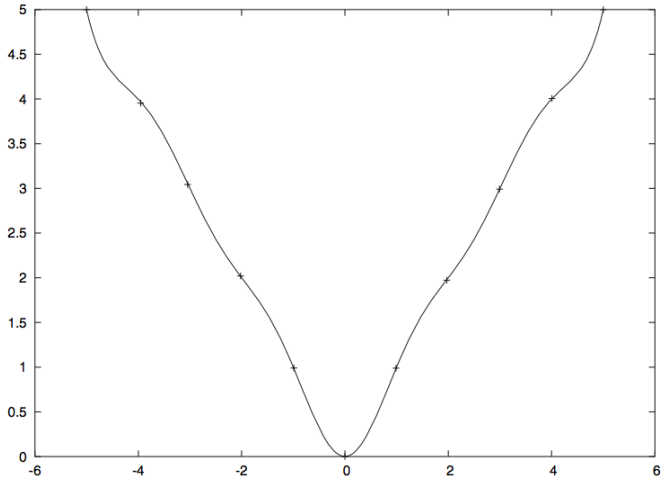
$$R(x) = \frac{\sum_{i=0}^{N} \frac{w_i}{x - x_i} y_i}{\sum_{i=0}^{N} \frac{w_i}{x - x_i}}$$

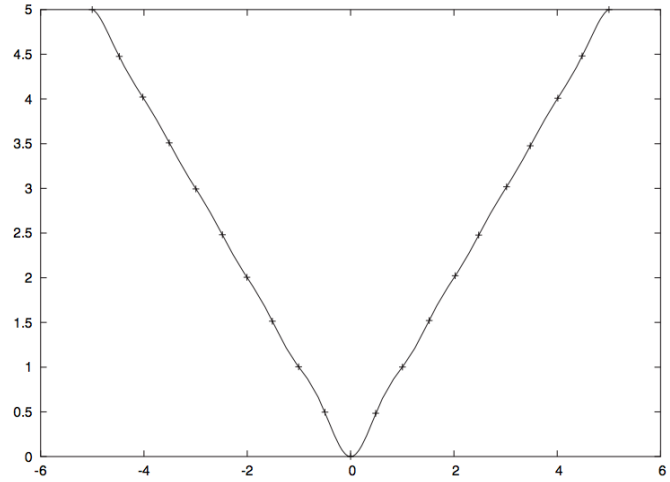
See *Numerical Recipes* for formulae of weights.

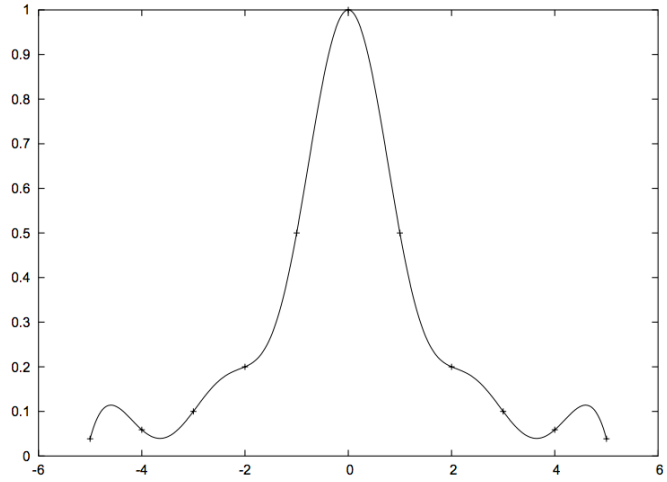
Can be written to achieve desired order (<N) smoothly with small error.

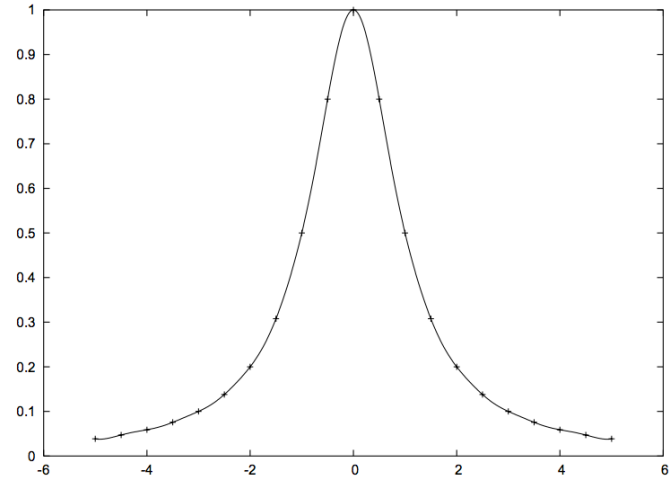
Rational function interpolation

n=10









n=20

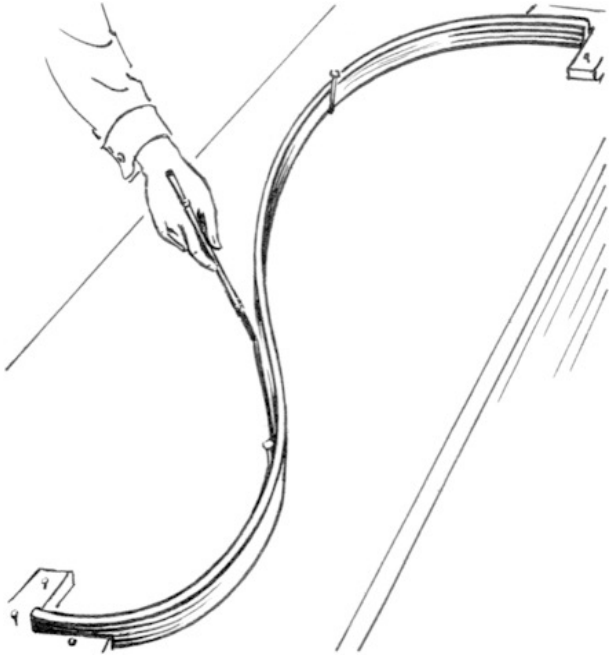
Spline interpolation

The issue with higher-order interpolation scheme motivates the use of lower-order, piecewise polynomials.

The piecewise polynomials should smoothly connect all data points.

Spline: a long strip with fixed in position at a number of points whose tension creates a smooth curve joining those points.

Spline interpolation is the mathematical formulae devised for a similar purpose.



Cubic spline interpolation

The most widely used form of spline interpolation.

We are given the value of $f(x)$ at a set of tabulated points, x_0, x_1, …, x_N,.

At each interval $[x_{i-1}, x_i]$ (i=1,..,N), construct a cubic polynomial $S_i(x)$ such that

$S_i(x) = f(x)$, for $x=x_{i-1}, x_i$;

$S'_i(x_i) = S'_{i+1}(x_i)$ continuity of 1st order derivative

$S''_i(x_i) = S''_{i+1}(x_i)$ continuity of 2nd order derivative

4N unknowns. 2N relations to match function values, 2(N-1) relations for continuity of first and second order derivatives. (all linear)

We need 2 more relations. They are given by boundary conditions (e.g., derivatives of $f(x)$ at the boundaries).

