

Numerical Analysis

Final Review

SMIE SYSU

Chang-Dong Wang

Homepage: www.scholat.com/~ChangDongWang

Course website: www.scholat.com/course/SMIENA

Email: wangchd3@mail.sysu.edu.cn

QQ Group: 342983926

Outline

- Solving Equations
- Systems of Equations
- Interpolation
- Least Squares
- Numerical Differentiation and Integration

Solving Equations

- The bisection method
- Fixed-point iteration
- Newton's method
- Root-finding without derivatives

The bisection method

Bisection Method

Given initial interval $[a, b]$ such that $f(a)f(b) < 0$

while $(b - a)/2 > \text{TOL}$

$c = (a + b)/2$

if $f(c) = 0$, **stop**, **end**

if $f(a)f(c) < 0$

$b = c$

else

$a = c$

end

end

The final interval $[a, b]$ contains a root.

The approximate root is $(a + b)/2$.

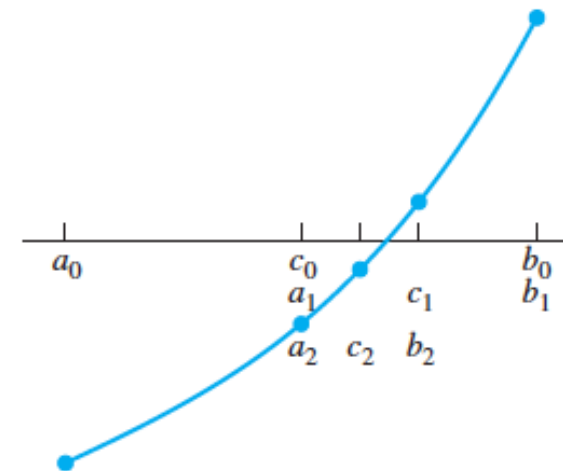


Figure 1.2 The Bisection Method. On the first step, the sign of $f(c_0)$ is checked.

Since $f(c_0)f(b_0) < 0$, set $a_1 = c_0, b_1 = b_0$, and the interval is replaced by the right half $[a_1, b_1]$. On the second step, the subinterval is replaced by its left half $[a_2, b_2]$.

Fixed-point iteration

- What is the fixed point of a function?

[FixedpointExample.m](#)

Definition:

The real number r is a **fixed point** of the function g if $g(r) = r$.

Fixed-Point Iteration

$x_0 = \text{initial guess}$

$x_{i+1} = g(x_i)$ for $i = 0, 1, 2, \dots$

The sequence x_i may or may not converge as the number of steps goes to infinity.

$$g(r) = g\left(\lim_{i \rightarrow \infty} x_i\right) = \lim_{i \rightarrow \infty} g(x_i) = \lim_{i \rightarrow \infty} x_{i+1} = r.$$

Fixed-point iteration

- Can every equation $f(x) = 0$ be turned into a fixed-point problem $g(x) = x$? Yes, and in many different ways. For instance, in the

$$x^3 + x - 1 = 0 \quad \longrightarrow \quad \begin{aligned} x &= 1 - x^3 \\ x &= \sqrt[3]{1 - x} \\ x &= \frac{1 + 2x^3}{1 + 3x^2} \end{aligned}$$

Fixed-point iteration

- Definition:

An iterative method is called **locally convergent** to r if the method converges to r for initial guesses sufficiently close to r . \square

- According to the previous theorem, Fixed-Point Iteration is locally convergent if $|g'(r)| < 1$.

$$g(x) = 1 - x^3$$

$$g'(x) = -3x^2$$

$$g(x) = \sqrt[3]{1-x}$$

$$g'(x) = 1/3(1-x)^{-2/3}(-1)$$

$$g(x) = (1 + 2x^3)/(1 + 3x^2)$$

$$g'(x) = \frac{6x(x^3 + x - 1)}{(1 + 3x^2)^2}$$

$$|g'(r)| \approx 1.3966 > 1,$$

$$\approx 0.716 < 1$$

$$= 0$$

Assume that g is continuously differentiable, that $g(r) = r$, and that $S = |g'(r)| < 1$. Then Fixed-Point Iteration converges linearly with rate S to the fixed point r for initial guesses sufficiently close to r . \blacksquare

Newton's method

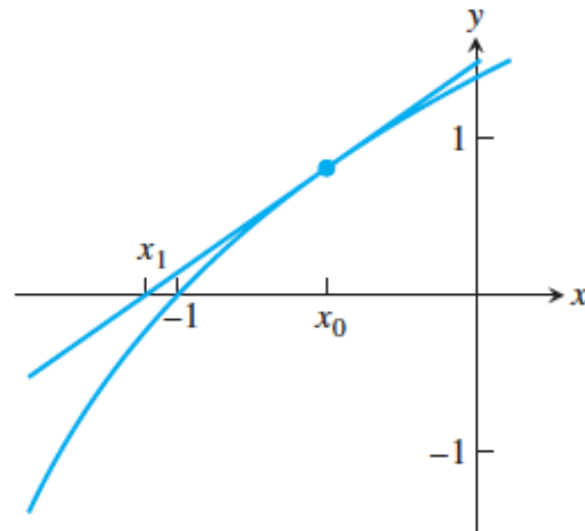


Figure 1.8 One step of Newton's Method. Starting with x_0 , the tangent line to the curve $y=f(x)$ is drawn. The intersection point with the x-axis is x_1 , the next approximation to the root.

$$x_0 = \text{initial guess}$$
$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \text{ for } i = 0, 1, 2, \dots$$

Root-finding without derivatives

- The Secant Method is similar to the Newton's Method, but replaces the derivative by a difference quotient $\frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}}$

Secant Method

$x_0, x_1 =$ initial guesses

$$x_{i+1} = x_i - \frac{f(x_i)(x_i - x_{i-1})}{f(x_i) - f(x_{i-1})} \text{ for } i = 1, 2, 3, \dots$$

- Unlike Fixed-Point Iteration and Newton's Method, **two starting guesses** are needed to begin the Secant Method.

superlinear convergence

Comparison

- To find the root of $x^3 + x - 1$ with 4 correct digits, e.g., 0.6823.

Methods		Iteration number	Side information	Convergence
Bisection method		9	No	Linearly
FPI	$g(x) = \sqrt[3]{1-x}$	25	No	Linearly
	$g(x) = (1 + 2x^3)/(1 + 3x^2)$	3	No	Linearly
Newton's method		5	Derivative	Quadratically
Secant method		6	No	Superlinearly

Outline

- Solving Equations
- Systems of Equations
- Interpolation
- Least Squares
- Numerical Differentiation and Integration

Systems of Equations

- Gaussian Elimination
- The LU Factorization
- Sources of Error
- The $PA = LU$ Factorization
- Iterative Methods
- Methods for symmetric positive-definite matrices
- Systems of Nonlinear Equations

Naive Gaussian elimination



- Exchange: Swap one equation for another.
- Replacement: Add or subtract a multiple of one equation from another.
- Scaling: Multiply an equation by a nonzero constant.

$$\begin{array}{l} x + y = 3 \\ 3x - 4y = 2 \end{array} \xrightarrow{\text{Elimination}} \begin{array}{l} x + y = 3 \\ -7y = -7 \end{array} \xrightarrow{\text{Back-substitution}} \begin{array}{l} -7y = -7 \rightarrow y = 1 \\ x + y = 3 \rightarrow x + (1) = 3 \rightarrow x = 2 \end{array}$$

Operation counts

Operation count for the elimination step of Gaussian elimination

The elimination step for a system of n equations in n variables can be completed in $\frac{2}{3}n^3 + \frac{1}{2}n^2 - \frac{7}{6}n$ operations.

Operation count for the back-substitution step of Gaussian elimination

The back-substitution step for a triangular system of n equations in n variables can be completed in n^2 operations.

- The time complexity of Naïve Gaussian elimination is $O(n^3)$.

The LU Factorization

- Consider the problem of solving a series of linear systems

$$Ax = b_1$$

$$Ax = b_2$$

$$\vdots$$

$$Ax = b_k$$

- How to avoid the redundant elimination steps used in Naïve Gaussian elimination?
- LU factorization.

