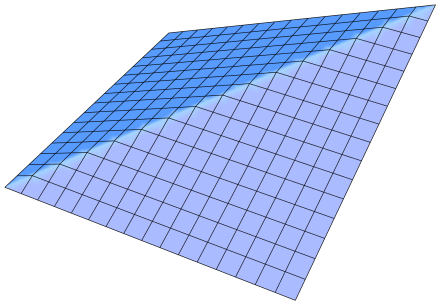


CS 371: Introduction to Computational Mathematics

Prof. Lori Case • Winter 2016 • University of Waterloo
TR 2:30–3:50PM in MC 4061

Transcribed, LaTeXed, and edited by Rob Zimmerman

Note: These are personal lecture notes, not official course materials. They may contain errors (which by default you should attribute to me, Rob Zimmerman, rather than the course instructor). The permanent home of these — and all of my other lecture notes — is https://rob-zimmerman.github.io/coursenotes.

Contents

Lecture 1 — Tuesday, January 05, 2016

1. Errors and Floating-Point Arithmetic

Numerical Error and Floating-Point Arithmetic

Computational mathematics uses algorithms to solve mathematical problems numerically. Important considerations include accuracy, efficiency, and robustness across a variety of inputs. These computations draw on floating-point arithmetic, polynomial interpolation, discrete Fourier methods, numerical linear algebra, numerical integration, and root finding.

Every computational problem raises three separate questions. First, the mathematical problem itself must be understood, because some problems are extremely sensitive to small perturbations in the data. Second, an algorithm must be chosen, and different algorithms may behave very differently on the same problem. Third, both the data and the algorithm must be represented numerically on a computer.

Some early examples show how small floating-point discrepancies can appear in harmless-looking calculations.

Example 1.1 Several MATLAB commands look completely harmless:

```
sqrt(10^15)*sqrt(10^15)
sqrt(10^15)*sqrt(10^15)-10^15
(1/5 + 1/5 + 1/5) * 5
(1/5 + 1/5 + 1/5) * 5 - 3
```

In double precision arithmetic, these produce

$$\sqrt{10^{15}} \cdot \sqrt{10^{15}} = 999999999999999.9,$$

$$\sqrt{10^{15}} \cdot \sqrt{10^{15}} - 10^{15} = -0.125,$$

$$\left(\frac{1}{5}+\frac{1}{5}+\frac{1}{5}\right)\cdot 5 = 3.0000000000000004,$$

and

$$\left(\frac{1}{5}+\frac{1}{5}+\frac{1}{5}\right)\cdot 5 - 3 = 4.440892098500626 \times 10^{-16}.$$

The errors are tiny, but they already show that computer arithmetic does not exactly obey the same rules as exact arithmetic.

These examples are only meant to set the stage. The more interesting question is what happens when such small discrepancies are repeatedly fed into a longer algorithm.

For a fixed parameter $\alpha > 0$ and an integer $n \geq 0$, define

$$I_n = \int_0^1 \frac{x^n}{x+\alpha}\,\mathrm{d}x.$$

At first glance this integral does not suggest an easy algorithm, but a simple algebraic manipulation produces a recurrence.

Proposition 1.2 For every integer $n \geq 1$,

$$I_n = \frac{1}{n} - \alpha I_{n-1}. \tag{1.1}$$

Moreover,

$$I_0 = \ln\left(\frac{1+\alpha}{\alpha}\right). \tag{1.2}$$

Proof. For $n \geq 1$,

$$\frac{x^n}{x+\alpha} = \frac{x^{n-1}(x+\alpha) - \alpha x^{n-1}}{x+\alpha} = x^{n-1} - \alpha\frac{x^{n-1}}{x+\alpha}.$$

Integrating from 0 to 1 gives

$$I_n = \int_0^1 x^{n-1}\,\mathrm{d}x - \alpha\int_0^1 \frac{x^{n-1}}{x+\alpha}\,\mathrm{d}x = \frac{1}{n} - \alpha I_{n-1},$$

which proves (1.1). For $n = 0$,

$$I_0 = \int_0^1 \frac{1}{x+\alpha}\,\mathrm{d}x = \left[\ln(x+\alpha)\right]_{x=0}^{x=1} = \ln(1+\alpha) - \ln(\alpha) = \ln\left(\frac{1+\alpha}{\alpha}\right).$$

□

Equations (1.1) and (1.2) appear to give a perfectly reasonable algorithm: compute I_0 exactly from the logarithm, and then generate $I_1, I_2, \ldots$ recursively.

Proposition 1.3 Suppose the computed values satisfy

$$I_n^{(\text{comp})} = \frac{1}{n} - \alpha I_{n-1}^{(\text{comp})},$$

while the exact values satisfy

$$I_n^{(\text{exact})} = \frac{1}{n} - \alpha I_{n-1}^{(\text{exact})}.$$

If

$$e_n = I_n^{(\text{comp})} - I_n^{(\text{exact})},$$

then

$$e_n = -\alpha e_{n-1}, \tag{1.3}$$

and hence

$$e_n = (-\alpha)^n e_0. \tag{1.4}$$

In particular,

$$|e_n| = \alpha^n |e_0|.$$

Proof. Subtracting the two recurrences gives

$$\begin{aligned} e_n &= \left(\frac{1}{n} - \alpha I_{n-1}^{(\text{comp})}\right) - \left(\frac{1}{n} - \alpha I_{n-1}^{(\text{exact})}\right) \\ &= -\alpha \left(I_{n-1}^{(\text{comp})} - I_{n-1}^{(\text{exact})}\right) \\ &= -\alpha e_{n-1}, \end{aligned}$$

which is (1.3). Iterating the relation yields (1.4). □

The sign in (1.4) alternates, but the magnitude is what matters numerically. If $0 < \alpha < 1$, then the initial error is damped as the iteration continues. If $\alpha > 1$, then the initial error is amplified exponentially.

Example 1.4 Consider the following MATLAB code:

```
% Floating-Point Blues
% Integral example
% Try alpha values of 0.5 and 2.
alpha = 0.5;
N = 100;
I = log((1+alpha) / alpha);
for n = 1:N
    I = 1/n - alpha * I;
end
disp(['Answer: ' num2str(I)]);
```

When $n = 100$, the reported values are approximately

$$I_{100} \approx 0.0066444 \quad \text{for } \alpha = 0.5,$$

and

$$I_{100} \approx 6.058010443603891 \times 10^{12} \quad \text{for } \alpha = 2.$$

The first answer is plausible, while the second is obviously absurd.

Proposition 1.5 If $\alpha > 1$, then

$$0 < I_{100} < \frac{1}{101}.$$

Proof. For $x \in [0, 1]$ and $\alpha > 1$, we have $x + \alpha > 1$, so

$$0 < \frac{x^{100}}{x + \alpha} < x^{100}.$$

Integrating over $[0, 1]$ gives

$$0 < I_{100} = \int_0^1 \frac{x^{100}}{x + \alpha}\, \mathrm{d}x < \int_0^1 x^{100}\, \mathrm{d}x = \frac{1}{101}.$$

□

This example is important because the math is correct, but the algorithm is only numerically reliable for some values of α. The recurrence is mathematically sound for every admissible α, yet it is computationally stable only when the factor α does not amplify the existing error.

Another example concerns the power series

$$e^x = \sum_{i=0}^{\infty} \frac{x^i}{i!}. \tag{1.5}$$

If we want to approximate $e^{-5.5}$, the exact value is

$$e^{-5.5} \approx 0.004086771438\ldots,$$

which rounds to 0.0040868 when written with five significant digits.

If we apply (1.5) directly with $x = -5.5$, then we obtain

$$e^{-5.5} = \sum_{i=0}^{\infty} \frac{(-5.5)^i}{i!}.$$

The series is infinite, so a computer must truncate it. The twenty-sixth term is already on the order of 10^{-8}, so keeping the first twenty-five terms seems reasonable if we only want about five significant digits.

Example 1.6 Using five-digit arithmetic, the partial sums evolve as follows:

i	ith term in the series	corresponding partial sum
0	1.0000000000000	1.0000000000000
1	−5.5000000000000	−4.5000000000000
2	15.1250000000000	10.6250000000000
3	−27.7300000000000	−17.1050000000000
⋮	⋮	⋮
20	0.0002637100000	0.0058115000000
21	−0.0000690670000	0.0057424000000
22	0.0000172670000	0.0057597000000
23	−0.0000041289000	0.0057556000000
24	0.0000009462300	0.0057565000000
25	−0.0000002081700	0.0057563000000
26	0.0000000440350	0.0057563000000

At this point the computed sum no longer changes, so the algorithm stops with the value 0.0057563.

That answer has the wrong size; it differs from the true value by about 41%. The issue is not the truncation of the infinite series, because exact summation of the same truncated series would be much more accurate. The real problem is repeated subtraction of numbers with similar magnitudes. This is a classic instance of **catastrophic cancellation**.

Example 1.7 Instead of summing alternating terms for e^{-x} directly, we may use the identity

$$e^{-x} = \frac{1}{e^x} = \left(\sum_{i=0}^{\infty} \frac{x^i}{i!} \right)^{-1}.$$

For $x = 5.5$, the denominator becomes

$$1 + 5.5 + \frac{5.5^2}{2!} + \cdots + \frac{5.5^{24}}{24!}.$$

Using the same five-digit arithmetic and the same twenty-five terms, this gives the approximation

$$e^{-5.5} \approx 0.0040865,$$

which agrees with the true value to four significant digits and has relative error about 6.6×10^{-5}.

The two formulas are mathematically equivalent, but one is stable for this data and the other is not. A correct formula is not automatically a good computational algorithm.