Cubic spline interpolation: implementation

Here we just give an outline, and it is usually implemented as standard packages.

Let M_i (i=0,…,N) be the 2nd derivatives of $f(x)$ at x_i. Once M_i are known, then each S_i (i=1,…,N) can be easily calculated.

The remaining task is to determine M_i (N+1 unknowns), determined by N-1 relations by requiring continuity of 1st derivative, and two boundary conditions.

This leads to a set of N+1 linear equations with a tridiagonal matrix:

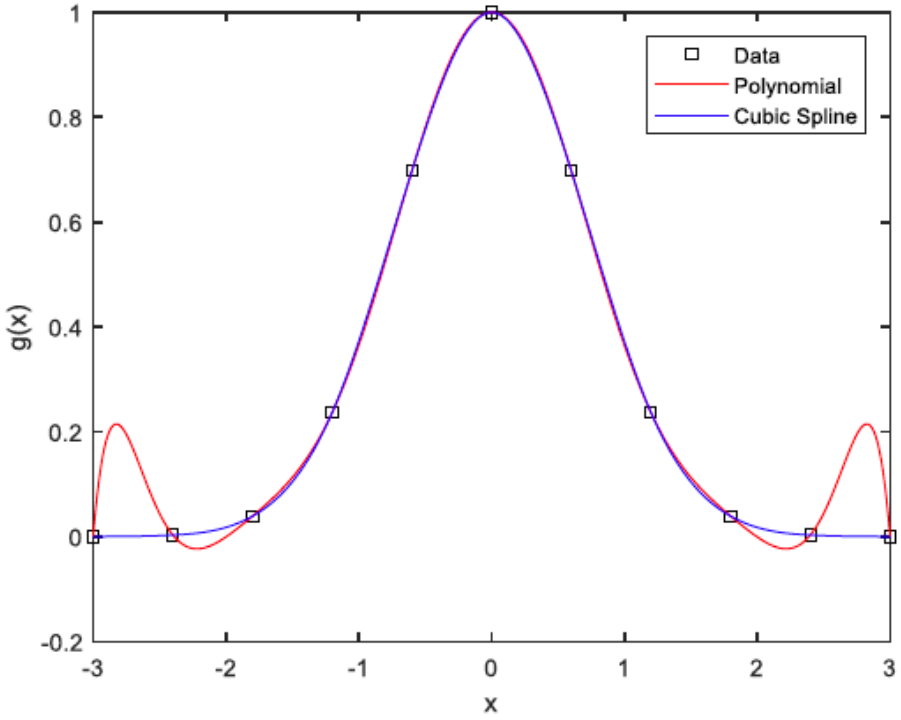
$$\begin{pmatrix} 2 & \lambda_0 & & & & \\ \mu_1 & 2 & \lambda_1 & & & \\ & \ddots & \ddots & \ddots & & \\ & & \mu_{N-1} & 2 & \lambda_{N-1} & \\ & & & & 2 & \lambda_N \end{pmatrix} \begin{pmatrix} M_0 \\ M_1 \\ \vdots \\ M_{N-1} \\ M_N \end{pmatrix} = \begin{pmatrix} d_0 \\ d_1 \\ \vdots \\ d_{N-1} \\ d_N \end{pmatrix}$$

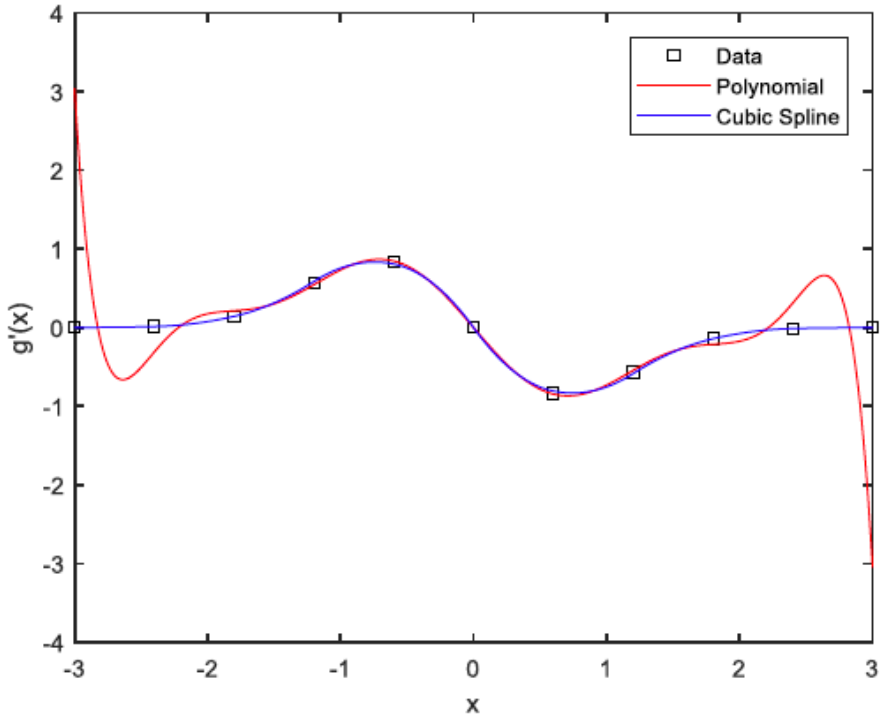
where λ_i and μ_i are related to x_i and their spacings, d_i are related to $f(x)$ around x_i and boundary conditions.

This matrix can be solved very efficiently in O(N) time.

Cubic spline interpolation: example

Function: $\exp(-x^2)$ and its derivative





Cubic spline interpolation: properties

Cubic spline has the desirable property that it minimizes the functional:

$$J(f) \equiv \int_{x_{\min}}^{x_{\max}} |f''(x)|^2 dx$$

This is what a spline likely satisfies (minimizing elastic energy from bending).

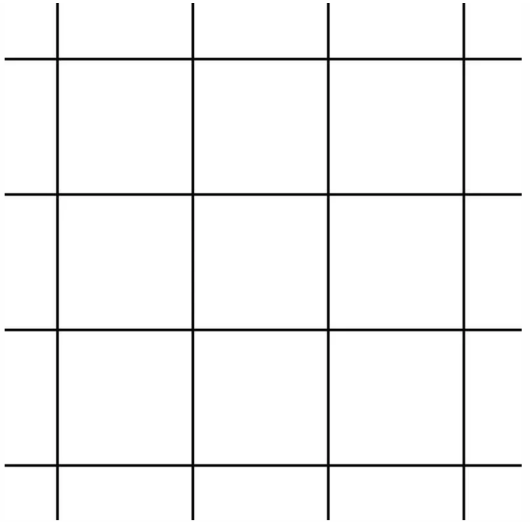
If f is C^4 smooth, then spline interpolation converges to f to 4^{th} order with respect to grid spacing h=max($|x_i - x_{i-1}|$).

In sum, spline interpolation is generally sufficiently accurate while avoids oscillations / Runge phenomenon encountered in high-order interpolations.