Matrix form of Gaussian elimination

- Example: Find the LU factorization of $A = \begin{bmatrix} 1 & 2 & -1 \\ 2 & 1 & -2 \\ -3 & 1 & 1 \end{bmatrix}$

$$\begin{bmatrix} 1 & 2 & -1 \\ 2 & 1 & -2 \\ -3 & 1 & 1 \end{bmatrix} \xrightarrow[\text{from row 2}]{\text{subtract } 2 \times \text{row 1}} \begin{bmatrix} 1 & 2 & -1 \\ 0 & -3 & 0 \\ -3 & 1 & 1 \end{bmatrix} \xrightarrow[\text{from row 3}]{\text{subtract } -3 \times \text{row 1}} \begin{bmatrix} 1 & 2 & -1 \\ 0 & -3 & 0 \\ 0 & 7 & -2 \end{bmatrix}$$

$$L = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -3 & -\frac{7}{3} & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 2 & -1 \\ 0 & -3 & 0 \\ 0 & 0 & -2 \end{bmatrix} = U$$

Diagram illustrating the steps of Gaussian elimination and the construction of L and U matrices. Red dashed arrows show the elimination steps: from row 1 to row 2 (multiplier 2), from row 1 to row 3 (multiplier -3), and from row 2 to row 3 (multiplier -7/3). Blue solid arrows show the row operations performed on the matrix.

$$LU = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -3 & -\frac{7}{3} & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & -1 \\ 0 & -3 & 0 \\ 0 & 0 & -2 \end{bmatrix} = \begin{bmatrix} 1 & 2 & -1 \\ 2 & 1 & -2 \\ -3 & 1 & 1 \end{bmatrix} = A$$

Back substitution with the LU factorization

Once L and U are known, the problem $Ax = b$ can be written as $LUx = b$. Define a new “auxiliary” vector $c = Ux$. Then back substitution is a two-step procedure:

- (a) Solve $Lc = b$ for c .
- (b) Solve $Ux = c$ for x .

Both steps are straightforward since L and U are triangular matrices.

$$\text{LU} \quad \begin{bmatrix} 1 & 1 \\ 3 & -4 \end{bmatrix} = LU = \begin{bmatrix} 1 & 0 \\ 3 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 0 & -7 \end{bmatrix}$$

$$\text{Step (a)} \quad \begin{bmatrix} 1 & 0 \\ 3 & 1 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} 3 \\ 2 \end{bmatrix} \longrightarrow c_1 = 3, c_2 = -7$$

$$\text{Step (b)} \quad \begin{bmatrix} 1 & 1 \\ 0 & -7 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 3 \\ -7 \end{bmatrix} \longrightarrow \begin{array}{l} x_1 + x_2 = 3 \\ -7x_2 = -7 \end{array} \longrightarrow x_2 = 1, x_1 = 2.$$

Complexity of the LU factorization

That is, we are presented with the set of problems

$$Ax = b_1$$

$$Ax = b_2$$

$$\vdots$$

$$Ax = b_k$$

with various right-hand side vectors b_i

Methods	# Elimination operations	# Back-substitution operations
Naïve Gaussian	$2kn^3/3$	kn^2
LU factorization	$2n^3/3$	$2kn^2$

Sources of Error

- Two major potential sources of error in Gaussian elimination:
 - ✧ Error caused by ill-conditioning
 - ✧ Error caused by swamping

Swamping

- Example: Analysis

Overpower or Swamp

$$\left[\begin{array}{cc|c} 10^{-20} & 1 & 1 \\ 1 & 2 & 4 \end{array} \right] \xrightarrow[\text{from row 2}]{\text{subtract } 10^{20} \times \text{row 1}} \left[\begin{array}{cc|c} 10^{-20} & 1 & 1 \\ 0 & 2 - 10^{20} & 4 - 10^{20} \end{array} \right].$$

$$\left[\begin{array}{cc|c} 1 & 2 & 4 \\ 10^{-20} & 1 & 1 \end{array} \right] \xrightarrow[\text{from row 2}]{\text{subtract } 10^{-20} \times \text{row 1}} \left[\begin{array}{cc|c} 1 & 2 & 4 \\ 0 & 1 - 2 \times 10^{-20} & 1 - 4 \times 10^{-20} \end{array} \right].$$

The $PA = LU$ Factorization

- Partial pivoting
- Permutation matrices
- $PA=LU$ factorization

Partial pivoting

- **Recall:** In Gaussian elimination, the first step is to use the diagonal element a_{11} as a pivot to eliminate the first column.
- **Partial pivoting:**

Before eliminating column k , the p with $k \leq p \leq n$ and largest $|a_{pk}|$ is located, and rows p and k are exchanged if necessary before continuing with the elimination.

All multipliers will be no greater than 1 in absolute value! No more swamping problem!

PA = LU factorization

- What is PA=LU factorization?
- It is the matrix formulation of Gaussian elimination with partial pivoting.
- The PA= LU factorization is the established workhorse for solving systems of linear equations.
- Simply the LU factorization of a row-exchanged version of A.

PA = LU factorization

- Example

Find the PA=LU factorization of the matrix $A = \begin{bmatrix} 2 & 1 & 5 \\ 4 & 4 & -4 \\ 1 & 3 & 1 \end{bmatrix}$.

$$\begin{aligned}
 & \begin{bmatrix} 2 & 1 & 5 \\ 4 & 4 & -4 \\ 1 & 3 & 1 \end{bmatrix} \xrightarrow{\text{exchange rows 1 and 2}} \begin{bmatrix} 4 & 4 & -4 \\ 2 & 1 & 5 \\ 1 & 3 & 1 \end{bmatrix} \quad P = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \\
 & \xrightarrow{\substack{\text{subtract } \frac{1}{2} \times \text{row 1} \\ \text{from row 2}}} \begin{bmatrix} 4 & 4 & -4 \\ \frac{1}{2} & -1 & 7 \\ 1 & 3 & 1 \end{bmatrix} \xrightarrow{\substack{\text{subtract } \frac{1}{4} \times \text{row 1} \\ \text{from row 3}}} \begin{bmatrix} 4 & 4 & -4 \\ \frac{1}{2} & -1 & 7 \\ \frac{1}{4} & 2 & 2 \end{bmatrix}
 \end{aligned}$$

PA = LU factorization

- Example (cont...)

$$P = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

→ exchange rows 2 and 3 →

$$\begin{bmatrix} 4 & 4 & -4 \\ \left(\frac{1}{4}\right) & 2 & 2 \\ \left(\frac{1}{2}\right) & -1 & 7 \end{bmatrix}$$

→ subtract $-\frac{1}{2} \times$ row 2 from row 3 →

$$\begin{bmatrix} 4 & 4 & -4 \\ \left(\frac{1}{4}\right) & 2 & 2 \\ \left(\frac{1}{2}\right) & \left(-\frac{1}{2}\right) & 8 \end{bmatrix}.$$

$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 2 & 1 & 5 \\ 4 & 4 & -4 \\ 1 & 3 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{4} & 1 & 0 \\ \frac{1}{2} & -\frac{1}{2} & 1 \end{bmatrix} \begin{bmatrix} 4 & 4 & -4 \\ 0 & 2 & 2 \\ 0 & 0 & 8 \end{bmatrix}$$

$P \qquad A \qquad L \qquad U$

PA = LU factorization

Multiply through the equation $Ax = b$ by P on the left, and then proceed as before:

$$PAx = Pb$$

$$LUx = Pb.$$

Solve

1. $Lc = Pb$ for c .
2. $Ux = c$ for x .

PA = LU factorization

- Example

Use the PA=LU factorization to solve the system $Ax = b$, where

$$A = \begin{bmatrix} 2 & 1 & 5 \\ 4 & 4 & -4 \\ 1 & 3 & 1 \end{bmatrix}, \quad b = \begin{bmatrix} 5 \\ 0 \\ 6 \end{bmatrix}.$$

The PA=LU factorization is known from (2.22). It remains to complete the two back substitutions. 1. $Lc = Pb$:

$$\begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{4} & 1 & 0 \\ \frac{1}{2} & -\frac{1}{2} & 1 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 5 \\ 0 \\ 6 \end{bmatrix} = \begin{bmatrix} 0 \\ 6 \\ 5 \end{bmatrix}.$$

$c_1 = 0$

$$\frac{1}{4}(0) + c_2 = 6 \Rightarrow c_2 = 6$$

$$\frac{1}{2}(0) - \frac{1}{2}(6) + c_3 = 5 \Rightarrow c_3 = 8.$$

PA = LU factorization

- Example (cont...)

2. $Ux = c$:

$$\begin{bmatrix} 4 & 4 & -4 \\ 0 & 2 & 2 \\ 0 & 0 & 8 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 6 \\ 8 \end{bmatrix}$$

Starting at the bottom,

$$\begin{aligned} 8x_3 &= 8 \Rightarrow x_3 = 1 \\ 2x_2 + 2(1) &= 6 \Rightarrow x_2 = 2 \\ 4x_1 + 4(2) - 4(1) &= 0 \Rightarrow x_1 = -1. \end{aligned} \tag{2.25}$$

Therefore, the solution is $x = [-1, 2, 1]$.