Example 1.8 Consider the binary numbers

$$0.1111_2, \qquad 0.0111_2, \qquad 0.0011_2, \qquad \text{and} \qquad 0.0001_2,$$

and suppose that we keep only four significant binary digits after each addition. If we add from largest to smallest, then

$$0.1111_2 + 0.0111_2 = 1.0110_2 = 0.1011_2 \times 2,$$

so the first sum is exact. Adding the next term gives

$$1.0110_2 + 0.0011_2 = 1.1001_2 = 0.11001_2 \times 2,$$

which must be chopped to $0.1100_2 \times 2 = 1.1000_2$. Adding 0.0001_2 now leaves the result unchanged after chopping, so the final answer is 1.1000_2.

If instead we add from smallest to largest, then

$$0.0001_2 + 0.0011_2 = 0.0100_2 \qquad \text{and} \qquad 0.0100_2 + 0.0111_2 = 0.1011_2,$$

and finally

$$0.1011_2 + 0.1111_2 = 1.1010_2 = 0.11010_2 \times 2.$$

Chopping the normalized result to four significant digits gives $0.1101_2 \times 2 = 1.1010_2$, which is the exact sum. Thus the two orders give different outputs: adding from largest to smallest gives 1.1000_2, while adding from smallest to largest gives 1.1010_2.

Note: In floating-point arithmetic, we should not assume that

$$(a+b)+c = a+(b+c) \qquad \text{and} \qquad a+e=a \qquad \text{implies} \qquad e=0,$$

or

$$\frac{a+b}{c} = \frac{a}{c} + \frac{b}{c}.$$

These identities are valid in exact arithmetic, but finite-precision arithmetic can violate them through rounding, chopping, overflow, or underflow.

Ignoring these effects can have serious consequences. Two well-known examples are the Patriot missile failure and the Ariane 5 explosion. In the first case, a small timing error accumulated over many hours to produce a large position error, which caused the missile to miss its target. In the second case, a software module that was perfectly fine for the Ariane 4 was reused in the Ariane 5 without realizing that the new rocket's faster speed would cause overflow in that module, which led to an explosion shortly after launch.

Representations of Numbers

Integers have infinite range in mathematics, but a computer can store only finitely many digits. Once a base is fixed, there is a smallest representable integer and a largest representable integer. Integer arithmetic is exact as long as every intermediate result remains inside that range. Once a computation falls outside the range, overflow occurs.

Real numbers are even more problematic, because they have both infinite range and infinite

precision. Between any two distinct real numbers there are infinitely many others, so a computer must approximate most real values. This is why computer arithmetic is best viewed as approximate arithmetic rather than exact arithmetic.

Definition 1.9 Let z be an exact value and let $\tilde{z}$ be an approximation to z. The **absolute error** is

$$\Delta z = z - \tilde{z}.$$

If $z \neq 0$, the **relative error** is

$$\delta z = \frac{z - \tilde{z}}{z} = \frac{\Delta z}{z}.$$

Definition 1.10 A **fixed-point number system** is determined by a base b, a number I of digits for the integer part, and a number F of digits for the fractional part. Its numbers have the form

$$\pm i_1 i_2 \cdots i_I . f_1 f_2 \cdots f_F,$$

where each digit lies in $\{0, 1, \ldots, b-1\}$.

Proposition 1.11 If leading zeros are allowed and zero is stored only once, then a fixed-point system with parameters (b, I, F) contains

$$2b^{I+F} - 1$$

representable numbers.

Proof. There are b^I choices for the integer digits and b^F choices for the fractional digits, so there are b^{I+F} nonnegative digit patterns. Every nonzero pattern may be given either sign, while zero has only one representation. Therefore the total number of representable values is

$$1 + 2\left(b^{I+F} - 1\right) = 2b^{I+F} - 1.$$

□

Definition 1.12 A **floating-point number system** $\mathrm{FL}(b, m, e)$ is determined by a base b, a mantissa length m, and an exponent length e. A nonzero normalized floating-point number has the form

$$\pm 0.x_1 x_2 \cdots x_m \times b^{\pm y_1 y_2 \cdots y_e},$$

where $x_1 \in \{1, \ldots, b-1\}$ and each later mantissa digit belongs to $\{0, 1, \ldots, b-1\}$. The value 0 is also included.

Example 1.13 Consider the decimal floating-point system $\mathrm{FL}(10, 4, 2)$, which has base 10, four mantissa digits, and two exponent digits. The number 12.3456789 is not representable in this system, so we must approximate it by chopping or rounding to four mantissa digits. First, normalize the number:

$$12.3456789 = 0.123456789 \times 10^2.$$

If we chop to four mantissa digits, we obtain

$$\mathrm{fl}_{\mathrm{chop}}(12.3456789) = 0.1234 \times 10^2 = 12.34.$$

The absolute error is

$$12.3456789 - 12.34 = 0.0056789 = 0.000056789 \times 10^2.$$

If we round to four mantissa digits, we obtain

$$\mathrm{fl}_{\mathrm{round}}(12.3456789) = 0.1235 \times 10^2 = 12.35.$$

The absolute error is

$$12.3456789 - 12.35 = -0.0043211 = -0.000043211 \times 10^2.$$

Rounding is therefore more accurate in this example, although chopping is often easier to analyze.

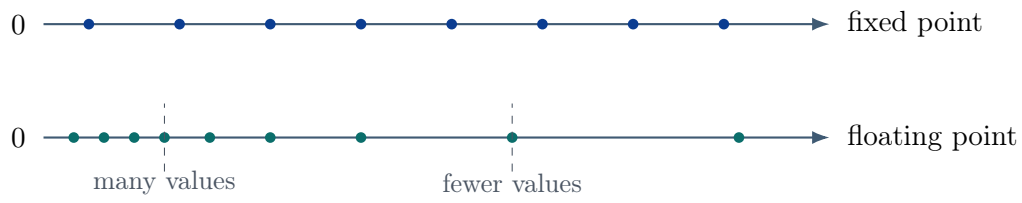


Figure 1

Figure 1 emphasizes the main tradeoff. Fixed-point numbers are evenly spaced, but they cannot represent extremely large or extremely small values without using many digits. Floating-point

numbers cover a much wider dynamic range, but their spacing is nonuniform, so nearby large numbers can be much farther apart than nearby small ones.

Machine Epsilon and Relative Error

Consider a general floating-point system $\mathrm{FL}(b, m, e)$. We need a uniform way to bound the relative error introduced when a real number is stored in this system.

Definition 1.14 In these notes, the **machine epsilon** $\varepsilon_{\text{mach}}$, also called the **unit roundoff**, is the smallest uniform bound such that

$$\mathrm{fl}(x) = x(1+\eta) \quad \text{with} \quad |\eta| \le \varepsilon_{\text{mach}}$$

for every nonzero x in the normal representable range.

Proposition 1.15 In a chopped floating-point system,

$$\varepsilon_{\text{mach}} = b^{1-m}.$$

If rounding is used instead, then

$$\varepsilon_{\text{mach}} = \frac{1}{2} b^{1-m}.$$

Proof. Write a normalized nonzero number as

$$x = \pm b^p (0.d_1 d_2 d_3 \ldots) \qquad \text{and} \qquad d_1 \neq 0.$$

Chopping after m digits discards a tail of magnitude less than b^{p-m}, while $|x| \ge b^{p-1}$. The relative error is therefore less than b^{1-m}, and this bound is sharp. Rounding to the nearest floating-point number reduces the largest possible error by a factor of two, giving $\frac{1}{2} b^{1-m}$. □

Proposition 1.16 Whenever x lies in the normal representable range and is stored as $\mathrm{fl}(x)$, we may write

$$\mathrm{fl}(x) = x(1+\eta),$$

where

$$|\eta| \le \varepsilon_{\text{mach}}.$$

Proof. Assume first that chopping is used, and write the normalized form of x as

$$x = \pm b^p \left(0.d_1 d_2 d_3 \ldots\right),$$

where $d_1 \in \{1, \ldots, b-1\}$. The chopped floating-point representation is

$$\mathrm{fl}(x) = \pm b^p \left(0.d_1 d_2 \ldots d_m\right).$$

If we write the discarded tail as

$$r = 0.\underbrace{00\ldots0}_{m \text{ digits}} d_{m+1} d_{m+2} \ldots,$$

then

$$x - \mathrm{fl}(x) = \pm b^p r \qquad \text{and} \qquad 0 \le r < b^{-m}.$$

Therefore

$$|x - \mathrm{fl}(x)| < b^{p-m}.$$

On the other hand, since the leading digit d_1 is nonzero, we have

$$|x| \ge b^p \cdot b^{-1} = b^{p-1}.$$

Dividing these inequalities gives

$$\left|\frac{x - \mathrm{fl}(x)}{x}\right| < \frac{b^{p-m}}{b^{p-1}} = b^{1-m} = \varepsilon_{\mathrm{mach}}.$$

Hence, if we define

$$\eta = \frac{\mathrm{fl}(x) - x}{x},$$

then $\mathrm{fl}(x) = x(1+\eta)$ and $|\eta| \le \varepsilon_{\mathrm{mach}}$. If rounding is used instead, the stored number is the nearest floating-point number to x, so the absolute error is at most half the spacing between consecutive chopped numbers at that scale. Thus

$$\left|\frac{x - \mathrm{fl}(x)}{x}\right| \le \frac{1}{2} b^{1-m} \le \varepsilon_{\mathrm{mach}},$$

because in the rounding model $\varepsilon_{\mathrm{mach}} = 1/2b^{1-m}$. This gives the same representation in the rounded case. □

This bound is the standard first-order model for floating-point arithmetic. In practice, we often summarize the consequence by saying that binary single precision stores about seven decimal digits

of accuracy, while binary double precision stores about sixteen.