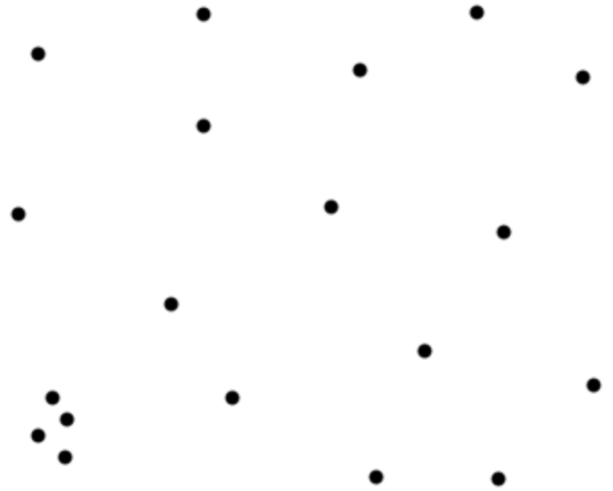
Interpolation in multiple dimensions

Focusing on 2D for now. Task is to seek an estimate of $y=f(x_1,x_2)$.

First question is, whether the data are given on a regular grid (i.e., tabulated), or as a set of scattered points?



Easier to deal with, but unrealistic at higher-dimensions (too much data).

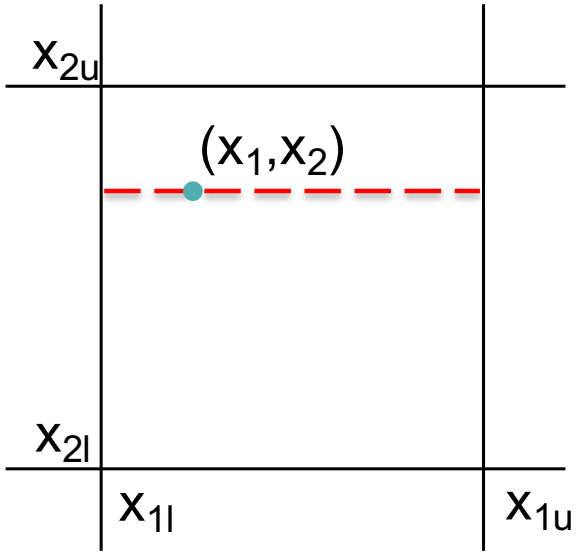


More expensive and less accurate, but common at higher dimension.

Interpolation on regular grid

General idea is to interpolate first in one dimension, then in the other.

$$y_{ij} = f(x_{1i}, x_{2j}) \quad (i=0,\ldots,M;\ j=0,\ldots,N)$$



Bilinear interpolation:

"Local" method that is piecewise linear in each dimension.

Bicubic spline interpolation:

Cubic spline interpolation at fixed x_2 in all x_1 columns, then cubic spline interpolation along x_1 at fixed x_2.

There is also the "bicubic interpolation", requiring knowing/approximating 2nd-order derivatives at grid points, which is more involved.

Interpolation with scattered points

Radial basis function (RBF) interpolation

Given a set of N points, imagine that they all influences the surroundings in the same way in all directions according to some assumed functional form $\phi(r)$, i.e., the radial basis function. We can approximate the interpolating function as

$$y(\boldsymbol{x}) = \sum_{i=1}^{N} w_i \phi(|\boldsymbol{x} - \boldsymbol{x}_i|)$$

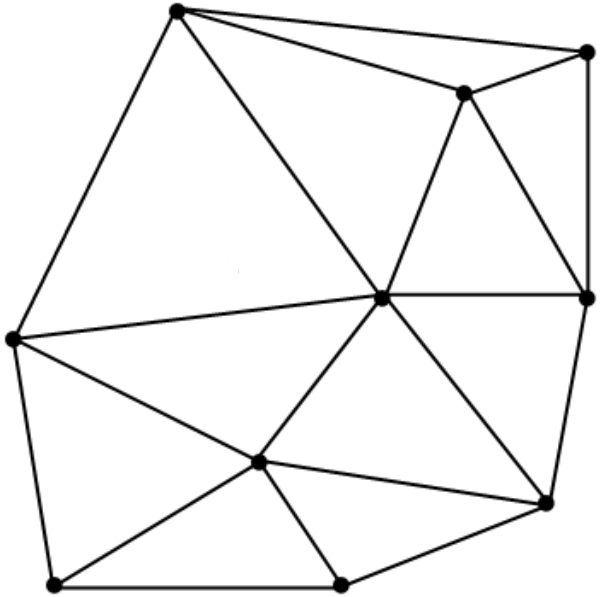
Requiring that y takes the value of $f(\boldsymbol{x})$ at the N data points leads to a linear system of equations to solve for the weights w_i.

One can experiment with different forms of $\phi(r)$, such as $\phi(r) = (r^2 + r_0^2)^{\pm 1/2}$ or Gaussian. Should choose r_0 as some intermediate size.

Convenient, but slow (solving linear system requires $O(N^3)$), good for small systems.

Interpolation with scattered points

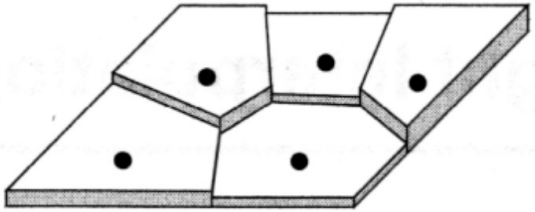
Delaunay triangulation



Dual of Voronoi diagram, can be constructed in ~O(N logN).

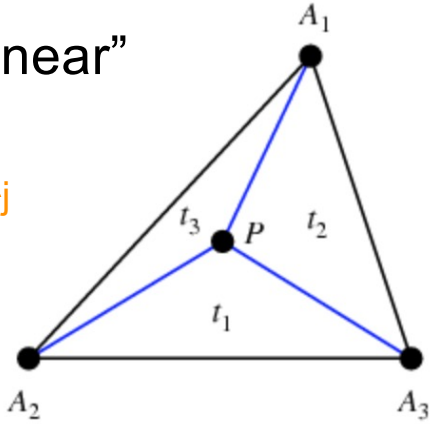
More effective when you have LOTS of data points (>1000)

nearest neighbor



"piecewise linear"

weight of A_j $\propto$ area of t_j



Interpolation using barycentric coordinates

Summary: interpolation

- If you only need a coarse approximation to a function, piecewise linear interpolation is not so bad.
- If a more accurate approximation is desired, cubic spline is a good balance between accuracy, robustness and computational cost.

If your data are noisy, interpolation essentially “overfits” data.

Smooth your data first (see below)! Or use regression (to be covered later in the course) instead!

Further reading: data smoothing

Interpolation implicitly assumes that data are accurate.

Often not the case: data can be highly noisy (in time series and in space).

Two approaches:

- Fitting (a.k.a., regression)

  Often involves using of an explicit function form for the result.

  Will be covered in 2nd half of the course.

- Data smoothing

  Reduce fine-structures and rapid variations in the data (i.e., low-pass filter), no use of any functional form.