[TestLu.m](#)

Comparison

- In the case of single equation, we have learned the Bisection method, the Fixed-Point-Iteration (FPI) method, the Newton's method and the Secant method.
- **Q&A:** What's major difference between **the methods of Bisection, FPI, Newton, Secant**, and **the Gaussian elimination a.w.a. its extension?** (refer to the textbook, three minutes)
- **ONLY single equation v.s. system of equations?**

Iterative Methods

- Two types of methods for solving systems:
 1. **Direct method** gives the exact solution within a finite number of steps (e.g. Gaussian elimination complexity $O(n^3)$), no convergence issue.
 2. **Iterative method** begins with an initial guess and refines the guess at each step, converging to the solution vector (like Fixed-Point Iteration in single equation).

Iterative Methods

- Jacobi Method
- Comparing Iterative methods and Direct methods from the perspective of **Sparse matrix computations**

Jacobi Method

- A form of fixed-point iteration for a system of equations with the following steps:
 1. Initial step: rewrite the equations to solve for the unknown, i.e., solving the i th equation for the i th unknown.
 2. Iterative step: iterate as in Fixed-Point Iteration, starting with an initial guess.

Jacobi Method

- Jacobi v.s. Fixed-Point-Iteration (FPI)
 1. Both of them are in some way finding fixed-points.
 2. Different rewrite results in different convergence property (Recall the three FPI formulas and the two matrix equations)
 3. In FPI, there is only one equation, we can easily rewrite the formula as input to FPI. **However in Jacobi, is it still very convenient to rewrite the formula as input to FPI??? No!**

Jacobi Method

- $A = L + D + U$
 - L : the lower triangle of A (entries below the main diagonal)
 - D : the main diagonal of A
 - U : the upper triangle (entries above the main diagonal)

$$Ax = b$$

$$(D + L + U)x = b$$

$$Dx = b - (L + U)x$$

$$x = D^{-1}(b - (L + U)x).$$

Jacobi Method

- Jacobi iteration:

$$x_0 = \text{initial vector}$$
$$x_{k+1} = D^{-1}(b - (L + U)x_k) \quad \text{for } k = 0, 1, 2, \dots$$

- For example: $\begin{bmatrix} 3 & 1 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} 5 \\ 5 \end{bmatrix}$

$$\begin{aligned} \begin{bmatrix} u_{k+1} \\ v_{k+1} \end{bmatrix} &= D^{-1}(b - (L + U)x_k) \\ &= \begin{bmatrix} 1/3 & 0 \\ 0 & 1/2 \end{bmatrix} \left(\begin{bmatrix} 5 \\ 5 \end{bmatrix} - \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} u_k \\ v_k \end{bmatrix} \right) \\ &= \begin{bmatrix} (5 - v_k)/3 \\ (5 - u_k)/2 \end{bmatrix}, \end{aligned}$$

Iterative methods vs Direct methods

- Two advantages of Iterative methods over Direct methods:
 1. When a good approximation to the solution is already known;
 2. Solve sparse systems of equations

In iterative methods, the number of operations in each iteration is only $O(n^2)$;

In direct methods, the Gaussian-elimination takes $O(n^3)$ operations.
If the number of iterations is far less than n , then iterative methods WINS!

Iterative methods vs Direct methods

- Scenario one: When a good approximation to the solution is already known
 1. Suppose that a solution to $Ax = b$ is known.
 2. A and/or b change by a small amount, e.g. in dynamic system.
 3. Using the solution to the previous problem as initial guess for Jacobi or Gauss-Seidel will converge very fast!

This technique is called **polishing**!

Notice that direct methods cannot reuse the solution to the previous problem but have to restart from scratch!

Nonlinear Systems of Equations

- Multivariate Newton's Method
- ~~Broyden's Method~~

Multivariate Newton's Method

Multivariate Newton's Method

$x_0 = \text{initial vector}$

$$x_{k+1} = x_k - (DF(x_k))^{-1} F(x_k) \quad \text{for } k = 0, 1, 2, \dots$$

- Need to find the inverse of Jacobian matrix.

$$s = (DF(x_k))^{-1} F(x_k)$$

$$DF(x_k)s = F(x_k)$$

- After getting s , we have the following iterative

$x_0 = \text{initial vector}$

$$\begin{cases} DF(x_k)s = -F(x_k) \\ x_{k+1} = x_k + s. \end{cases} \quad (2.51)$$

Outline

- Solving Equations
- Systems of Equations
- Interpolation
- Least Squares
- Numerical Differentiation and Integration

Interpolation

- Data and Interpolating Functions
- Chebyshev Interpolation
- Cubic Splines
- Bézier Curves

Interpolation

- The 1st question is what is the interpolation?
- Interpolation can be viewed as data compression, i.e., approximating data by a function, e.g., polynomial.
- A function is said to interpolate a set of data points if it passes through those points.
- Finding a **polynomial** through the set of data means replacing the information with **a rule** that can be evaluated **in a finite number of steps**.

Data and Interpolating Functions

- Interpolating polynomial: the function is a polynomial.
- There always exists a polynomial given a set of data points with distinct x-coordinates.
- Interpolating polynomial is the reverse of polynomial evaluation.
- Why use polynomials? Straightforward mathematical properties: only adding and multiplying, which are fundamental in computer hardware.

Data and Interpolating Functions

- Lagrange interpolation
- Newton's divided differences
- How many degree d polynomials pass through n points?

Lagrange interpolation

Assume that n data points $(x_1, y_1), \dots, (x_n, y_n)$ are given, and that we would like to find an interpolating polynomial. There is an explicit formula, called the Lagrange interpolating formula, for writing down a polynomial of degree $d = n - 1$ that interpolates the points. For example, suppose that we are given three points $(x_1, y_1), (x_2, y_2), (x_3, y_3)$. Then the polynomial

$$P_2(x) = y_1 \frac{(x - x_2)(x - x_3)}{(x_1 - x_2)(x_1 - x_3)} + y_2 \frac{(x - x_1)(x - x_3)}{(x_2 - x_1)(x_2 - x_3)} + y_3 \frac{(x - x_1)(x - x_2)}{(x_3 - x_1)(x_3 - x_2)} \quad (3.1)$$

is the **Lagrange interpolating polynomial** for these points.

Main Theorem of Polynomial Interpolation. Let $(x_1, y_1), \dots, (x_n, y_n)$ be n points in the plane with distinct x_i . Then there exists one and only one polynomial P of degree $n - 1$ or less that satisfies $P(x_i) = y_i$ for $i = 1, \dots, n$. ■

Newton's divided differences

Newton's divided difference formula

$$\begin{aligned} P(x) = & f[x_1] + f[x_1 \ x_2](x - x_1) \\ & + f[x_1 \ x_2 \ x_3](x - x_1)(x - x_2) \\ & + f[x_1 \ x_2 \ x_3 \ x_4](x - x_1)(x - x_2)(x - x_3) \\ & + \dots \\ & + f[x_1 \ \dots \ x_n](x - x_1) \dots (x - x_{n-1}). \end{aligned} \quad (3.2)$$

- How to calculate the coefficient $f[x_1 \ x_2 \ \dots \ x_n]$?
You may try to substitute the data points into (3.2) and see how to find the coefficient...

Newton's divided differences

the coefficients $f[x_1 \dots x_k]$ from the above definition can be recursively calculated as follows. List the data points in a table:

x_1	$f(x_1)$
x_2	$f(x_2)$
\vdots	\vdots
x_n	$f(x_n)$.