Floating-Point Operations and Conditioning

If a and b are floating-point numbers, we denote their computed sum by $a \oplus b$. The computation consists of representing a and b, performing the exact arithmetic on those represented values, and then rounding or chopping the result back into the floating-point system. Consequently, each arithmetic operation introduces a fresh error factor of the form $(1+\eta)$ with $|\eta| \le \varepsilon_{\text{mach}}$, provided no overflow or underflow occurs. For a single operation the effect may be tiny, but repeated operations can accumulate or amplify those errors.

Catastrophic cancellation occurs when we subtract two nearly equal numbers and lose many significant digits in the process. **Overflow** occurs when a computed value is too large to be represented in the number system. **Underflow** occurs when a computed value is so small that it cannot be distinguished from zero.

The 1994 Intel Pentium division bug is a reminder that finite arithmetic affects real systems. A lookup table used for floating-point division was missing five entries out of 1066, and Intel spent nearly half a billion dollars to recall and replace the affected chips.

Not every bad numerical answer should be blamed on the algorithm. Sometimes the mathematical problem itself is sensitive to perturbations.

Example 1.17 Consider

$$y = \frac{x}{1-x}.$$

If $x = 0.93$, then

$$y = \frac{0.93}{0.07} = \frac{93}{7} = 13\frac{2}{7}.$$

Now perturb the input to $\tilde{x} = 0.94$, so that $\Delta x = 0.01$. The relative error in the input is

$$\frac{\Delta x}{x} = \frac{0.01}{0.93} = \frac{1}{93} \approx 0.01075.$$

The corresponding output is

$$\tilde{y} = \frac{0.94}{0.06} = \frac{94}{6} = \frac{47}{3} = 15\frac{2}{3}.$$

The relative error in the output is

$$\frac{\tilde{y}-y}{y} = \frac{\frac{47}{3}-\frac{93}{7}}{\frac{93}{7}} = \frac{50}{279} \approx 0.1792.$$

Thus a roughly 1% relative perturbation in the input produces a roughly 18% relative perturbation in the output.

Definition 1.18 Suppose a scalar problem maps an input x to an output $z = f_P(x)$, where f_P is differentiable at x. The problem is **well-conditioned** at x if small input perturbations produce comparably small output perturbations, and it is **ill-conditioned** if they can be strongly amplified.

The local **absolute condition number** is

$$\kappa_A(x) = \left|f_P'(x)\right|,$$

and, when $x \neq 0$ and $f_P(x) \neq 0$, the local **relative condition number** is

$$\kappa_R(x) = \left|\frac{x f_P'(x)}{f_P(x)}\right|.$$

For a particular finite perturbation, the ratios $|\Delta z|/|\Delta x|$ and $|\Delta z/z|/|\Delta x/x|$ are the corresponding observed amplification factors; they approach the local condition numbers as the perturbation tends to zero.

The previous example shows that poor numerical behavior may come from the problem itself rather than from any particular implementation. Vector norms provide a way to measure perturbations in higher-dimensional settings.

Definition 1.19 For $\boldsymbol{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$, the **2-norm** and **1-norm** are

$$\|\boldsymbol{x}\|_2 = \left(\sum_{i=1}^{n} x_i^2\right)^{1/2} \qquad \text{and} \qquad \|\boldsymbol{x}\|_1 = \sum_{i=1}^{n} |x_i|.$$

These norms satisfy the Cauchy–Schwarz inequality

$$|\boldsymbol{x} \cdot \boldsymbol{y}| \leq \|\boldsymbol{x}\|_2 \|\boldsymbol{y}\|_2.$$

Algorithmic Stability

Even for a well-conditioned problem, one algorithm may be reliable while another is not.

Example 1.20 Consider

$$x^2 + 62.1x + 1 = 0.$$

The exact roots are

$$x_{1,2} = \frac{-62.1 \pm \sqrt{62.1^2 - 4}}{2}.$$

Since

$$62.1^2 - 4 = 3852.41 - 4 = 3848.41 \qquad \text{and} \qquad \sqrt{3848.41} \approx 62.0678,$$

the roots are approximately

$$x_1 \approx \frac{-62.1 + 62.0678}{2} \approx -0.0161 \qquad \text{and} \qquad x_2 \approx \frac{-62.1 - 62.0678}{2} \approx -62.08.$$

The difficulty is that the formula for x_1 subtracts two nearly equal numbers, so catastrophic cancellation destroys accuracy. To avoid this, we rewrite one root as

$$x_1 = \frac{-2}{62.1 + \sqrt{62.1^2 - 4}},$$

which produces the small root without subtracting nearly equal quantities. The large root can then be computed from the product relation $x_1 x_2 = 1$, namely

$$x_2 = \frac{1}{x_1}.$$

If we instead use the reciprocal-style formula for the wrong root, then the same cancellation problem reappears in a different place.

Assume that a problem P is well-conditioned. An algorithm A for solving P is **unstable** if either small perturbations in the data produce much larger changes in the computed answer, or small roundoff and representation errors at one stage are greatly amplified as the computation proceeds. If this does not happen, the algorithm is called **stable**.

The integral recurrence is stable when $\alpha < 1$ and unstable when $\alpha > 1$. For the exponential example, direct summation of the alternating series is unstable for negative x, while the reciprocal formula is stable there. If $x > 0$, their roles reverse. Stability therefore depends not only on the formula but also on the data being used.

Errors can enter through the mathematical model, the representation of numbers, roundoff in arithmetic, truncation of infinite processes, ill-conditioning of the underlying problem, instability of the algorithm, and ordinary human mistakes such as measurement or coding errors. Our central goal is to recognize these effects, quantify them, and design algorithms that control them.

Lecture 2 — Thursday, January 14, 2016

2. Interpolation and Curves

Interpolation

Interpolation begins with finitely many known data values and asks for a function that matches them exactly. If we are given points

$$(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n),$$

then the interpolation problem is to find a function f such that

$$f(x_i) = y_i, \quad i = 0, 1, \ldots, n.$$

Once such a function is available, we may use it to estimate values between the given data points, and sometimes to estimate derivatives as well.

Example 2.1 A digital image is a rectangular array of pixels, and each pixel stores a numerical colour value. If an image is enlarged, rotated, or otherwise transformed, the pixels of the new image generally do not line up exactly with the old ones. Interpolation is then used to estimate pixel values between the original sample points.

Example 2.2 Consider the population of Canada, measured in millions, at five-year intervals:

Year	1971	1976	1981	1986	1991	1996	2001	2006	2011
Population	21.57	22.99	24.34	25.31	27.30	28.85	30.01	31.61	33.48

An interpolating function can be used to estimate the population in years between the census dates.

Polynomial interpolants are especially useful because they are easy to evaluate, differentiate, and integrate. For $n+1$ data points, the most direct approach is to seek a polynomial of degree at most n,

$$p_n(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n.$$

Vandermonde Formulation

Imposing the interpolation conditions $p_n(x_i) = y_i$ produces the linear system

$$\begin{pmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{pmatrix}.$$

The coefficient matrix is called the **Vandermonde matrix**.

Proposition 2.3 Let $\mathbf{V}$ be the Vandermonde matrix for the nodes $x_0, \ldots, x_n$:

$$\mathbf{V} = \left(x_i^j\right)_{0 \le i,j \le n},$$

where the first column corresponds to $j = 0$. Then

$$\det(\mathbf{V}) = \prod_{0 \le i < j \le n} (x_j - x_i).$$

In particular, $\det(\mathbf{V}) \neq 0$ if and only if the interpolation nodes $x_0, \ldots, x_n$ are distinct.

Proof. Viewed as a polynomial in the variables $x_0, \ldots, x_n$, the determinant changes sign whenever two variables are interchanged. Therefore $\det(\mathbf{V})$ vanishes whenever $x_i = x_j$, so each factor $(x_j - x_i)$ divides $\det(\mathbf{V})$. The product

$$\prod_{0 \le i < j \le n} (x_j - x_i)$$

has total degree

$$1 + 2 + \cdots + n = \frac{n(n+1)}{2},$$

which is also the total degree of the determinant. Hence

$$\det(\mathbf{V}) = C \prod_{0 \le i < j \le n} (x_j - x_i)$$

for some constant C. To determine C, expand the determinant and look at the coefficient of

$$x_1 x_2^2 \cdots x_n^n.$$

It appears with coefficient 1 on both sides, so $C = 1$. □

Corollary 2.4 If $x_0, \ldots, x_n$ are distinct, then there is a unique polynomial p_n of degree at most n such that

$$p_n(x_i) = y_i, \quad i = 0, 1, \ldots, n.$$

Proof. Distinct nodes imply $\det(\mathbf{V}) \neq 0$, so the Vandermonde system has a unique solution for the coefficients $a_0, \ldots, a_n$. □

Example 2.5 Extracting four points from Example 2.2 and taking x to be the number of years since 1995, the data

$$(1, 28.85), \qquad (6, 30.01), \qquad (11, 31.61), \qquad \text{and} \qquad (16, 33.48)$$

lead to a cubic interpolating polynomial. The Vandermonde system is

$$\begin{pmatrix} 1 & 1 & 1^2 & 1^3 \\ 1 & 6 & 6^2 & 6^3 \\ 1 & 11 & 11^2 & 11^3 \\ 1 & 16 & 16^2 & 16^3 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} 28.85 \\ 30.01 \\ 31.61 \\ 33.48 \end{pmatrix}.$$

Solving this system gives

$$p_3(x) = -0.000226666\ldots x^3 + 0.01288x^2 + 0.151586666\ldots x + 28.68576,$$

which we can write in rounded form as

$$p_3(x) \approx -0.000227x^3 + 0.01288x^2 + 0.1516x + 28.6858.$$

Thus, for example, the estimated population (in millions) in 2000 (corresponding to $x = 5$) is

$$p_3(5) \approx 29.74.$$

The cubic from Example 2.5 passes through each of the selected data points, as shown in Figure 2.