  Extensive literature, we just mention two useful methods in 1D.

Simple smoothing

Moving average (MA):

Many flavors: simple MA, exponential MA, cumulative MA, etc…

Another useful smoothing filter: $y_n = \frac{x_{n-1} + 2x_n + x_{n+1}}{4}$.

Can be applied multiple times.

Savitzky-Golay smoothing filters

Consider equally spaced data in 1D, given by f_i. Define a linear filter as:

$$g_i = \sum_{n=-n_L}^{n_R} c_n f_{i+n}$$

Perform a least square fit to a higher-order (M) polynomial at each data point i.

Set g_i to the value of that polynomial at position i.

The expression of g_i gives the coefficients c_n.

Then move to the next data point i+1 (i.e., sliding window).

This way, the result preserves higher-order moments of any features (e.g., pulses, spectral line) without, e.g., reducing peaks at maxima.

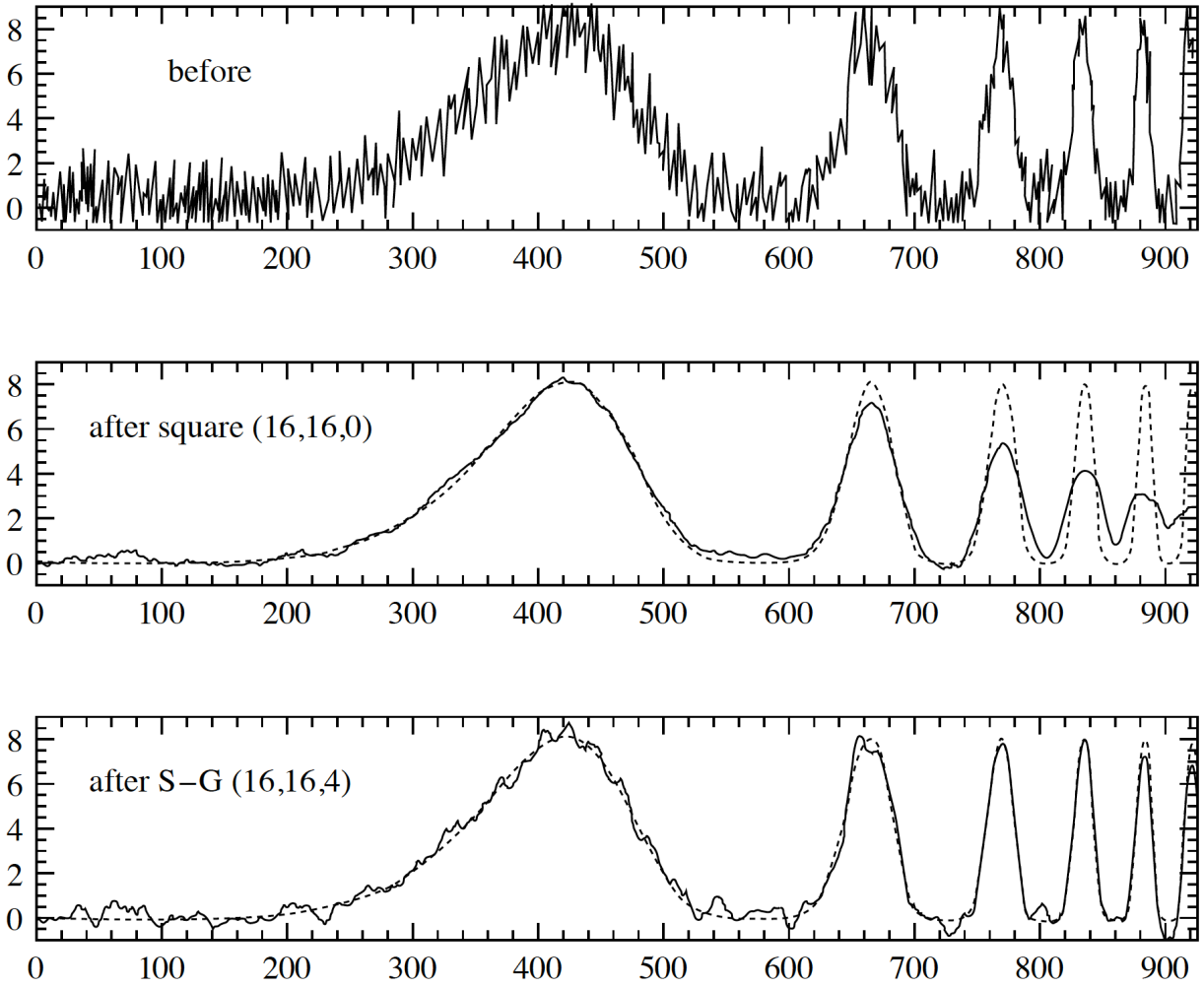
Savitzky-Golay smoothing filters

For evenly sampled data, the coefficients can be calculated once-and-for-all.

M	n_L	n_R	Sample Savitzky-Golay Coefficients										
2	2	2				−0.086	0.343	0.486	0.343	−0.086			
2	3	1			−0.143	0.171	0.343	0.371	0.257				
2	4	0		0.086	−0.143	−0.086	0.257	0.886					
2	5	5	−0.084	0.021	0.103	0.161	0.196	0.207	0.196	0.161	0.103	0.021	−0.084
4	4	4		0.035	−0.128	0.070	0.315	0.417	0.315	0.070	−0.128	0.035	
4	5	5	0.042	−0.105	−0.023	0.140	0.280	0.333	0.280	0.140	−0.023	−0.105	0.042

If the data are not evenly sampled, then need to do the fitting at every location (can be very computationally intensive).

Savitzky-Golay smoothing filters



Outline

- Numerical differentiation
- Numerical interpolation
- Numerical integration

Numerical integration

Also known as quadrature. Goal is to compute: $I = \int_a^b f(x)dx$

To do so, divide the interval [a,b] into a set of points, and approximate the integral by a weighted sum of f over these points.

Note that this is equivalent to solving for y below, i.e., an ODE with boundary value $y(a)=0$:

$$\frac{dy}{dx} = f(x)$$

Emphasis for solving ODEs: the context is broader, emphasizing adaptive stepsize

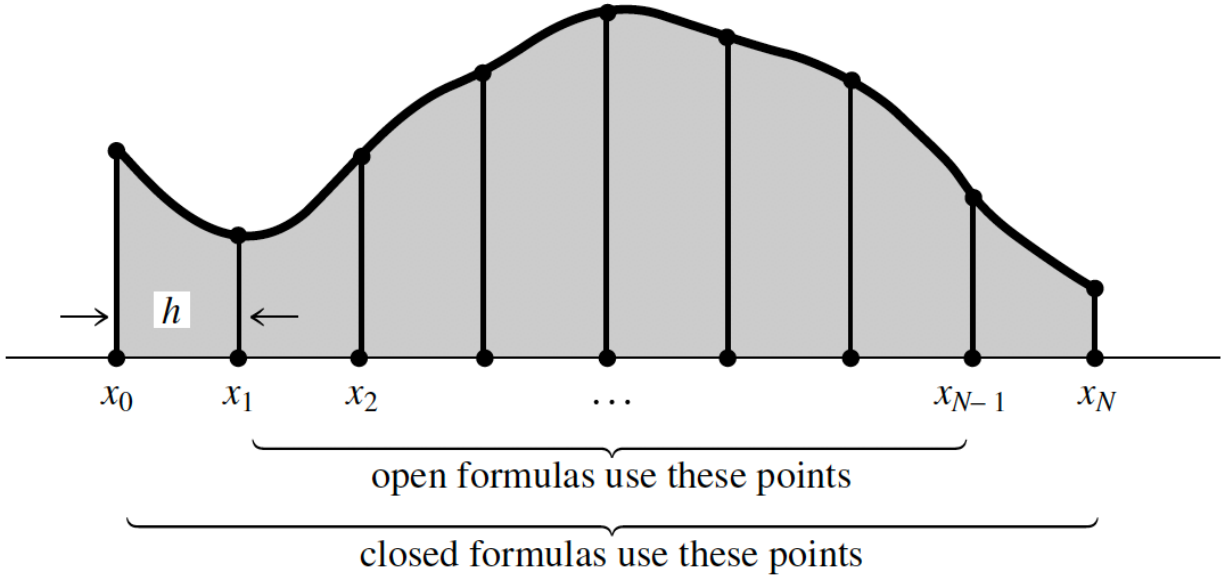
Emphasis for quadrature: adding up values at a sequence of abscissas, within certain range of integration.

Newton-Cotes integration

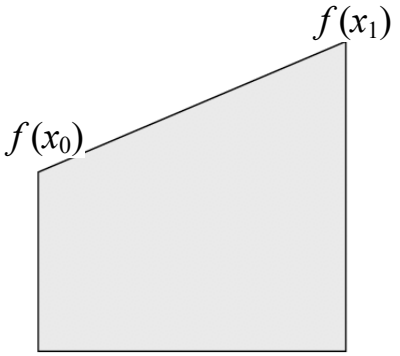
First consider equally-spaced abscissas.

Basic idea: integrate over the interpolating Lagrange polynomials.

$$f(x) = \sum_{k=0}^{N} f(x_k) l_k(x) + O(h^{N+1} f^{(N+1)})$$

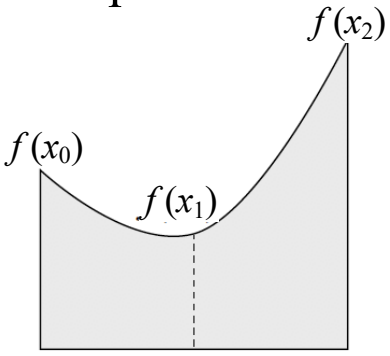


Closed Newton-Cotes formulas



Trapezoidal

Trapezoidal rule: $$\int_{x_0}^{x_1} f(x)dx = h\left(\frac{1}{2}f_0 + \frac{1}{2}f_1\right) + O(h^3 f'')$$



Simpson

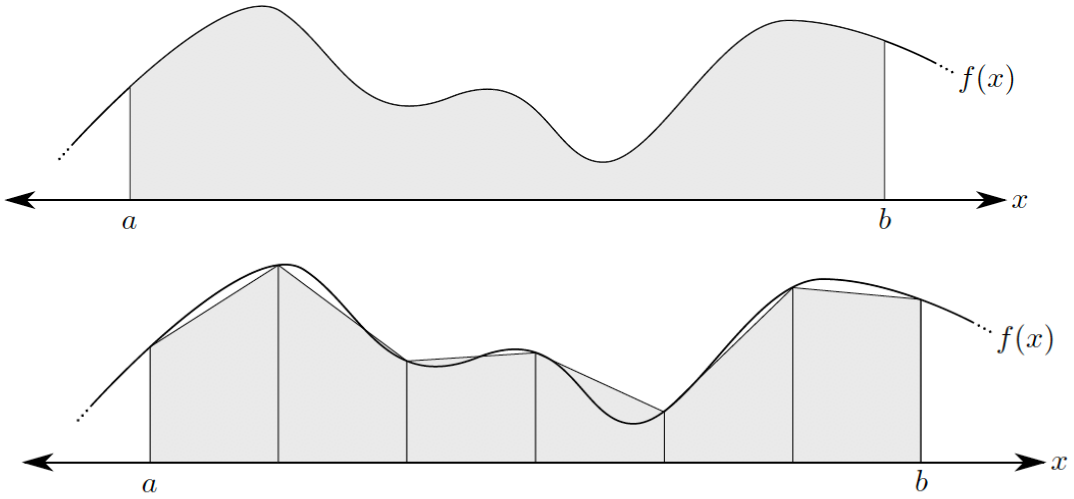
Simpson's rule: $$\int_{x_0}^{x_2} f(x)dx = h\left(\frac{1}{3}f_0 + \frac{4}{3}f_1 + \frac{1}{3}f_2\right) + O(h^5 f^{(4)})$$

lucky cancelation
leads to higher order!

Simpson's 3/8 rule: $$\int_{x_0}^{x_3} f(x)dx = h\left(\frac{3}{8}f_0 + \frac{9}{8}f_1 + \frac{9}{8}f_2 + \frac{3}{8}f_4\right) + O(h^5 f^{(4)})$$

Composite (extended) Newton-Cotes integration

To be more accurate, divide the interval $[a,b]$ into N equally-spaced steps, then apply one of the rules on each step.



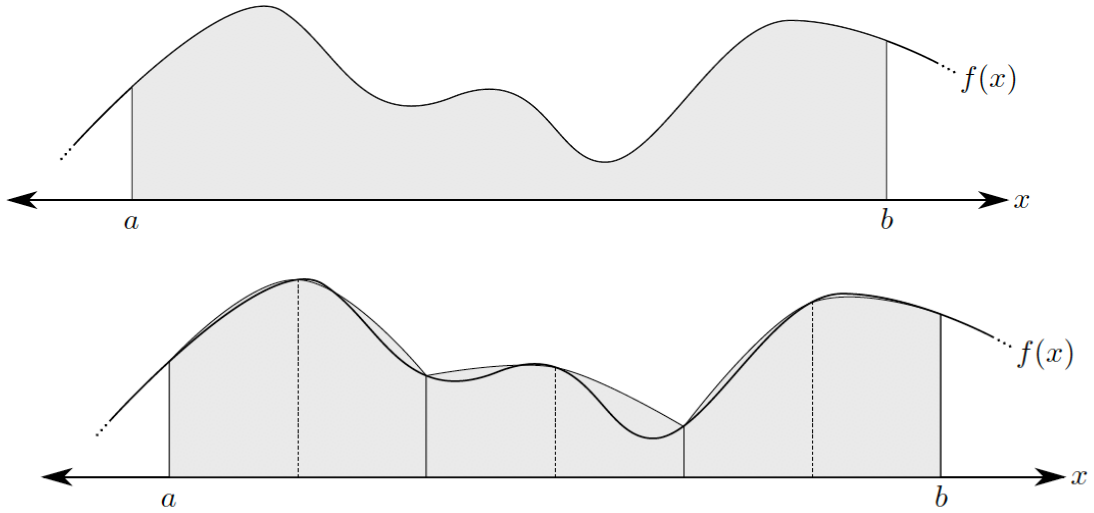
Composite trapezoidal rule

$$\int_a^b f(x)dx = \frac{b-a}{N}\left(\frac{1}{2}f_0 + f_1 + f_2 + \cdots + f_{N-1} + \frac{1}{2}f_N\right) + O\left[\frac{(b-a)^3}{N^2}f''\right]$$

2nd order accurate

Composite (extended) Newton-Cotes integration

To be more accurate, divide the interval $[a,b]$ into N equally-spaced steps, then apply one of the rules on each step.