Now define the divided differences, which are the real numbers

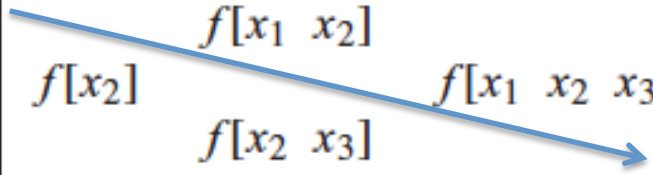
$$\begin{aligned}
 f[x_k] &= f(x_k) \\
 f[x_k \ x_{k+1}] &= \frac{f[x_{k+1}] - f[x_k]}{x_{k+1} - x_k} \\
 f[x_k \ x_{k+1} \ x_{k+2}] &= \frac{f[x_{k+1} \ x_{k+2}] - f[x_k \ x_{k+1}]}{x_{k+2} - x_k} \\
 f[x_k \ x_{k+1} \ x_{k+2} \ x_{k+3}] &= \frac{f[x_{k+1} \ x_{k+2} \ x_{k+3}] - f[x_k \ x_{k+1} \ x_{k+2}]}{x_{k+3} - x_k}, \quad (3.3)
 \end{aligned}$$

and so on. - -

Newton's divided differences

The recursive definition of the Newton's divided differences allows arrangement into a convenient table. For three points the table has the form

x_1	$f[x_1]$		
		$f[x_1 \ x_2]$	
x_2	$f[x_2]$		$f[x_1 \ x_2 \ x_3]$
		$f[x_2 \ x_3]$	
x_3	$f[x_3]$		



The coefficients of the polynomial (3.2) can be read from the top edge of the triangle.

How many degree d polynomials pass through n points?

- For n points, there is **one and only one** interpolating polynomial of degree $n-1$ or less, according to Main Theorem of Polynomial Interpolation.
- For n points, there are infinitely many interpolating polynomials of degree n or greater: Just add the **extract polynomial of the desired degree** that passes the n points to **the existing polynomial**.

Any degree 3 polynomial of the form $P_3(x) = P_2(x) + cx(x-2)(x-3)$ with $c \neq 0$ will pass through $(0, 1)$, $(2, 2)$, and $(3, 4)$.

Chebyshev Interpolation

- Recall that in the previous examples, we choose evenly spaced base points.
- However, the choice of base point spacing has a significant effect on the interpolation error (see Eq. (3.6)).
$$f(x) - P(x) = \frac{(x - x_1)(x - x_2) \cdots (x - x_n)}{n!} f^{(n)}(c)$$
- Chebyshev interpolation refers to a particular optimal way of spacing the base points so as to reduce the interpolation error.

Chebyshev's theorem

- Theorem

The choice of real numbers $-1 \leq x_1, \dots, x_n \leq 1$ that makes the value of

$$\max_{-1 \leq x \leq 1} |(x - x_1) \cdots (x - x_n)|$$

as small as possible is

$$x_i = \cos \frac{(2i - 1)\pi}{2n} \quad \text{for } i = 1, \dots, n,$$

and the minimum value is $1/2^{n-1}$. In fact, the minimum is achieved by

$$(x - x_1) \cdots (x - x_n) = \frac{1}{2^{n-1}} T_n(x), \quad \text{Chebyshev roots}$$

where $T_n(x)$ denotes the degree n Chebyshev polynomial.

$$x_i = \cos \frac{\text{odd } \pi}{2n}$$

We will call the interpolating polynomial that uses the Chebyshev roots as base points the

Chebyshev interpolating polynomial.

Chebyshev polynomials

Define the n th Chebyshev polynomial by $T_n(x) = \cos(n \arccos x)$.

for $n = 0$ it gives the degree 0 polynomial 1

for $n = 1$ we get $T_1(x) = \cos(\arccos x) = x$.

For $n = 2$ $T_2(x) = \cos 2y = \cos^2 y - \sin^2 y = 2\cos^2 y - 1 = 2x^2 - 1$

$\cos(a + b) = \cos a \cos b - \sin a \sin b$. Set $y = \arccos x$, so that $\cos y = x$.

In general, note that

$$T_{n+1}(x) = \cos(n + 1)y = \cos(ny + y) = \cos ny \cos y - \sin ny \sin y$$

$$T_{n-1}(x) = \cos(n - 1)y = \cos(ny - y) = \cos ny \cos y - \sin ny \sin(-y).$$

Because $\sin(-y) = -\sin y$, we can add the preceding equations to get

$$T_{n+1}(x) + T_{n-1}(x) = 2\cos ny \cos y = 2xT_n(x).$$



$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x)$, **recursion relation** for the Chebyshev polynomials.

Cubic Splines

- **Polynomial** interpolation uses a **single** formula, given by a polynomial, to meet all data points.
- **Spline** interpolation uses **several** formulas, each a low-degree polynomial, to pass through the data points.

Properties of splines

To be a little more precise about the properties of a cubic spline, we make the following definition: Assume that we are given the n data points $(x_1, y_1), \dots, (x_n, y_n)$, where the x_i are distinct and in increasing order. A **cubic spline** $S(x)$ through the data points $(x_1, y_1), \dots, (x_n, y_n)$ is a set of cubic polynomials

$$\begin{aligned} S_1(x) &= y_1 + b_1(x - x_1) + c_1(x - x_1)^2 + d_1(x - x_1)^3 \text{ on } [x_1, x_2] \\ S_2(x) &= y_2 + b_2(x - x_2) + c_2(x - x_2)^2 + d_2(x - x_2)^3 \text{ on } [x_2, x_3] \\ &\vdots \end{aligned} \tag{3.17}$$

$$S_{n-1}(x) = y_{n-1} + b_{n-1}(x - x_{n-1}) + c_{n-1}(x - x_{n-1})^2 + d_{n-1}(x - x_{n-1})^3 \text{ on } [x_{n-1}, x_n]$$

with the following properties:

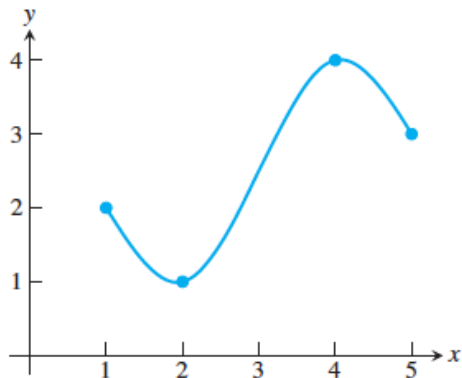
Property 1 $S_i(x_i) = y_i$ and $S_i(x_{i+1}) = y_{i+1}$ for $i = 1, \dots, n - 1$.

Property 2 $S'_{i-1}(x_i) = S'_i(x_i)$ for $i = 2, \dots, n - 1$.

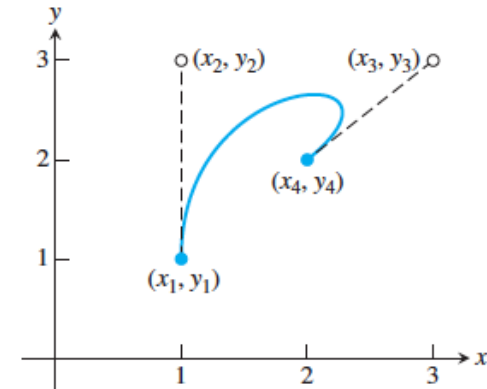
Property 3 $S''_{i-1}(x_i) = S''_i(x_i)$ for $i = 2, \dots, n - 1$.

Bezier Curves

- Recall that in cubic spline, each piece of splines are determined to as to ensure the smoothness of the first and second derivatives across the knot.
- Bezier curves are splines that allow the user to control the slopes at the knot with the **loss** of the smoothness of the first and second derivatives across the knot: suitable for cases where corners (discontinuous first derivatives) and abrupt changes in curvature (discontinuous second derivatives) are occasionally needed.

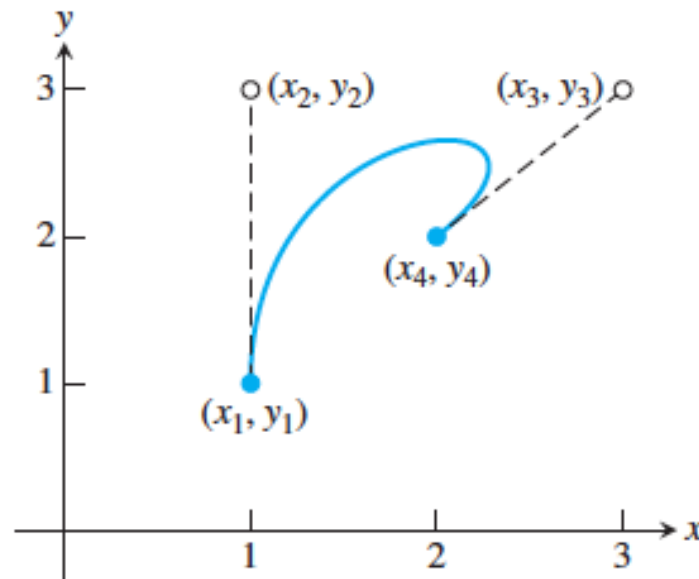


SMIE Dr. Wang



Bezier Curves

- Basic idea: Each **piece** of a planar Bezier spline is determined by four points $(x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4)$.
- End points: $(x_1, y_1), (x_4, y_4)$; Control points: $(x_2, y_2), (x_3, y_3)$



The curve leaves

(x_1, y_1) along the tangent direction $(x_2 - x_1, y_2 - y_1)$ and ends at (x_4, y_4) along the tangent direction $(x_4 - x_3, y_4 - y_3)$. The equations that accomplish this are expressed as a parametric curve $(x(t), y(t))$ for $0 \leq t \leq 1$.