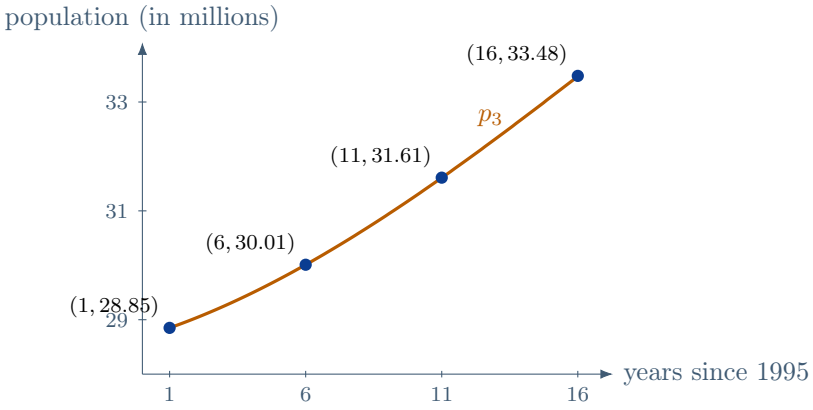


FIGURE 2

Although the Vandermonde formulation is conceptually simple, it is not always numerically attractive. Solving a dense linear system is expensive, the matrix becomes ill-conditioned when nodes are close together, and the situation generally worsens as n grows.

Lagrange Interpolation

There is a more direct way to write the interpolating polynomial. To motivate the idea, suppose we have points (x_0, y_0) and (x_1, y_1), and we want a linear interpolant $p(x) = a_0 + a_1 x$. With

$$p(x_0) = a_0 + a_1 x_0 = y_0 \qquad \text{and} \qquad p(x_1) = a_0 + a_1 x_1 = y_1,$$

we solve the system to find

$$a_0 = \frac{y_0 x_1 - y_1 x_0}{x_1 - x_0} \qquad \text{and} \qquad a_1 = \frac{y_1 - y_0}{x_1 - x_0}.$$

Substituting back gives

$$p(x) = \frac{y_0 x_1 - y_1 x_0}{x_1 - x_0} + \frac{y_1 - y_0}{x_1 - x_0} x = y_0 \frac{x - x_1}{x_0 - x_1} + y_1 \frac{x - x_0}{x_1 - x_0}.$$

The two terms are linear polynomials that vanish at one of the nodes and equal 1 at the other. This idea generalizes to higher degree.

Definition 2.6 For distinct nodes $x_0, \ldots, x_n$, define the **Lagrange basis functions** $\ell_0, \ldots, \ell_n$ by

$$\ell_k(x) = \prod_{\substack{0 \le j \le n \\ j \ne k}} \frac{x - x_j}{x_k - x_j}, \quad k = 0, 1, \ldots, n.$$

Proposition 2.7 For all $j, k \in \{0, \ldots, n\}$,

$$\ell_k(x_j) = \begin{cases} 1, & j = k, \\ 0, & j \ne k. \end{cases}$$

Proof. If $j = k$, then every factor in the product equals 1, so $\ell_k(x_k) = 1$. If $j \ne k$, then the factor with index j is

$$\frac{x_j - x_j}{x_k - x_j} = 0,$$

so $\ell_k(x_j) = 0$. □

Theorem 2.8 The unique polynomial of degree at most n satisfying

$$p_n(x_i) = y_i, \quad i = 0, 1, \ldots, n,$$

is the **Lagrange interpolating polynomial**

$$p_n(x) = \sum_{k=0}^{n} y_k \ell_k(x).$$

Proof. Each ℓ_k has degree at most n, so p_n also has degree at most n. For any node x_j,

$$p_n(x_j) = \sum_{k=0}^{n} y_k \ell_k(x_j) = \sum_{k=0}^{n} y_k \delta_{jk} = y_j.$$

Thus p_n interpolates the data. Uniqueness follows from Corollary 2.4. □

For the nodes $0, 1, 2$, the three Lagrange basis functions are shown in Figure 3. Each basis function is equal to 1 at its own node and 0 at the other two nodes.

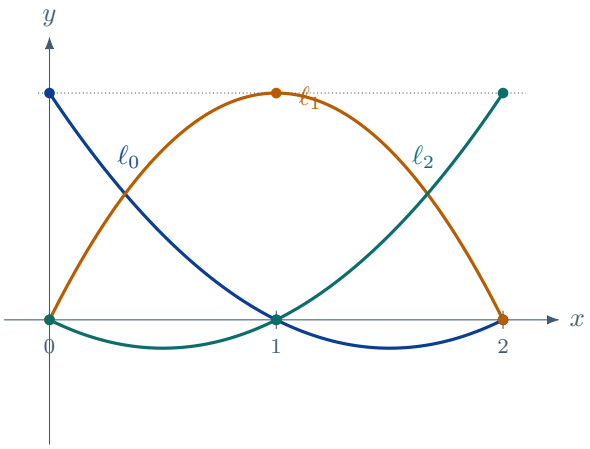


FIGURE 3

Remark 2.9 The vector space

$$P_n = \{q(x) : q \text{ is a polynomial of degree at most } n\}$$

has the usual monomial basis $\{1, x, x^2, \ldots, x^n\}$, but the Lagrange polynomials $\{\ell_0, \ell_1, \ldots, \ell_n\}$ form another basis. The interpolating polynomial is the same object regardless of which basis is used to construct it.

Example 2.10 Using the same four points in Example 2.5, the Lagrange basis functions are

$$\begin{aligned}
\ell_0(x) &= \frac{(x-6)(x-11)(x-16)}{(1-6)(1-11)(1-16)} \\
&= -\frac{(x-6)(x-11)(x-16)}{750} = -\frac{x^3 - 33x^2 + 262x - 1056}{750}, \\
\ell_1(x) &= \frac{(x-1)(x-11)(x-16)}{(6-1)(6-11)(6-16)} \\
&= \frac{(x-1)(x-11)(x-16)}{250} = \frac{x^3 - 28x^2 + 227x - 176}{250},
\end{aligned}$$

$$\ell_2(x) = \frac{(x-1)(x-6)(x-16)}{(11-1)(11-6)(11-16)}$$
$$= -\frac{(x-1)(x-6)(x-16)}{250} = -\frac{x^3 - 23x^2 + 182x - 96}{250},$$

and

$$\ell_3(x) = \frac{(x-1)(x-6)(x-11)}{(16-1)(16-6)(16-11)}$$
$$= \frac{(x-1)(x-6)(x-11)}{750} = \frac{x^3 - 18x^2 + 137x - 66}{750}.$$

Hence

$$p_3(x) = 28.85\,\ell_0(x) + 30.01\,\ell_1(x) + 31.61\,\ell_2(x) + 33.48\,\ell_3(x),$$

which works out to the same cubic found from the Vandermonde system.

The accuracy of the Lagrange interpolating polynomial is controlled by the interpolation error.

Proposition 2.11 (Generalized Rolle's theorem) Let $n \geq 1$, let $f \in C^n([a,b])$, and suppose that

$$f(x_0) = f(x_1) = \cdots = f(x_n) = 0$$

for distinct points

$$a \leq x_0 < x_1 < \cdots < x_n \leq b.$$

Then there exists $c \in (a,b)$ such that

$$f^{(n)}(c) = 0.$$

Proof. By the ordinary Rolle's theorem, for each interval (x_{k-1}, x_k) there exists a point $c_k^{(1)}$ such that

$$f'(c_k^{(1)}) = 0, \quad k = 1, \ldots, n.$$

These give n distinct zeros of f'. Applying Rolle's theorem to f' on consecutive intervals between these zeros produces $n-1$ distinct zeros of f''. Repeating the argument n times yields a point c where

$$f^{(n)}(c) = 0.$$

□

Theorem 2.12 Let $x_0, \ldots, x_n$ be distinct points in $[a, b]$, and let $f \in C^{n+1}([a, b])$. If p_n is the Lagrange interpolating polynomial for f at these nodes, then for every $x \in [a, b]$ there exists $\xi(x) \in [a, b]$ such that

$$f(x) - p_n(x) = \frac{f^{(n+1)}(\xi(x))}{(n+1)!} \prod_{k=0}^{n} (x - x_k).$$

Proof. Fix $x \in [a, b]$. If $x = x_k$ for some k, then both sides are zero and there is nothing to prove. Assume therefore that x is different from every node, and set

$$\omega(t) = \prod_{k=0}^{n} (t - x_k).$$

Define

$$g(t) = f(t) - p_n(t) - \frac{f(x) - p_n(x)}{\omega(x)} \omega(t).$$

For each interpolation node x_k we have $\omega(x_k) = 0$ and $f(x_k) - p_n(x_k) = 0$, so

$$g(x_k) = 0.$$

Also,

$$g(x) = f(x) - p_n(x) - \frac{f(x) - p_n(x)}{\omega(x)} \omega(x) = 0.$$

Thus g has $n + 2$ distinct zeros, namely $x_0, \ldots, x_n, x$. By the generalized Rolle's theorem, there exists $\xi(x) \in (a, b)$ such that

$$g^{(n+1)}(\xi(x)) = 0.$$

Since p_n has degree at most n, its $(n + 1)$st derivative vanishes, and since ω is monic of degree $n + 1$, we have

$$\omega^{(n+1)}(t) = (n + 1)!.$$

Therefore

$$0 = g^{(n+1)}(\xi(x)) = f^{(n+1)}(\xi(x)) - \frac{f(x) - p_n(x)}{\omega(x)} (n + 1)!.$$

Solving for $f(x) - p_n(x)$ gives

$$f(x) - p_n(x) = \frac{f^{(n+1)}(\xi(x))}{(n+1)!} \omega(x) = \frac{f^{(n+1)}(\xi(x))}{(n+1)!} \prod_{k=0}^{n} (x - x_k).$$

□

Remark 2.13 This formula explains why interpolation by a single high-degree polynomial can become problematic. Even if $f^{(n+1)}$ is moderate, the factor

$$\prod_{k=0}^{n}(x - x_k)$$

may become large or highly oscillatory, especially when many nodes are used.