Composite Simpson's rule

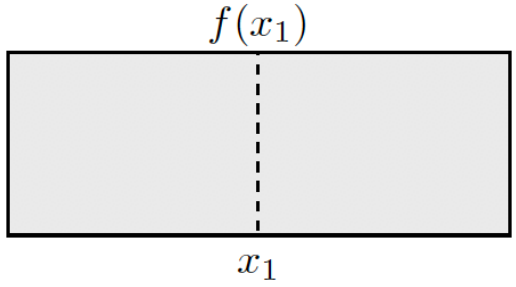
$$\int_a^b f(x)dx = \frac{b-a}{N}\left(\frac{1}{3}f_0+\frac{4}{3}f_1+\frac{2}{3}f_2+\frac{4}{3}f_3+\cdots+\frac{2}{3}f_{N-2}+\frac{4}{3}f_{N-1}+\frac{1}{3}f_N\right)+O\left[\frac{(b-a)^5}{N^4}f^{(4)}\right]$$

4th order accurate

Open formulas

A special open formula is the midpoint rule:

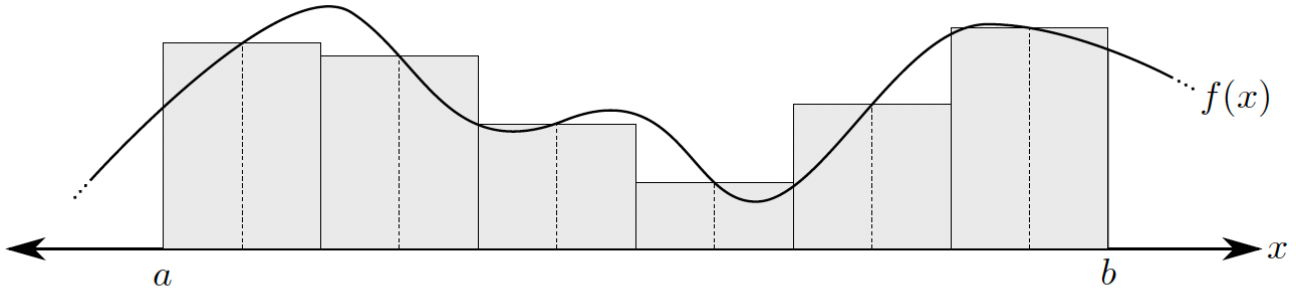
$$\int_a^b f(x)\,dx \approx (b-a) f\left(\frac{a+b}{2}\right).$$



Midpoint rule

Composite (extended) midpoint rule:

$$\int_a^b f(x)dx = \frac{b-a}{N}\left(f_{1/2} + f_{3/2} + f_{5/2} + \cdots + f_{(N-1)/2}\right) + O\left[\frac{(b-a)^3}{N^2} f''\right]$$



Composite midpoint rule (6 samples)

Romberg integration

An important property of the Trapezoidal rule is that its truncation error has only terms in even powers of 1/N:

$$\int_a^b f(x)dx = \boxed{\frac{b-a}{N}\left(\frac{1}{2}f_0+f_1+\cdots+f_{N-1}+\frac{1}{2}f_N\right)} + \frac{A}{N^2}(f'_N-f'_0)+\frac{B}{N^4}(f^{(3)}_N-f^{(3)}_0)+\cdots$$

I_N

where A, B, etc. are fixed numbers.

We can refine by calculating I_{2N}, then the leading error term is 1/4 of that in I_N.

We can combine them as $\frac{4}{3}I_{2N}-\frac{1}{3}I_N$ ⇨ Error in leading order cancels, leaving an error term that scales as $O(1/N^4)$!

In fact, we arrive at the composite Simpson's rule (with 2N points)!

The generic process of combining two lower order estimates at different grid spacings to get a higher order estimate is called Richardson extrapolation.

Romberg integration

More generically, let I be the accurate result of the integral, we may write:

$$I_0 = I - \frac{c_1}{N^2} - \frac{c_2}{N^4} - \frac{c_3}{N^6} + \cdots$$

$$I_1 = I - \frac{c_1}{4N^2} - \frac{c_2}{16N^4} - \frac{c_3}{64N^6} + \cdots$$

$$I_2 = I - \frac{c_1}{16N^2} - \frac{c_2}{256N^4} - \frac{c_3}{4096N^6} + \cdots$$

Combine 2nd order estimates to generate 4th order estimate:

$$I_{01} = \frac{4I_1 - I_0}{3} = I + \frac{c_2}{4N^4} + \frac{5c_3}{16N^6} + \cdots$$

$$I_{12} = \frac{4I_2 - I_1}{3} = I + \frac{c_2}{64N^4} + \frac{5c_3}{1024N^6} + \cdots$$

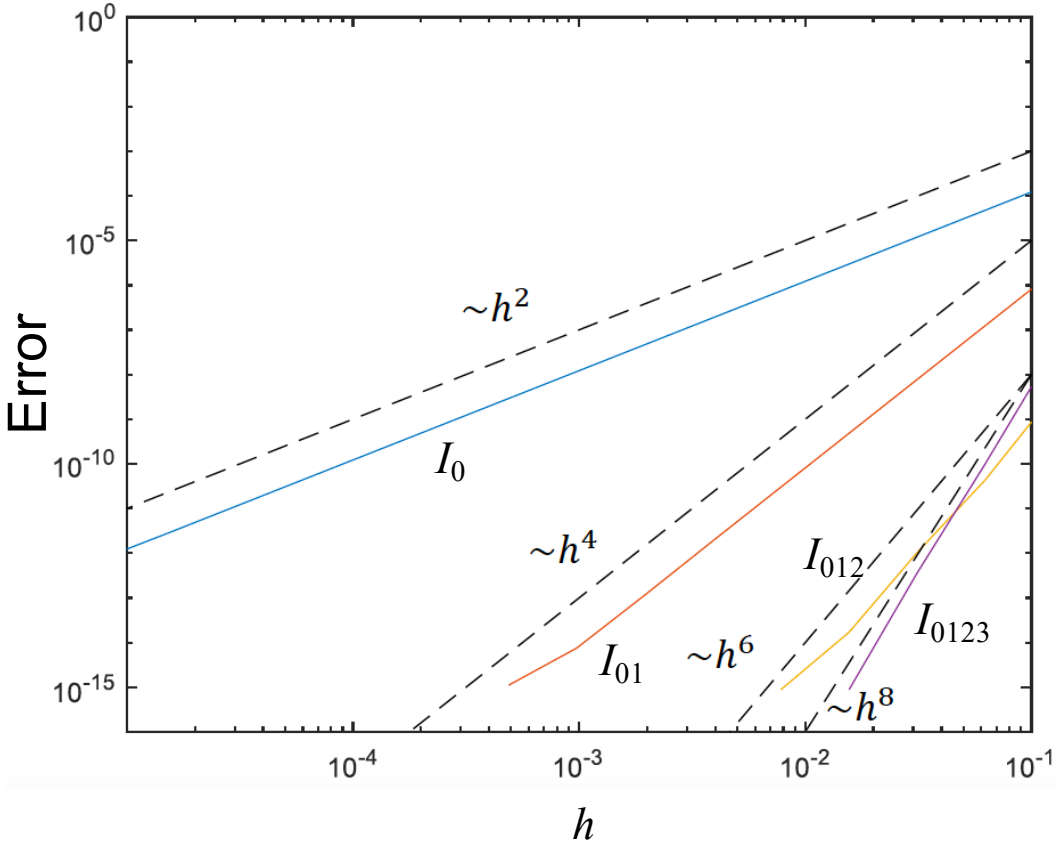
Combine 4th order estimates to generate 6th order estimate:

$$I_{012} = \frac{16I_{12} - I_{01}}{15} = I - \frac{c_3}{64N^6} + \cdots$$

This process can continue, until it reaches the desired precision.

Example

Consider: $I = \int_a^b e^{-x^2} dx = \frac{\sqrt{\pi}}{2}[\mathrm{erf}(b) - \mathrm{erf}(a)]\ , \quad \text{with } a = -2,\ b = 2.$



Romberg integration with midpoint

Midpoint rule shares similar error properties to Trapezoidal rule:

$$\int_a^b f(x)dx = \frac{b-a}{N}\left(f_{1/2}+f_{3/2}+\cdots+f_{(N-1)/2}\right)+\frac{A}{N^2}(f'_N-f'_0)+\frac{B}{N^4}(f^{(3)}_N-f^{(3)}_0)+\cdots$$

Can similarly do Richardson extrapolation towards higher order, but N increases by a factor of 3 at a time instead of 2:

$$I_0 = I - \frac{c_1}{N^2} - \frac{c_2}{N^4} - \cdots$$
$$I_1 = I - \frac{c_1}{9N^2} - \frac{c_2}{81N^4} - \cdots$$
$$I_2 = I - \frac{c_1}{81N^2} - \frac{c_2}{6561N^4} - \cdots$$

$$I_{01} = \frac{9I_1 - I_0}{8} = I + \frac{c_2}{9N^4} + \cdots$$
$$I_{12} = \frac{9I_2 - I_1}{8} = I + \frac{c_2}{729N^4} + \cdots$$
$$I_{012} = \frac{81I_{12} - I_{01}}{80} = I + O(N^{-6})$$

Improper integrals

Improper integrals include but are not limited to cases as follows:

$$\int_0^{\pi} \frac{\sin x}{x} dx \ , \quad \int_0^1 x^{-1/2} dx \ , \quad \int_0^{\infty} e^{-x} dx$$

When there are singularities involved (but still integrable), it is better to use open rather than closed formulas.

When the upper/lower limit is infinity, one could change variables to convert the interval into finite ranges. Here are some examples:

$$\int_a^b f(x)dx = \int_{1/b}^{1/a} \frac{1}{t^2} f\left(\frac{1}{t}\right) dt \qquad ab > 0 \ ; \qquad \int_{x=a}^{x=\infty} f(x)dx = \int_{t=0}^{t=e^{-a}} f(-\log t)\frac{dt}{t}$$

More examples in Chapter 4.5 of Numerical Recipes.

Gaussian quadrature

General procedure: $$\int_a^b W(x)f(x)dx = \sum_{i=0}^{N} w_i f(x_i)$$

We now abandon the requirement of evenly spaced abscissas.

Goal is to smartly choose the abscissas (x_i) and weights (w_i) to achieve best accuracy with fewer function evaluations.

We have 2(N+1) degrees of freedom. The optimal choice should be able to exactly integrate a polynomial of degree 2N+1.

That is, for smooth functions, convergence is exponential (rather than power-law) as N increases!

Gaussian quadrature provides a generic framework to do this.

Gaussian quadrature: basic principle

Define scalar product of two functions f and g as

$$\langle f|g\rangle \equiv \int_a^b W(x) f(x) g(x) dx$$

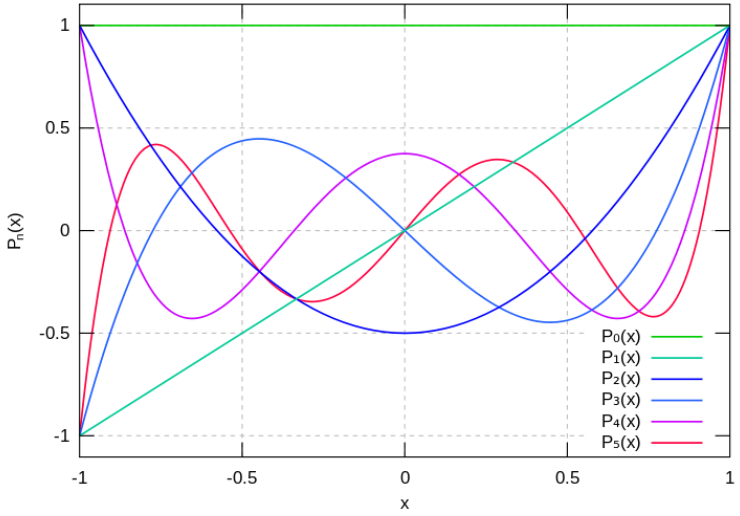
weight function

We can find, via a recursive relation, a set of polynomials, exactly one at each order j, $p_j(x)$, such that they are orthogonal to each other [and $p_{-1}(x)=0$, $p_0(x)=1$].

In the case of $W(x)=1$, $(a,b)=(-1,1)$, the recursive relation is

$$(j+1)P_{j+1} = (2j+1)xP_j - jP_{j-1}$$

This is the Legendre polynomial!



It can be shown that polynomial $p_j(x)$ has exactly j distinct roots in (a, b), which interleave with the previous j-1 roots of $p_{j-1}(x)$.

Gaussian quadrature: basic principle

In Gaussian quadrature, the abscissas x_i (i=0,…,N) are chosen to be the N+1 roots of polynomial $p_{N+1}(x)$.