Bezier Curves

Bézier curve

Given endpoints $(x_1, y_1), (x_4, y_4)$
control points $(x_2, y_2), (x_3, y_3)$

Set

$$b_x = 3(x_2 - x_1)$$

$$c_x = 3(x_3 - x_2) - b_x$$

$$d_x = x_4 - x_1 - b_x - c_x$$

$$b_y = 3(y_2 - y_1)$$

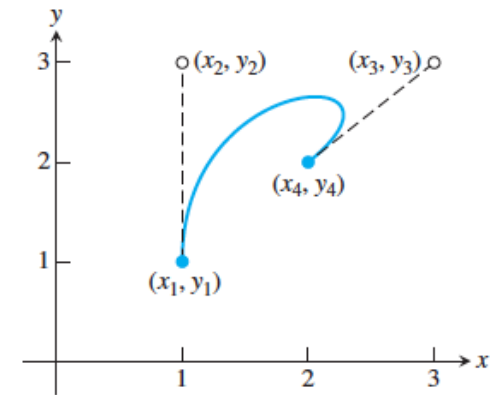
$$c_y = 3(y_3 - y_2) - b_y$$

$$d_y = y_4 - y_1 - b_y - c_y.$$

The Bézier curve is defined for $0 \leq t \leq 1$ by

$$x(t) = x_1 + b_x t + c_x t^2 + d_x t^3$$

$$y(t) = y_1 + b_y t + c_y t^2 + d_y t^3.$$



$$x(0) = x_1$$

$$x'(0) = 3(x_2 - x_1)$$

$$x(1) = x_4$$

$$x'(1) = 3(x_4 - x_3)$$

the analogous facts hold for $y(t)$.

Bezier Curves


- Why use these complicated formulas?

Linear Bezier curves through P_0 and P_1

$$\mathbf{B}(t) = \mathbf{P}_0 + t(\mathbf{P}_1 - \mathbf{P}_0) = (1-t)\mathbf{P}_0 + t\mathbf{P}_1, t \in [0, 1]$$

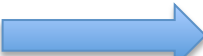
Quadratic Bezier curves through P_0 , P_1 and P_2 : A linear combination of the two linear Bezier curves.

$$\mathbf{B}(t) = (1-t)[(1-t)\mathbf{P}_0 + t\mathbf{P}_1] + t[(1-t)\mathbf{P}_1 + t\mathbf{P}_2], t \in [0, 1]$$


$$\mathbf{B}(t) = (1-t)^2\mathbf{P}_0 + 2(1-t)t\mathbf{P}_1 + t^2\mathbf{P}_2, t \in [0, 1].$$

Cubic Bezier curves through P_0 , P_1 , P_2 and P_3 : A linear combination of the two Quadratic Bezier curves.

$$\mathbf{B}(t) = (1-t)\mathbf{B}_{\mathbf{P}_0, \mathbf{P}_1, \mathbf{P}_2}(t) + t\mathbf{B}_{\mathbf{P}_1, \mathbf{P}_2, \mathbf{P}_3}(t), t \in [0, 1].$$


$$\mathbf{B}(t) = (1-t)^3\mathbf{P}_0 + 3(1-t)^2t\mathbf{P}_1 + 3(1-t)t^2\mathbf{P}_2 + t^3\mathbf{P}_3, t \in [0, 1].$$

This formula explains why the Bezier formula shown in the previous slide works.

Outline

- Solving Equations
- Systems of Equations
- Interpolation
- Least Squares
- Numerical Differentiation and Integration

Least Squares

- Least Squares and the Normal Equations
- QR Factorization
- Nonlinear Least Squares

Least Squares and the Normal Equations

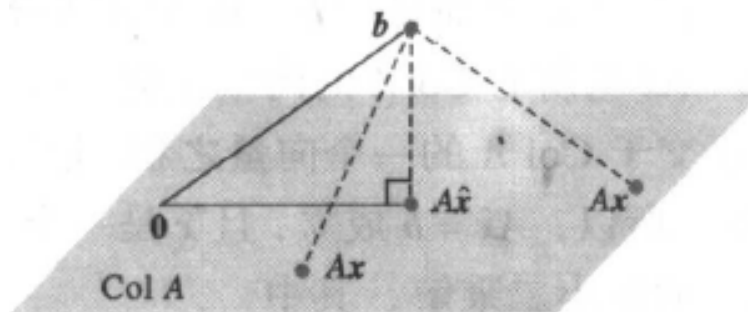
- **Problem:** What do we do when $A\mathbf{x} = \mathbf{b}$ has no solution \mathbf{x} , i.e., inconsistent?

Answer: Find $\hat{\mathbf{x}}$ such that $A\hat{\mathbf{x}}$ is as “close” as possible to \mathbf{b} . (*Least Squares Problem*)

If A is $m \times n$ and \mathbf{b} is in \mathbf{R}^m , a **least-squares solution** of $A\mathbf{x} = \mathbf{b}$ is an $\hat{\mathbf{x}}$ in \mathbf{R}^n such that

$$\|\mathbf{b} - A\hat{\mathbf{x}}\| \leq \|\mathbf{b} - A\mathbf{x}\|$$

for all \mathbf{x} in \mathbf{R}^n .

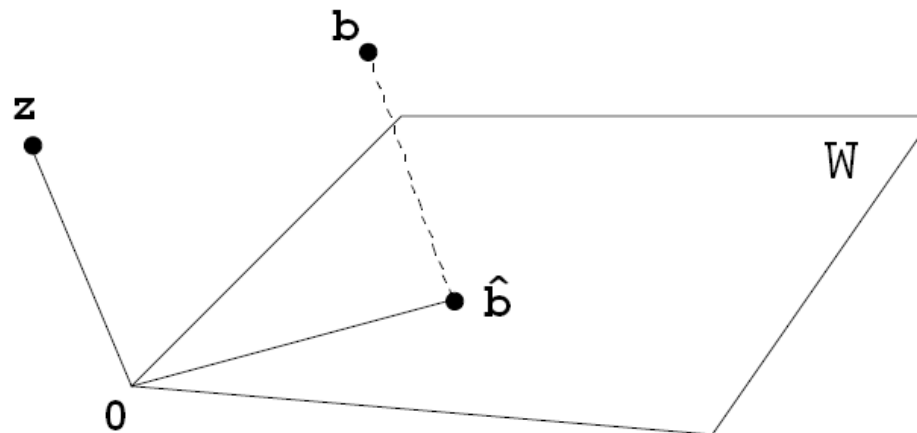


Least Squares and the Normal Equations S2

- Solution of the General Least-Squares Problem**

Let $W = \text{Col } A$ where A is $m \times n$ and $A = \begin{bmatrix} \mathbf{a}_1 & \mathbf{a}_2 & \cdots & \mathbf{a}_n \end{bmatrix}$.

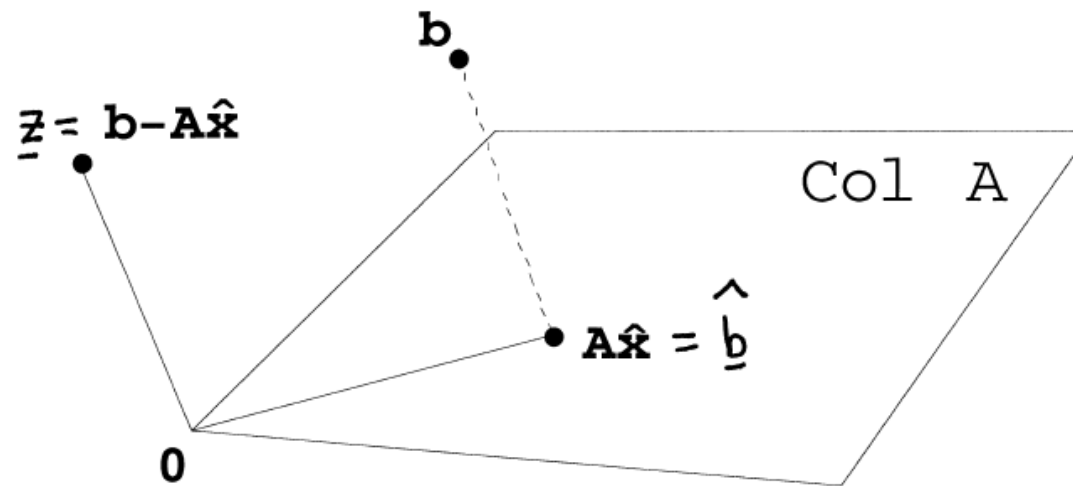
Suppose \mathbf{b} is in \mathbf{R}^m and $\hat{\mathbf{b}} = \text{proj}_W \mathbf{b}$.