Hermite and Piecewise Interpolation

Sometimes function values and first derivatives are both available. These are valuable! If the data are

$$x_0, \ldots, x_n, \qquad y_0, \ldots, y_n, \qquad \text{and} \qquad y'_0, \ldots, y'_n,$$

then there are $2(n+1)$ interpolation conditions, so it is natural to look for a polynomial of degree at most $2n+1$. If we try to solve for the coefficients of such a polynomial in the monomial basis, we get a Vandermonde-like system that is even more ill-conditioned than before:

$$\begin{pmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^{2n+1} \\ 0 & 1 & 2x_0 & \cdots & (2n+1)x_0^{2n} \\ 1 & x_1 & x_1^2 & \cdots & x_1^{2n+1} \\ 0 & 1 & 2x_1 & \cdots & (2n+1)x_1^{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^{2n+1} \\ 0 & 1 & 2x_n & \cdots & (2n+1)x_n^{2n} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{2n+1} \end{pmatrix} = \begin{pmatrix} y_0 \\ y'_0 \\ y_1 \\ y'_1 \\ \vdots \\ y_n \\ y'_n \end{pmatrix}.$$

Fortunately, there is a more direct way to write the interpolating polynomial using basis functions tailored to Hermite interpolation.

Proposition 2.14 There is at most one polynomial H of degree at most $2n+1$ satisfying

$$H(x_k) = y_k \qquad \text{and} \qquad H'(x_k) = y'_k, \quad k = 0, 1, \ldots, n.$$

Proof. Suppose H_1 and H_2 both satisfy the conditions, and let

$$q = H_1 - H_2.$$

Then

$$q(x_k) = 0 \qquad \text{and} \qquad q'(x_k) = 0$$

for every k. Thus each x_k is a double root of q, so

$$\prod_{k=0}^{n}(x-x_k)^2$$

divides $q(x)$. The divisor has degree $2n+2$, but q has degree at most $2n+1$. Hence $q \equiv 0$, so $H_1 = H_2$. □

Theorem 2.15 Let ℓ_k be the Lagrange basis functions for the nodes $x_0, \ldots, x_n$, and define

$$H_k(x) = \big(1 - 2(x-x_k)\ell_k'(x_k)\big)\,\ell_k(x)^2,$$

$$K_k(x) = (x-x_k)\ell_k(x)^2.$$

Then the unique polynomial of degree at most $2n+1$ satisfying

$$H(x_k) = y_k \qquad \text{and} \qquad H'(x_k) = y_k', \quad k = 0, 1, \ldots, n,$$

is the **Hermite interpolating polynomial**

$$H(x) = \sum_{k=0}^{n} y_k H_k(x) + \sum_{k=0}^{n} y_k' K_k(x).$$

Proof. Since $\ell_k(x_j) = \delta_{jk}$, we immediately get

$$H_k(x_j) = \delta_{jk} \qquad \text{and} \qquad K_k(x_j) = 0.$$

Next, because $\ell_k(x_j) = 0$ when $j \neq k$, differentiating shows

$$H_k'(x_j) = 0 \qquad \text{and} \qquad K_k'(x_j) = 0, \quad j \neq k.$$

At $x = x_k$, we have $\ell_k(x_k) = 1$, so

$$K_k'(x_k) = 1.$$

Also,

$$H_k'(x) = -2\ell_k'(x_k)\ell_k(x)^2 + \big(1 - 2(x-x_k)\ell_k'(x_k)\big)\,2\ell_k(x)\ell_k'(x),$$

and evaluating at $x = x_k$ gives

$$H_k'(x_k) = -2\ell_k'(x_k) + 2\ell_k'(x_k) = 0.$$

Therefore, for each node x_j,

$$H(x_j) = \sum_{k=0}^{n} y_k H_k(x_j) + \sum_{k=0}^{n} y'_k K_k(x_j) = y_j,$$

and

$$H'(x_j) = \sum_{k=0}^{n} y_k H'_k(x_j) + \sum_{k=0}^{n} y'_k K'_k(x_j) = y'_j.$$

By Proposition 2.14, this interpolating polynomial is unique. □

Hermite interpolation matches both the function values and the prescribed tangent data. Figure 4 shows the two-node case schematically.

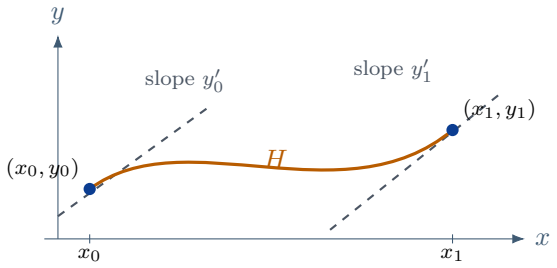


FIGURE 4

Using a single high-degree polynomial is not always the best choice. A simpler alternative is to interpolate one interval at a time.

Definition 2.16 Assume

$$x_0 < x_1 < \cdots < x_n.$$

On each interval $[x_k, x_{k+1}]$, define the **linear interpolant**

$$p_k(x) = y_k + \frac{y_{k+1} - y_k}{x_{k+1} - x_k}(x - x_k).$$

The **piecewise linear interpolant** p is defined by

$$p(x) = p_k(x), \quad x_k \le x \le x_{k+1}.$$

The piecewise linear interpolant connects consecutive data points by line segments, as shown in Figure 5. The dashed curve is the original function f, and the solid curve is the piecewise linear interpolant p.

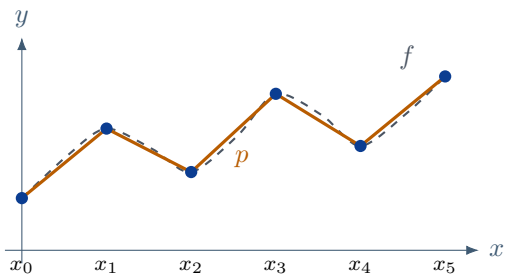


FIGURE 5

How accurate is the piecewise linear interpolant? The error is controlled by the second derivative of f:

Theorem 2.17 Suppose $f \in C^2([x_0, x_n])$ and

$$|f''(x)| \leq M$$

for all $x \in [x_0, x_n]$. Let

$$h_k = x_{k+1} - x_k \qquad \text{and} \qquad h_{\max} = \max_{0 \leq k \leq n-1} h_k.$$

Then for every $x \in [x_k, x_{k+1}]$ there exists $\xi_x \in [x_k, x_{k+1}]$ such that

$$f(x) - p(x) = \frac{f''(\xi_x)}{2}(x - x_k)(x - x_{k+1}),$$

and consequently

$$|f(x) - p(x)| \leq \frac{M h_{\max}^2}{8}.$$

Proof. On a fixed interval $[x_k, x_{k+1}]$, the function p_k is exactly the degree-1 interpolating polynomial for f at the nodes x_k and x_{k+1}. Applying Theorem 2.12 with $n = 1$ yields

$$f(x) - p_k(x) = \frac{f''(\xi_x)}{2}(x - x_k)(x - x_{k+1})$$

for some $\xi_x \in [x_k, x_{k+1}]$. Since

$$|(x - x_k)(x - x_{k+1})|$$

is maximized at the midpoint of the interval, its maximum value is $h_k^2/4$. Hence

$$|f(x) - p(x)| \leq \frac{M}{2} \cdot \frac{h_k^2}{4} = \frac{Mh_k^2}{8} \leq \frac{Mh_{\max}^2}{8}.$$

□

Piecewise linear interpolation is easy to compute and easy to evaluate, but the derivative is usually discontinuous at the interpolation nodes. If smoother approximations are required, then piecewise cubic interpolation is usually much more effective.

Cubic Spline Interpolation

Definition 2.18 A **cubic spline** on the nodes $x_0 < \cdots < x_n$ is a function S such that on each interval $[x_k, x_{k+1}]$,

$$S(x) = S_k(x),$$

where

$$S_k(x) = a_k + b_k(x - x_k) + c_k(x - x_k)^2 + d_k(x - x_k)^3, \quad k = 0, 1, \ldots, n-1,$$

and the resulting piecewise function belongs to $C^2([x_0, x_n])$.

For a cubic spline, adjacent cubic pieces share their value and enough derivative information at each interior node. This smooth joining is illustrated in Figure 6.

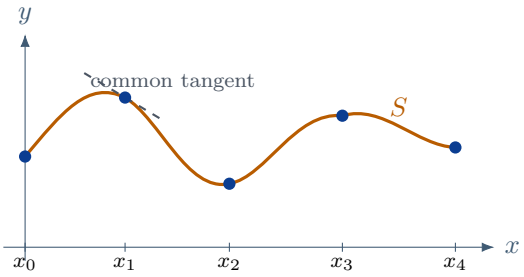


FIGURE 6

Because there are n intervals and four coefficients on each one, the spline has $4n$ unknown coefficients. These are determined by interpolation conditions and smoothness conditions.