For a polynomial $f(x)$ of degree 2N+1, Lagrange interpolation at these N+1 locations:

$$f(x) = L_N(x) + p_{N+1}(x)g_N(x) = \sum_{k=0}^{N} f(x_k)l_k(x) + p_{N+1}(x)g_N(x)$$

Therefore, $$\int_a^b W(x)f(x)dx = \sum_{k=0}^{N} f(x_k)\int_a^b W(x)l_k(x)dx + \int_a^b W(x)p_{N+1}(x)g_N(x)dx$$

weights =0 (orthogonality)

There are more handy ways to compute the weights: $$w_j = \frac{\langle p_N | p_N \rangle}{p_N(x_j)p'_{N+1}(x_j)}$$ (there are standard tables)

Gauss quadrature: varieties

There are a variety of "classic" weight functions, leading to:

Gauss-Legendre:

$$W(x) = 1 \qquad -1 < x < 1$$

$$(j+1)P_{j+1} = (2j+1)xP_j - jP_{j-1}$$

Gauss-Chebyshev:

$$W(x) = (1-x^2)^{-1/2} \qquad -1 < x < 1$$

$$T_{j+1} = 2xT_j - T_{j-1}$$

Gauss-Laguerre:

$$W(x) = x^{\alpha}e^{-x} \qquad 0 < x < \infty$$

$$(j+1)L_{j+1}^{\alpha} = (-x+2j+\alpha+1)L_j^{\alpha} - (j+\alpha)L_{j-1}^{\alpha}$$

Gauss-Hermite:

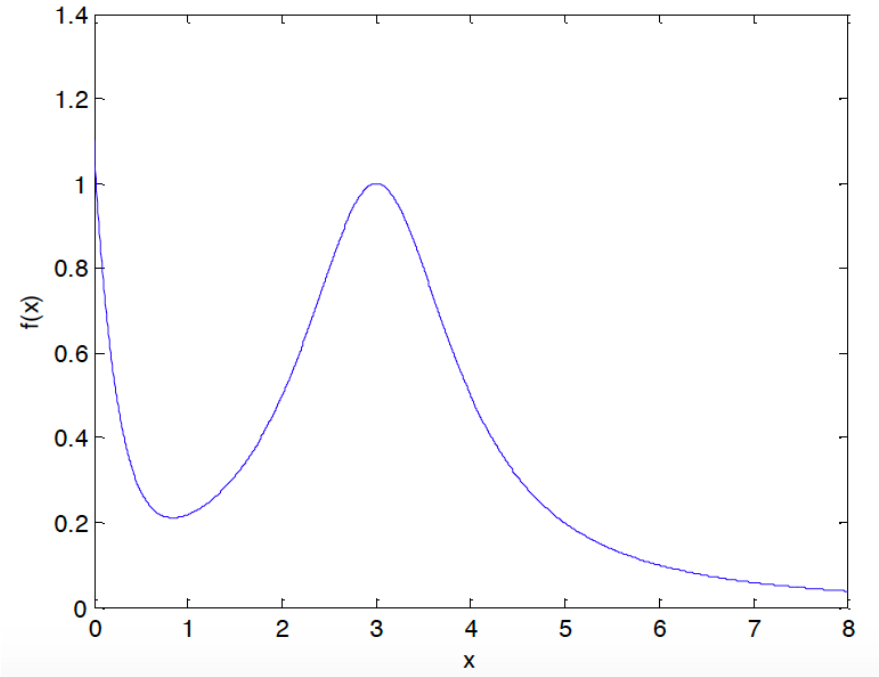
$$W(x) = e^{-x^2} \qquad -\infty < x < \infty$$

$$H_{j+1} = 2xH_j - 2jH_{j-1}$$

Gauss quadrature: example

Integrate this function: $f(x) = \frac{1}{(x-3)^2+1} + e^{-4x}$

$$\Rightarrow \int_a^b f(x)dx = \tan^{-1}(b) - \tan^{-1}(a) - \frac{1}{4}(e^{-4b} - e^{-4a})$$



Result:

ϵ	Uniform	Adaptive	Gauss
10^{-8}	1025	217	168
10^{-10}	2049	609	224
10^{-12}	8193	1513	360

(Simpson)

Higher-dimensions

Break multi-D integral into tensor product of 1D integrals.

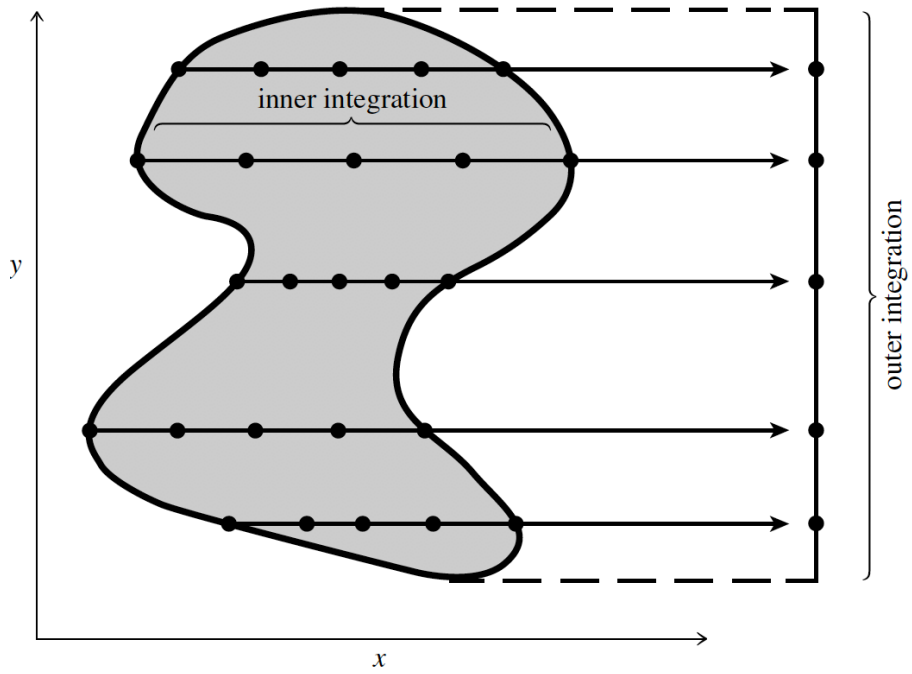
$$I \equiv \iiint dx\, dy\, dz\, f(x, y, z) = \int_{x_1}^{x_2} dx \int_{y_1(x)}^{y_2(x)} dy \int_{z_1(x,y)}^{z_2(x,y)} dz\, f(x, y, z)$$

For each dimension, use Newton-Cotes or Gaussian quadrature.

However, curse of dimensionality:

Need ~N^D functional evaluations.

Alternative approach: Monte Carlo (see later in this course).



Summary: numerical integration

- For simple applications, using composite Trapezoidal or Simpson's rules is generally sufficient.
- For more accurate approximations, knowing that the function is smooth, Romberg (Richardson extrapolation) integration and Gaussian quadrature are better choices.

Next lecture: experiment

I will upload the ppt for the next lecture (to be held on Sep. 29) today.

Please carefully study the slides before the next lecture.

[on numerical linear algibra but excluding the SVD part.]

We will experiment with a new teaching format in the next lecture:

- I will go through the slides much quicker than usual.
- I will ask questions about certain slides.
- You will form groups according to the seating to discuss them internally.
- I will randomly pick a group to address the questions.

Let's see how it works, and I will deliver a questionaire at the end of the class.