$\hat{\mathbf{b}}$ is the point in $W = \text{Col } A$ closest to \mathbf{b}

Least Squares and the Normal Equations S3

Since $\hat{\mathbf{b}}$ is in $\text{Col } A$, then $\hat{\mathbf{x}}$ is a vector in \mathbf{R}^n such that $\hat{\mathbf{b}} = A\hat{\mathbf{x}}$.



Least Squares and the Normal Equations S4

By the Orthogonal Projection Theorem, \mathbf{z} is in W^\perp where $\mathbf{z} = \mathbf{b} - A\hat{\mathbf{x}}$.

Then $\mathbf{b} - A\hat{\mathbf{x}}$ is orthogonal to every column of A . Meaning that

$$\mathbf{a}_1^T(\mathbf{b} - A\hat{\mathbf{x}}) = 0 \quad \mathbf{a}_2^T(\mathbf{b} - A\hat{\mathbf{x}}) = 0 \quad \dots \quad \mathbf{a}_n^T(\mathbf{b} - A\hat{\mathbf{x}}) = 0$$

$$\begin{bmatrix} \mathbf{a}_1^T \\ \mathbf{a}_2^T \\ \vdots \\ \mathbf{a}_n^T \end{bmatrix} (\mathbf{b} - A\hat{\mathbf{x}}) = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

$$A^T(\mathbf{b} - A\hat{\mathbf{x}}) = \mathbf{0}$$

$$A^T\mathbf{b} - A^T A\hat{\mathbf{x}} = \mathbf{0}$$

$$\boxed{A^T A\hat{\mathbf{x}} = A^T \mathbf{b}}$$

(normal equations for $\hat{\mathbf{x}}$)

Least Squares and the Normal Equations S5

- **Solution of the General Least-Squares Problem (Another deduction in text book)**

Now we return to our search for a formula for \bar{x} . We have established that

$$(b - A\bar{x}) \perp \{Ax | x \in R^n\}.$$

Expressing the perpendicularity in terms of matrix multiplication, we find that

$$(Ax)^T (b - A\bar{x}) = 0 \text{ for all } x \text{ in } R^n.$$

Using the preceding fact about transposes, we can rewrite this expression as

$$x^T A^T (b - A\bar{x}) = 0 \text{ for all } x \text{ in } R^n,$$

meaning that the n -dimensional vector $A^T (b - A\bar{x})$ is perpendicular to every vector x in R^n , including itself. There is only one way for that to happen:

$$A^T (b - A\bar{x}) = 0.$$

Least Squares and the Normal Equations

This gives a system of equations that defines the least squares solution,

$$A^T A \bar{x} = A^T b. \quad (4.6)$$

The system of equations (4.6) is known as the **normal equations**. Its solution \bar{x} is the so-called least squares solution of the system $Ax = b$.

THEOREM 13

The set of least squares solutions of $A\mathbf{x} = \mathbf{b}$ is the set of all solutions of the normal equations $A^T A \hat{\mathbf{x}} = A^T \mathbf{b}$.

Least Squares and the Normal Equations

- Example

Use the normal equations to find the least squares solution of the inconsistent system (4.1).

$$A = \begin{bmatrix} 1 & 1 \\ 1 & -1 \\ 1 & 1 \end{bmatrix}, \quad b = \begin{bmatrix} 2 \\ 1 \\ 3 \end{bmatrix}.$$

The components of the normal equations are

$$A^T A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 3 & 1 \\ 1 & 3 \end{bmatrix} \quad A^T b = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -1 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 1 \\ 3 \end{bmatrix} = \begin{bmatrix} 6 \\ 4 \end{bmatrix}.$$

The normal equations $\begin{bmatrix} 3 & 1 \\ 1 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 6 \\ 4 \end{bmatrix}$

can now be solved by Gaussian elimination. The tableau form is

$$\left[\begin{array}{cc|c} 3 & 1 & 6 \\ 1 & 3 & 4 \end{array} \right] \longrightarrow \left[\begin{array}{cc|c} 3 & 1 & 6 \\ 0 & 8/3 & 2 \end{array} \right]$$

which can be solved to get $\bar{x} = (\bar{x}_1, \bar{x}_2) = (7/4, 3/4)$.

QR Factorization

- This section develops QR factorization as a way to solve least squares problem superior to the normal equations in **accuracy**.
- After introducing the factorization by way of Gram-Schmidt orthogonalization, we return to Example 4.5, for which the normal equations turned out to be inadequate.
- Recall: LU factorization is used to solve matrix equations superior to Gaussian elimination in **computational complexity** in a series of systems.

Gram-Schmidt orthogonalization and least squares

Classical Gram-Schmidt orthogonalization

Let $A_j, j = 1, \dots, n$ be linearly independent vectors.

```
for  $j = 1, 2, \dots, n$ 
   $y = A_j$ 
  for  $i = 1, 2, \dots, j - 1$ 
     $r_{ij} = q_i^T A_j$ 
     $y = y - r_{ij}q_i$ 
  end
   $r_{jj} = \|y\|_2$ 
   $q_j = y/r_{jj}$ 
end
```

$$(A_1 | \dots | A_n) = (q_1 | \dots | q_n) \begin{bmatrix} r_{11} & r_{12} & \cdots & r_{1n} \\ & r_{22} & \cdots & r_{2n} \\ & & \ddots & \vdots \\ & & & r_{nn} \end{bmatrix}$$

A is $m \times n$, Q is $m \times n$ and R is $n \times n$. Q may be not square, not to say orthogonal.

Gram-Schmidt orthogonalization and least squares

Least squares by QR factorization

Given the $m \times n$ inconsistent system

$$Ax = b,$$

find the full QR factorization $A = QR$ and set

$$\begin{aligned}\hat{R} &= \text{upper } n \times n \text{ submatrix of } R \\ \hat{d} &= \text{upper } n \text{ entries of } d = Q^T b\end{aligned}$$

Solve $\hat{R}\bar{x} = \hat{d}$ for least squares solution \bar{x} .

Gram-Schmidt orthogonalization and least squares

- Example


Use the full QR factorization to solve the least squares problem $\begin{bmatrix} 1 & -4 \\ 2 & 3 \\ 2 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} -3 \\ 15 \\ 9 \end{bmatrix}$.

We need to solve $Rx = Q^T b$, or

$$\begin{bmatrix} 3 & 2 \\ 0 & 5 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \frac{1}{15} \begin{bmatrix} 5 & 10 & 10 \\ -14 & 5 & 2 \\ 2 & 10 & -11 \end{bmatrix} \begin{bmatrix} -3 \\ 15 \\ 9 \end{bmatrix} = \begin{bmatrix} 15 \\ 9 \\ 3 \end{bmatrix}.$$

The least squares error will be $\|e\|_2 = \|(0, 0, 3)\|_2 = 3$. Equating the upper parts yields

$$\begin{bmatrix} 3 & 2 \\ 0 & 5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 15 \\ 9 \end{bmatrix},$$

whose solution is $\bar{x}_1 = 3.8, \bar{x}_2 = 1.8$. This least squares problem was solved by the normal equations in Example 4.2. 

Nonlinear Least Squares

- Gauss-Newton Method
- Models with nonlinear parameters
- ~~The Levenberg-Marquardt Method~~

Gauss–Newton Method

Gauss–Newton Method

To minimize

Which is equal to set $F(x)=0$.
 r is $m \times 1$, $F(x)$ is $1 \times n$ vector, Dr is $m \times n$.

$$r_1(x)^2 + \cdots + r_m(x)^2.$$

Set x^0 = initial vector,
 for $k = 0, 1, 2, \dots$

$$F(x)^T = (r^T Dr)^T = (Dr)^T r.$$

$$\begin{aligned} A &= Dr(x^k) \\ A^T A v^k &= -A^T r(x^k) \end{aligned} \tag{4.33}$$

$$x^{k+1} = x^k + v^k \tag{4.34}$$

end

$$\begin{cases} DF(x_k)s = -F(x_k) \\ x_{k+1} = x_k + s. \end{cases}$$

$$DF(x)^T = D((Dr)^T r) = (Dr)^T \cdot Dr + \sum_{i=1}^m r_i Dc_i$$

The Levenberg–Marquardt Method

Least squares minimization is especially challenging when the coefficient matrix turns out to be ill-conditioned. In Example 4.5, large errors were encountered in the least squares solution of $Ax = b$ when using the normal equations, since $A^T A$ had large condition number.