Proposition 2.19 Let

$$h_k = x_{k+1} - x_k.$$

For a natural cubic spline, the coefficients satisfy:

$$a_k = y_k, \quad k = 0, \ldots, n-1,$$

$$b_k h_k + c_k h_k^2 + d_k h_k^3 = y_{k+1} - a_k, \quad k = 0, \ldots, n-1,$$

$$b_k + 2c_k h_k + 3d_k h_k^2 = b_{k+1}, \quad k = 0, \ldots, n-2,$$

$$c_k + 3d_k h_k = c_{k+1}, \quad k = 0, \ldots, n-2,$$

and the natural boundary conditions are

$$c_0 = 0 \qquad \text{and} \qquad c_n = 0,$$

where c_n is defined by the continuity relation at the last node.

Proof. The interpolation condition at the left endpoint gives

$$S_k(x_k) = a_k = y_k.$$

At the right endpoint,

$$S_k(x_{k+1}) = a_k + b_k h_k + c_k h_k^2 + d_k h_k^3 = y_{k+1},$$

which gives the second relation. Differentiating yields

$$S_k'(x) = b_k + 2c_k(x - x_k) + 3d_k(x - x_k)^2,$$

so continuity of the first derivative at x_{k+1} gives

$$b_k + 2c_k h_k + 3d_k h_k^2 = b_{k+1}.$$

Similarly,

$$S_k''(x) = 2c_k + 6d_k(x - x_k),$$

and continuity of the second derivative at x_{k+1} gives

$$2c_k + 6d_k h_k = 2c_{k+1},$$

or equivalently

$$c_k + 3d_k h_k = c_{k+1}.$$

Finally, the natural spline conditions require

$$S''(x_0) = 0 \qquad \text{and} \qquad S''(x_n) = 0,$$

which are equivalent to $c_0 = 0$ and $c_n = 0$ in this notation. □

Other boundary conditions are also common. A **complete (clamped) spline** prescribes endpoint derivatives:

$$S'(x_0) = y'_0 \qquad \text{and} \qquad S'(x_n) = y'_n.$$

A **periodic (repeating) spline** requires the first and second derivatives to agree at the two endpoints:

$$S'(x_0) = S'(x_n) \qquad \text{and} \qquad S''(x_0) = S''(x_n).$$

A **not-a-knot spline** imposes continuity of the third derivative at the first and last interior nodes:

$$S_0^{(3)}(x_1) = S_1^{(3)}(x_1) \qquad \text{and} \qquad S_{n-2}^{(3)}(x_{n-1}) = S_{n-1}^{(3)}(x_{n-1}).$$

Proposition 2.20 Let $a_k = y_k$, and let $c_0, \ldots, c_n$ be the quadratic coefficients in the natural spline notation of Proposition 2.19. Then

$$h_{k-1}c_{k-1} + 2(h_{k-1} + h_k)c_k + h_k c_{k+1} = 3\left(\frac{a_{k+1} - a_k}{h_k} - \frac{a_k - a_{k-1}}{h_{k-1}}\right)$$

for $k = 1, \ldots, n-1$, together with

$$c_0 = 0 \qquad \text{and} \qquad c_n = 0.$$

Equivalently,

$$\begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ h_0 & 2(h_0+h_1) & h_1 & \cdots & 0 \\ 0 & h_1 & 2(h_1+h_2) & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & h_{n-1} \\ 0 & \cdots & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{n-1} \\ c_n \end{pmatrix} = \begin{pmatrix} 0 \\ r_1 \\ \vdots \\ r_{n-1} \\ 0 \end{pmatrix},$$

where

$$r_k = 3\left(\frac{a_{k+1}-a_k}{h_k} - \frac{a_k - a_{k-1}}{h_{k-1}}\right).$$

Proof. From

$$c_k + 3d_k h_k = c_{k+1},$$

we obtain

$$d_k = \frac{c_{k+1}-c_k}{3h_k}.$$

Substituting this into

$$b_k h_k + c_k h_k^2 + d_k h_k^3 = a_{k+1} - a_k$$

gives

$$b_k = \frac{a_{k+1}-a_k}{h_k} - \frac{h_k}{3}(2c_k + c_{k+1}).$$

Now impose continuity of the first derivative:

$$b_{k-1} + 2c_{k-1}h_{k-1} + 3d_{k-1}h_{k-1}^2 = b_k.$$

Substituting the formulas for b_{k-1}, b_k, and d_{k-1}, and then simplifying, yields

$$h_{k-1}c_{k-1} + 2(h_{k-1}+h_k)c_k + h_k c_{k+1} = 3\left(\frac{a_{k+1}-a_k}{h_k} - \frac{a_k - a_{k-1}}{h_{k-1}}\right).$$

This holds for each interior index $k = 1, \ldots, n-1$, and the natural boundary conditions supply the first and last equations. □

The spline matrix is tridiagonal, and its interior diagonal entries satisfy

$$2(h_{k-1}+h_k) > h_{k-1} + h_k.$$

Thus the matrix is strictly diagonally dominant and therefore nonsingular. In practice, spline systems are typically much better behaved numerically than Vandermonde systems, and their tridiagonal structure allows us to solve them in $O(n)$ operations rather than $O(n^3)$.

Parametric and Bézier Curves

Interpolation is also useful when the data lie on a curve in the plane. In that case it is often better to interpolate the coordinates separately with respect to a parameter.

Example 2.21 The unit circle can be parameterized as $(x(t), y(t))$ with

$$x(t) = \cos(2\pi t) \qquad \text{and} \qquad y(t) = \sin(2\pi t), \quad 0 \le t \le 1.$$

The same circle can also be traversed at a different speed by using, for example,

$$x(t) = \cos\bigl(2\pi t^4\bigr) \qquad \text{and} \qquad y(t) = \sin\bigl(2\pi t^4\bigr), \quad 0 \le t \le 1.$$

These equations describe the same path in the plane, but they do not describe the same motion along that path.

To interpolate a planar curve from data points (x_j, y_j), one common approach is to choose parameter values t_j, interpolate the pairs (t_j, x_j) and (t_j, y_j) separately, and then draw the parametric curve

$$\bigl(x(t), y(t)\bigr).$$

In practice, cubic splines are often used for the coordinate functions.

Interpolation is not the only way to define a useful curve. In computer graphics, we often want a curve that passes through the first and last points but is only influenced by the interior points.

Definition 2.22 For $0 \le t \le 1$ and $k = 0, 1, \ldots, n$, define the **Bernstein polynomials** by

$$B_{k,n}(t) = \binom{n}{k} t^k (1-t)^{n-k}.$$

Definition 2.23 Given control values $x_0, x_1, \ldots, x_n$, the **Bézier curve** is

$$p(t) = \sum_{k=0}^{n} x_k B_{k,n}(t), \quad 0 \le t \le 1.$$

For control points $P_k = (x_k, y_k)$ in the plane, the **parametric Bézier curve** is

$$P(t) = \sum_{k=0}^{n} P_k B_{k,n}(t).$$

Proposition 2.24 Let

$$p(t) = \sum_{k=0}^{n} x_k B_{k,n}(t).$$

Then

$$p(0) = x_0 \qquad \text{and} \qquad p(1) = x_n,$$

and

$$p'(t) = n \sum_{k=0}^{n-1} (x_{k+1} - x_k) B_{k,n-1}(t).$$

In particular,

$$p'(0) = n(x_1 - x_0) \qquad \text{and} \qquad p'(1) = n(x_n - x_{n-1}).$$

Proof. At $t = 0$, every Bernstein polynomial vanishes except $B_{0,n}(0) = 1$, so $p(0) = x_0$. Similarly, at $t = 1$ only $B_{n,n}(1) = 1$ survives, so $p(1) = x_n$. For the derivative, differentiate term by term:

$$\frac{\mathrm{d}}{\mathrm{d}t} B_{k,n}(t) = \binom{n}{k} \left(k t^{k-1} (1-t)^{n-k} - (n-k) t^k (1-t)^{n-k-1} \right).$$

Using the identities

$$k\binom{n}{k} = n\binom{n-1}{k-1} \qquad \text{and} \qquad (n-k)\binom{n}{k} = n\binom{n-1}{k},$$

we obtain

$$B'_{k,n}(t) = n \left(B_{k-1,n-1}(t) - B_{k,n-1}(t) \right),$$

where terms with out-of-range indices are interpreted as zero. Therefore

$$p'(t) = n \sum_{k=0}^{n} x_k \left(B_{k-1,n-1}(t) - B_{k,n-1}(t) \right) = n \sum_{k=0}^{n-1} (x_{k+1} - x_k) B_{k,n-1}(t).$$

Evaluating at $t = 0$ and $t = 1$ yields the endpoint derivative formulas. □

Figure 7 shows the control polygon and the resulting cubic Bézier curve.

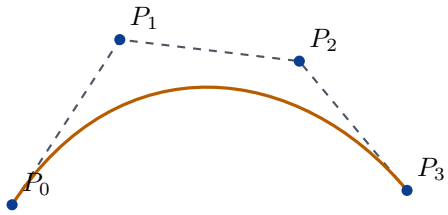


Figure 7

As Figure 7 shows, the interior control points generally do not lie on the curve, but they pull the curve toward themselves. This is why Bézier curves are so useful in fonts, drawing programs, and computer graphics more generally.

Lecture 3 — Tuesday, January 26, 2016

3. Fourier Analysis

Fourier Series

Fourier analysis studies the approximation of periodic data by combinations of sine and cosine waves, or equivalently by complex exponentials. It converts information in the time or spatial domain into information in the frequency domain. This perspective appears in touch-tone dialing, audio processing, image compression, disk storage, DVDs, JPEGs, and MP3 compression.

Example 3.1 Each row and each column of a telephone keypad is assigned its own frequency. When a button is pressed, one row frequency and one column frequency are played simultaneously. The two frequencies are chosen from different ranges so that ordinary speech is less likely to interfere with the signal. In a noisy recording of a phone number, we can often see spikes in the waveform, but the actual digits are identified by extracting the dominant frequencies in each local time window.

A telephone keypad can be organized by the dual-tone frequency structure in Figure 8: each key is identified by one low row frequency and one higher column frequency.

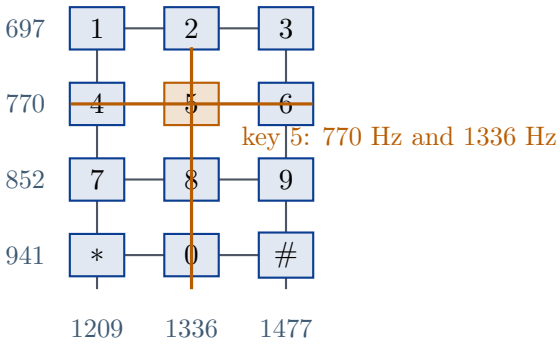


Figure 8

The same idea is visible locally in Figure 9: a time signal may show only bursts of activity, while a local frequency spectrum reveals the pair of tones in one burst.

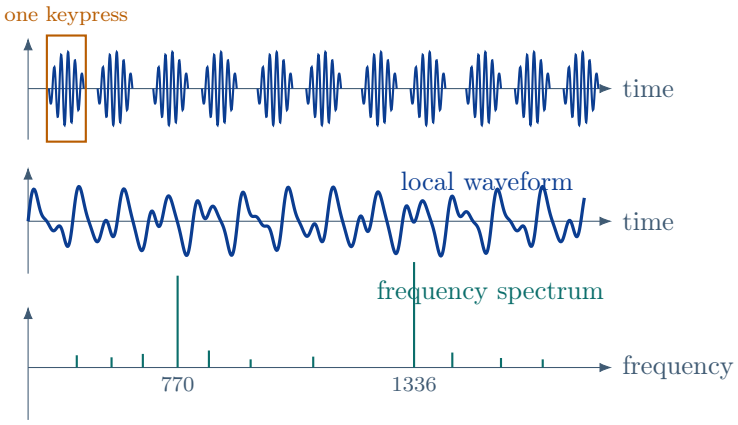


Figure 9

Example 3.2 In audio processing, Fourier methods make it possible to separate low-frequency and high-frequency contributions and to discard frequencies that carry little perceptual information. In image processing, a grayscale image is simply a large array of numbers, often with entries between 0 and 255 or between 0 and 1. Fourier compression works by observing that much of the useful information is concentrated in relatively few frequency coefficients.

The effect of discarding less important coefficients is shown schematically in Figure 10: the main image structure remains visible while fine detail is progressively removed.

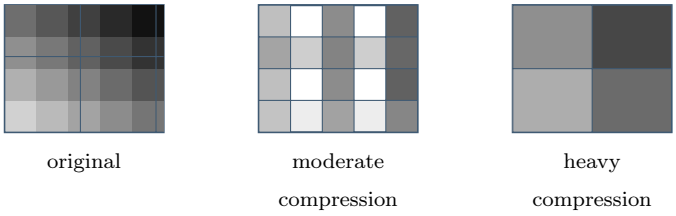


FIGURE 10

The central computational tool is the **fast Fourier transform** (**FFT**), which is widely regarded as one of the most important practical algorithms in numerical computation.

Let $f(t)$ be a periodic function with period T, so that

$$f(t+T) = f(t).$$

The basic building blocks are the **trigonometric functions** with period T:

$$\sin\left(\frac{2\pi kt}{T}\right) \qquad \text{and} \qquad \cos\left(\frac{2\pi kt}{T}\right), \quad k = 0, 1, 2, \ldots.$$

Each of these is T-periodic whenever k is an integer, and finite linear combinations of them are therefore also T-periodic.

Figure 11 shows the basic sine and cosine waves with period 1.

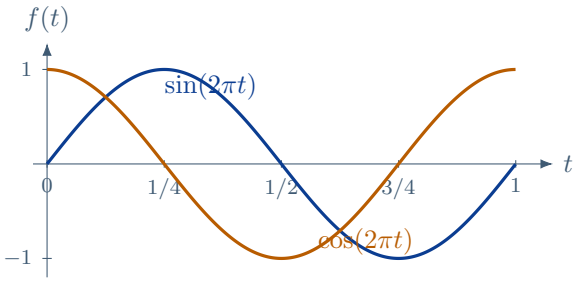


FIGURE 11

Example 3.3 The functions $\sin t$ and $\cos t$ have period 2π. The functions $\sin(2\pi t)$ and

$\cos(2\pi t)$ have period 1. More generally,

$$\sin\left(\frac{2\pi t}{T}\right) \qquad \text{and} \qquad \cos\left(\frac{2\pi t}{T}\right)$$

have period T, while

$$\sin\left(\frac{2\pi kt}{T}\right) \qquad \text{and} \qquad \cos\left(\frac{2\pi kt}{T}\right)$$

have period T/k.

For example, the function

$$f(t) = \sin\left(\frac{2\pi t}{7}\right) + \frac{1}{5}\sin\left(\frac{24\pi t}{7}\right)$$

is 7-periodic. The first term has fundamental period 7, and the second has fundamental period 7/12, but both are still 7-periodic, so their sum is 7-periodic as well.

This example combines a low-frequency term with a smaller high-frequency oscillation. Figure 12 shows how both terms remain compatible with the longer period.

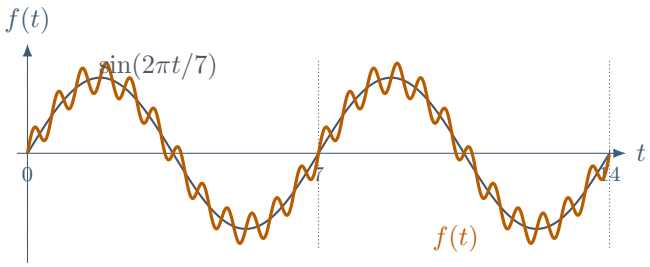


FIGURE 12

For functions with suitable regularity, we expect an expansion of the form

$$f(t) = a_0 + \sum_{k=1}^{\infty} a_k \cos\left(\frac{2\pi kt}{T}\right) + \sum_{k=1}^{\infty} b_k \sin\left(\frac{2\pi kt}{T}\right).$$

To compute the coefficients, it is enough to understand the orthogonality of the trigonometric basis.

Orthogonality

For simplicity, take $T = 2\pi$ and work on the interval $[0, 2\pi]$. Then the Fourier series becomes

$$f(t) = a_0 + \sum_{k=1}^{\infty} a_k \cos(kt) + \sum_{k=1}^{\infty} b_k \sin(kt).$$

Proposition 3.4 For integers $j, k \geq 1$,

$$\int_0^{2\pi} \cos(kt)\sin(jt)\,\mathrm{d}t = 0,$$

$$\int_0^{2\pi} \cos(kt)\cos(jt)\,\mathrm{d}t = \begin{cases} 0, & k \neq j, \\ \pi, & k = j, \end{cases}$$

and

$$\int_0^{2\pi} \sin(kt)\sin(jt)\,\mathrm{d}t = \begin{cases} 0, & k \neq j, \\ \pi, & k = j. \end{cases}$$

Also,

$$\int_0^{2\pi} \sin(kt)\,\mathrm{d}t = 0 \qquad \text{and} \qquad \int_0^{2\pi} \cos(kt)\,\mathrm{d}t = 0.$$

Proof. Using the product-to-sum identities,

$$\cos(kt)\cos(jt) = \frac{1}{2}\cos((k-j)t) + \frac{1}{2}\cos((k+j)t),$$

$$\sin(kt)\sin(jt) = \frac{1}{2}\cos((k-j)t) - \frac{1}{2}\cos((k+j)t),$$

and

$$\sin(jt)\cos(kt) = \frac{1}{2}\sin((j+k)t) + \frac{1}{2}\sin((j-k)t).$$

Each sine term integrates to zero over a full period, and each cosine term with nonzero frequency also integrates to zero over $[0, 2\pi]$. This immediately proves the mixed orthogonality and the $k \neq j$ cases. When $k = j$, we use

$$\cos^2(kt) = \frac{1 + \cos(2kt)}{2} \qquad \text{and} \qquad \sin^2(kt) = \frac{1 - \cos(2kt)}{2}.$$

Integrating over $[0, 2\pi]$ gives

$$\int_0^{2\pi} \cos^2(kt)\,\mathrm{d}t = \int_0^{2\pi} \sin^2(kt)\,\mathrm{d}t = \pi.$$

□

Theorem 3.5 Assume the trigonometric series

$$f(t) = a_0 + \sum_{k=1}^{\infty} a_k \cos(kt) + \sum_{k=1}^{\infty} b_k \sin(kt)$$

converges to f in $L^2([0, 2\pi])$. Then

$$a_0 = \frac{1}{2\pi} \int_0^{2\pi} f(t)\,\mathrm{d}t,$$

and for $k \geq 1$,

$$a_k = \frac{1}{\pi} \int_0^{2\pi} f(t)\cos(kt)\,\mathrm{d}t \qquad \text{and} \qquad b_k = \frac{1}{\pi} \int_0^{2\pi} f(t)\sin(kt)\,\mathrm{d}t.$$