QR factorization

The problem is often worse for nonlinear least squares minimization. Many plausible model definitions yield poorly conditioned Dr matrices.

The Levenberg–Marquardt Method uses a “regularization term” to partially remedy the conditioning problem.

The Levenberg–Marquardt Method

Levenberg–Marquardt Method

To minimize

$$r_1(x)^2 + \cdots + r_m(x)^2.$$

Set x^0 = initial vector, λ = constant
for $k = 0, 1, 2, \dots$

$$\begin{aligned} A &= Dr(x^k) \\ (A^T A + \lambda \operatorname{diag}(A^T A)) v^k &= -A^T r(x^k) \\ x^{k+1} &= x^k + v^k \end{aligned}$$

end

The $\lambda = 0$ case is identical to Gauss–Newton. Increasing the regularization parameter λ accentuates the effect of the diagonal of the matrix $A^T A$, which improves the condition number and generally allows the method to converge from a broader set of initial guesses x_0 than Gauss–Newton.

Outline

- Solving Equations
- Systems of Equations
- Interpolation
- Least Squares
- Numerical Differentiation and Integration

Numerical Differentiation and Integration

- Numerical Differentiation
- Newton–Cotes Formulas for Numerical Integration

Numerical Differentiation

- Finite difference formulas
- Rounding error
- Extrapolation

Finite difference formulas

Two-point forward-difference formula

$$f'(x) = \frac{f(x+h) - f(x)}{h} - \frac{h}{2} f''(c), \quad (5.3)$$

where c is between x and $x+h$. The quotient will closely approximate the derivative if h is small.

We use (5.3) by computing the approximation

$$f'(x) \approx \frac{f(x+h) - f(x)}{h} \quad (5.4)$$

and treating the last term in (5.3) as error.

The first-order method for approximating the first derivative.

In general, if the error is $O(h^n)$, we call the formula an **order n** approximation.

Finite difference formulas

Three-point centered-difference formula

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} - \frac{h^2}{6} f'''(c), \quad (5.7)$$

where $x - h < c < x + h$.

Three-point centered-difference formula for second derivative

$$f''(x) = \frac{f(x-h) - 2f(x) + f(x+h)}{h^2} - \frac{h^2}{12} f^{(iv)}(c) \quad (5.8)$$

for some c between $x - h$ and $x + h$.

Rounding error

Denote the floating point version of the input $f(x + h)$ by $\hat{f}(x + h)$, which will differ from the correct value $f(x + h)$ by a number on the order of machine epsilon in relative terms.

Since $\hat{f}(x + h) = f(x + h) + \epsilon_1$ and $\hat{f}(x - h) = f(x - h) + \epsilon_2$, where $|\epsilon_1|, |\epsilon_2| \approx \epsilon_{\text{mach}}$, the difference between the correct $f'(x)$ and the machine version of the three-point centered-difference formula (5.7) is

$$\begin{aligned}
 f'(x)_{\text{correct}} - f'(x)_{\text{machine}} &= f'(x) - \frac{\hat{f}(x + h) - \hat{f}(x - h)}{2h} \\
 \text{Total error} \quad &= f'(x) - \frac{f(x + h) + \epsilon_1 - (f(x - h) + \epsilon_2)}{2h} \\
 &= \left(f'(x) - \frac{f(x + h) - f(x - h)}{2h} \right) + \frac{\epsilon_2 - \epsilon_1}{2h} \\
 &= \underbrace{\left(f'(x)_{\text{correct}} - f'(x)_{\text{formula}} \right)}_{\text{Truncation error}} + \underbrace{\frac{\epsilon_2 - \epsilon_1}{2h}}_{\text{Rounding error}}
 \end{aligned}$$

Rounding error

The rounding error has absolute value

$$\left| \frac{\epsilon_2 - \epsilon_1}{2h} \right| \leq \frac{2\epsilon_{\text{mach}}}{2h} = \frac{\epsilon_{\text{mach}}}{h},$$

where ϵ_{mach} represents machine epsilon. Therefore, the absolute value of the error of the machine approximation of $f'(x)$ is bounded above by

$$E(h) \equiv \frac{h^2}{6} f'''(c) + \frac{\epsilon_{\text{mach}}}{h}, \quad (5.11)$$

where $x - h < c < x + h$. Previously we had considered only the first term of the error, the mathematical error. The preceding table forces us to consider the loss of significance term as well.

Because there are two terms in the absolute error, how to find the best h that minimizes the solution error?

Rounding error

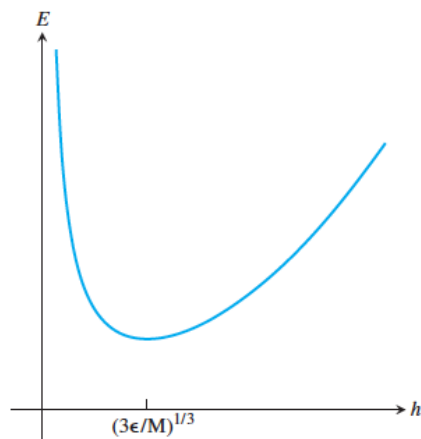
It is instructive to plot the function $E(h)$, shown in Figure 5.1. The minimum of $E(h)$ occurs at the solution of

$$0 = E'(h) = -\frac{\epsilon_{\text{mach}}}{h^2} + \frac{M}{3}h, \quad (5.12)$$

where we have approximated $|f'''(c)| \approx |f'''(x)|$ by M . Solving (5.12) yields

$$h = (3\epsilon_{\text{mach}}/M)^{1/3}$$

for the increment size h that gives smallest overall error, including the effects of computer rounding. In double precision, this is approximately $\epsilon_{\text{mach}}^{1/3} \approx 10^{-5}$, consistent with the table.



The main message is that the three-point centered difference formula will improve in accuracy as h is decreased until h becomes about the size of the cube root of machine epsilon. As h drops below this size, the error may begin increasing again.

$$f'(x) = \frac{f(x+h) - f(x)}{h} - \frac{h}{2} f''(c),$$

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} - \frac{h^2}{6} f'''(c),$$

Extrapolation

Assume that we are presented with an order n formula $F(h)$ for approximating a given quantity Q . The order means that

The formula depends on h instead of on x .

$$Q \approx F(h) + Kh^n,$$

where K is roughly constant over the range of h in which we are interested. A relevant example is

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} - \boxed{\frac{f'''(c_h)}{6}} h^2, \quad \approx f'''(x)/6 \quad (5.13)$$

In a case like this, a little bit of algebra can be used to leverage an order n formula into one of higher order. Because we know the order of the formula $F(h)$ is n , if we apply the formula again with $h/2$ instead of h , our error should be reduced from a constant times h^n to a constant times $(h/2)^n$, or reduced by a factor of 2^n . In other words, we expect

$$Q - F(h/2) \approx \frac{1}{2^n} (Q - F(h)). \quad (5.14)$$

We are relying on the assumption that K is roughly constant.

Extrapolation

Extrapolation for order n formula

$$Q \approx \frac{2^n F(h/2) - F(h)}{2^n - 1}. \quad (5.15)$$

This is the **extrapolation** formula for $F(h)$. Extrapolation, sometimes called **Richardson extrapolation**, typically gives a higher-order approximation of Q than $F(h)$. To understand why, assume that the n th-order formula $F_n(h)$ can be written

$$Q = F_n(h) + Kh^n + O(h^{n+1}).$$

Then cutting h in half yields $Q = F_n(h/2) + K\frac{h^n}{2^n} + O(h^{n+1})$, and the extrapolated version, which we call $F_{n+1}(h)$, will satisfy

$$\begin{aligned} F_{n+1}(h) &= \frac{2^n F_n(h/2) - F_n(h)}{2^n - 1} \\ &= \frac{2^n (Q - Kh^n/2^n - O(h^{n+1})) - (Q - Kh^n - O(h^{n+1}))}{2^n - 1} \\ &= Q + \frac{-Kh^n + Kh^n + O(h^{n+1})}{2^n - 1} = Q + O(h^{n+1}). \end{aligned}$$

Therefore, $F_{n+1}(h)$ is (at least) an order $n + 1$ formula for approximating the quantity Q . 85

Extrapolation


- Example

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} - \frac{f'''(c_h)}{6}h^2, \quad (5.13)$$

Apply extrapolation to formula (5.13).

We start with the second-order centered-difference formula $F_2(h)$ for the derivative $f'(x)$. The extrapolation formula (5.15) gives a new formula for $f'(x)$ as

$$\begin{aligned} \cancel{F_4(x)} &= \frac{2^2 F_2(h/2) - F_2(h)}{2^2 - 1} \\ F_4(h) &= \left[4 \frac{f(x+h/2) - f(x-h/2)}{h} - \frac{f(x+h) - f(x-h)}{2h} \right] / 3 \\ &= \frac{f(x-h) - 8f(x-h/2) + 8f(x+h/2) - f(x+h)}{6h}. \end{aligned} \quad (5.16)$$

This is a five-point centered-difference formula. The previous argument guarantees that this formula is of order at least three, but it turns out to have order four, because the order three error terms cancel out. In fact, since $F_4(h) = F_4(-h)$ by inspection, the error must be the same for h as for $-h$. Therefore, the error terms can be even powers of h only. 

Extrapolation


$$f''(x) = \frac{f(x-h) - 2f(x) + f(x+h)}{h^2} - \frac{h^2}{12} f^{(iv)}(c) \quad (5.8)$$

- Example

Apply extrapolation to the second derivative formula (5.8).

Again, the method is second order, so the extrapolation formula (5.15) is used with $n = 2$. The extrapolated formula is

$$\begin{aligned} \cancel{F_4(x)} &= \frac{2^2 F_2(h/2) - F_2(h)}{2^2 - 1} \\ F_4(h) &= \left[4 \frac{f(x+h/2) - 2f(x) + f(x-h/2)}{h^2/4} - \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} \right] / 3 \\ &= \frac{-f(x-h) + 16f(x-h/2) - 30f(x) + 16f(x+h/2) - f(x+h)}{3h^2}. \end{aligned}$$

The new method for approximating second derivatives is fourth order, for the same reason as the previous example. 

Newton-Cotes Formulas for Numerical Integration

- Given a function f defined on an interval $[a, b]$, two approaches for numerical integration:
- **Newton-Cotes approach**: Draw an **interpolating polynomial** through some of the points of $f(x)$ and evaluate the definite integral of the polynomial as the approximation of the integrals.
- **Gaussian Quadrature approach**: Find a **low-degree polynomial approximating** the function in the sense of least squares, and evaluate the definite integral of the polynomial as the approximation of the integrals.

Newton-Cotes Formulas for Numerical Integration

Trapezoid Rule

$$\int_{x_0}^{x_1} f(x) dx = \frac{h}{2}(y_0 + y_1) - \frac{h^3}{12}f''(c), \quad (5.21)$$

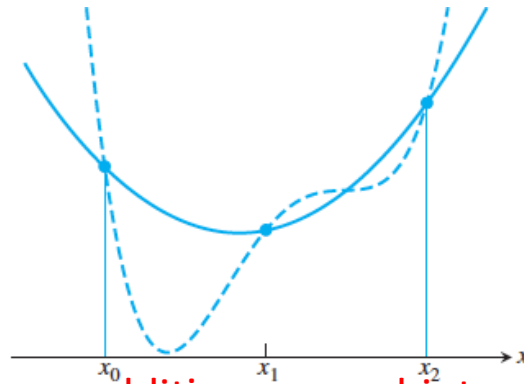
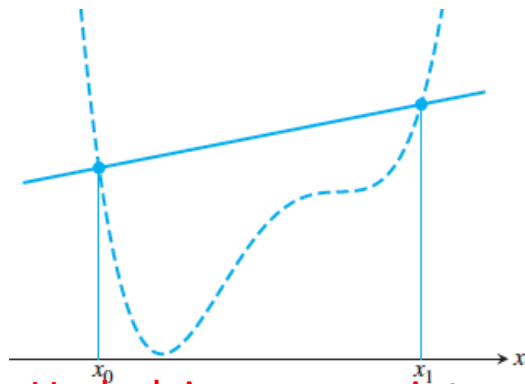
where $h = x_1 - x_0$ and c is between x_0 and x_1 .

Simpson's Rule

$$\int_{x_0}^{x_2} f(x) dx = \frac{h}{3}(y_0 + 4y_1 + y_2) - \frac{h^5}{90}f^{(iv)}(c), \quad (5.22)$$

where $h = x_2 - x_1 = x_1 - x_0$ and c is between x_0 and x_2 .

Composite Newton-Cotes formulas

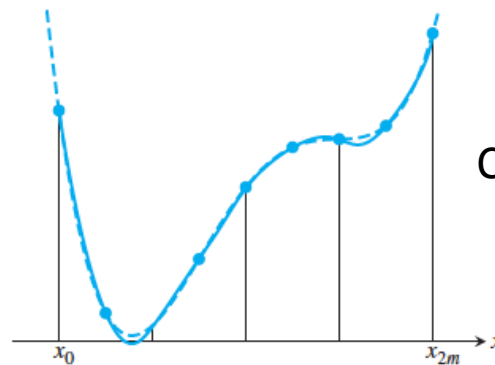
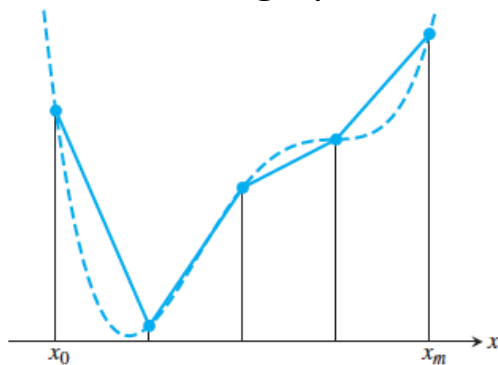


Single numerical integration

Underlying reason: integrals are additive over subintervals

Basic idea: evaluate an integral by dividing the interval up into several subintervals, applying the rule separately on each one, and then totaling up

VS



Composite numerical integration

Composite Newton-Cotes formulas

Composite Trapezoid Rule

$$\int_a^b f(x) dx = \frac{h}{2} \left(y_0 + y_m + 2 \sum_{i=1}^{m-1} y_i \right) - \frac{(b-a)h^2}{12} f''(c) \quad (5.24)$$

where $h = (b-a)/m$ and c is between a and b .

Composite Simpson's Rule

$$\int_a^b f(x) dx = \frac{h}{3} \left[y_0 + y_{2m} + 4 \sum_{i=1}^m y_{2i-1} + 2 \sum_{i=1}^{m-1} y_{2i} \right] - \frac{(b-a)h^4}{180} f^{(iv)}(c), \quad (5.25)$$

where c is between a and b .

Open Newton-Cotes Methods

- **Closed Newton-Cotes Methods:** Trapezoid Rule and Simpson's Rule Both use interval endpoints, i.e., require input values from the ends of the integration interval.
- **Open Newton-Cotes Methods:** which does not use values from the endpoints, especially useful for the integrands that have a **removable singularity** at an interval endpoint

Open Newton-Cotes Methods

Midpoint Rule

$$\int_{x_0}^{x_1} f(x) dx = hf(w) + \frac{h^3}{24} f''(c), \quad (5.26)$$

where $h = (x_1 - x_0)$, w is the midpoint $x_0 + h/2$, and c is between x_0 and x_1 .

- The Midpoint Rule is also useful for cutting the number of function evaluations needed.
- Compared with the Trapezoid Rule, the closed Newton–Cotes Method of **the same order**, it requires **one function evaluation** rather than two.
- The error term is **half the size** of the Trapezoid Rule error term.

Open Newton-Cotes Methods

Composite Midpoint Rule

$$\int_a^b f(x) dx = h \sum_{i=1}^m f(w_i) + \frac{(b-a)h^2}{24} f''(c), \quad (5.27)$$

where $h = (b-a)/m$ and c is between a and b . The w_i are the midpoints of the m equal subintervals of $[a, b]$.

Another useful open Newton-Cotes Rule is

$$\int_{x_0}^{x_4} f(x) dx = \frac{4h}{3} [2f(x_1) - f(x_2) + 2f(x_3)] + \frac{14h^5}{45} f^{(iv)}(c), \quad (5.28)$$

where $h = (x_4 - x_0)/4$, $x_1 = x_0 + h$, $x_2 = x_0 + 2h$, $x_3 = x_0 + 3h$, and where $x_0 < c < x_4$.