Proof. Taking the inner product of the partial sums with the constant function and passing to the L^2 limit yields

$$\int_0^{2\pi} f(t)\,\mathrm{d}t = 2\pi a_0.$$

Next, take the inner product with $\cos(mt)$:

$$\begin{aligned}\int_0^{2\pi} f(t)\cos(mt)\,\mathrm{d}t = a_0 &\int_0^{2\pi} \cos(mt)\,\mathrm{d}t \\ &+ \sum_{k=1}^{\infty} a_k \int_0^{2\pi} \cos(kt)\cos(mt)\,\mathrm{d}t \\ &+ \sum_{k=1}^{\infty} b_k \int_0^{2\pi} \sin(kt)\cos(mt)\,\mathrm{d}t.\end{aligned}$$

Orthogonality makes every term vanish except the one with $k = m$ in the cosine–cosine sum, and continuity of the inner product under L^2 convergence gives

$$\int_0^{2\pi} f(t)\cos(mt)\,\mathrm{d}t = \pi a_m.$$

The same argument with $\sin(mt)$ gives the formula for b_m. □

Example 3.6 Consider the 2π-periodic function

$$f(t) = \begin{cases} 1, & 0 \le t < \pi, \\ 0, & \pi \le t < 2\pi. \end{cases}$$

Its average value is

$$a_0 = \frac{1}{2\pi}\int_0^{2\pi} f(t)\,\mathrm{d}t = \frac{1}{2\pi}\int_0^{\pi} 1\,\mathrm{d}t = \frac{1}{2}.$$

For $k \ge 1$,

$$a_k = \frac{1}{\pi}\int_0^{2\pi} f(t)\cos(kt)\,\mathrm{d}t = \frac{1}{\pi}\int_0^{\pi} \cos(kt)\,\mathrm{d}t = \frac{\sin(k\pi)}{\pi k} = 0.$$

Similarly,

$$b_k = \frac{1}{\pi}\int_0^{2\pi} f(t)\sin(kt)\,\mathrm{d}t = \frac{1}{\pi}\int_0^{\pi} \sin(kt)\,\mathrm{d}t = \frac{1-\cos(k\pi)}{\pi k} = \frac{1-(-1)^k}{\pi k}.$$

Thus

$$b_k = \begin{cases} \dfrac{2}{\pi k}, & k \text{ odd}, \\ 0, & k \text{ even}, \end{cases}$$

Thus, at every continuity point of the periodically extended function, the Fourier series converges to

$$\frac{1}{2} + \frac{2}{\pi}\left(\sin t + \frac{1}{3}\sin(3t) + \frac{1}{5}\sin(5t) + \cdots\right).$$

At each jump $t \in \pi\mathbb{Z}$, the series converges instead to the midpoint of the one-sided limits, namely $1/2$.

The resulting square wave is shown in Figure 13.

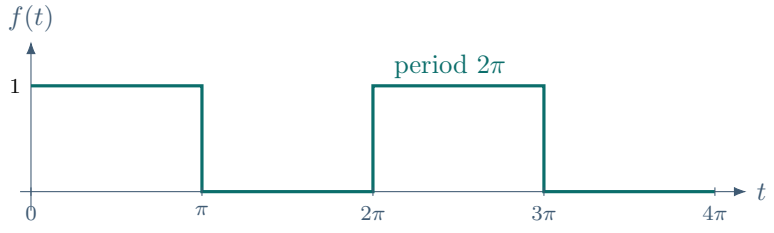


FIGURE 13

Complex Form

Definition 3.7 A complex number has the form

$$z = a + ib,$$

where $a, b \in \mathbb{R}$ and $i^2 = -1$. Its complex conjugate is

$$\overline{z} = a - ib.$$

If

$$z_1 = a_1 + ib_1 \qquad \text{and} \qquad z_2 = a_2 + ib_2,$$

then

$$z_1 + z_2 = (a_1 + a_2) + i(b_1 + b_2),$$

$$z_1 z_2 = (a_1 a_2 - b_1 b_2) + i(a_1 b_2 + a_2 b_1),$$

and

$$|z|^2 = z\overline{z} = a^2 + b^2.$$

The rectangular and polar descriptions of $z = a + ib$ are shown in Figure 14.

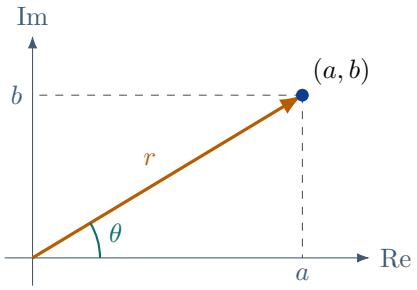


FIGURE 14

Proposition 3.8 (Euler's formula) For every real θ,

$$e^{i\theta} = \cos\theta + i\sin\theta.$$

Remark 3.9 Every nonzero complex number can be written in polar form as

$$z = re^{i\theta},$$

where $r = |z|$ and θ is an argument of z. In this form,

$$z_1 z_2 = r_1 r_2 e^{i(\theta_1+\theta_2)} \qquad \text{and} \qquad \overline{z} = re^{-i\theta}.$$

Proposition 3.10 The real Fourier series

$$f(t) = a_0 + \sum_{k=1}^{\infty} a_k \cos(kt) + \sum_{k=1}^{\infty} b_k \sin(kt)$$

can be written equivalently as

$$f(t) = \sum_{k=-\infty}^{\infty} c_k e^{ikt},$$

where

$$c_0 = a_0, \qquad c_k = \frac{a_k - ib_k}{2} \quad (k \geq 1), \qquad \text{and} \qquad c_{-k} = \frac{a_k + ib_k}{2} \quad (k \geq 1).$$

Proof. Using Euler's identities,

$$\cos(kt) = \frac{e^{ikt} + e^{-ikt}}{2} \qquad \text{and} \qquad \sin(kt) = \frac{e^{ikt} - e^{-ikt}}{2i}.$$

Substituting into the real Fourier series and collecting the coefficients of e^{ikt} and e^{-ikt} gives the stated formulas. □

For many signals, the higher-frequency harmonics contribute much less than the lower ones. This is often visible in the power spectrum

$$|c_k|^2,$$

which measures the power in the kth harmonic. A common approximation is to truncate the series:

$$f(t) \approx \sum_{k=-M}^{M} c_k e^{ikt}.$$

This centered truncation preserves the natural symmetry between positive and negative frequen-

cies. Figure 15 illustrates the truncation in the frequency domain: coefficients with $|k| > M$ are discarded.

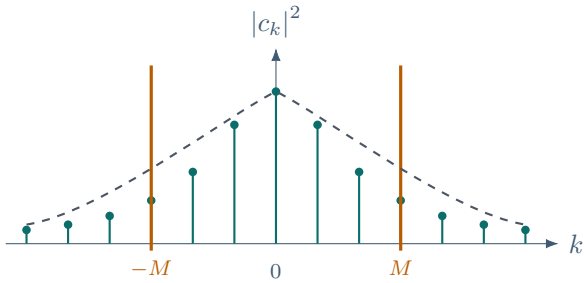


Figure 15

Discrete Fourier Transform

In practice, we usually do not know the function $f(t)$ exactly. Instead, we have sampled values. Assume the data are periodic with period T, assume N is even, and suppose N equally spaced observations are taken at

$$t_j = \frac{jT}{N}, \quad j = 0, 1, \ldots, N-1.$$

Write the sampled values as

$$f_j = f(t_j),$$

and interpret $f_N = f_0$ because of periodicity.

Trigonometric interpolation seeks coefficients F_k such that

$$f_n = \sum_{k=-N/2}^{N/2-1} F_k e^{i2\pi kn/N}, \quad n = 0, 1, \ldots, N-1.$$

This is the discrete analogue of the complex Fourier series.

Figure 16 shows the basic interpretation: the data f_n live in the time or spatial domain, while the coefficients F_k measure frequency content.

The centered frequency range can be converted into the stored array order used by the discrete

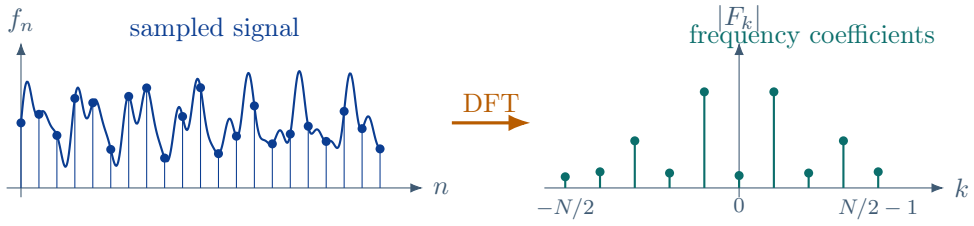


FIGURE 16

Fourier transform. Split the interpolation formula as

$$f_n = \sum_{k=-N/2}^{-1} F_k e^{i2\pi kn/N} + \sum_{k=0}^{N/2-1} F_k e^{i2\pi kn/N}.$$

In the first sum, set $s = k + N$, so that $s = N/2, \ldots, N-1$ and $k = s - N$. Then

$$f_n = \sum_{s=N/2}^{N-1} F_{s-N} e^{i2\pi(s-N)n/N} + \sum_{k=0}^{N/2-1} F_k e^{i2\pi kn/N}.$$

Since $e^{-i2\pi n} = 1$ for every integer n, this is

$$f_n = \sum_{s=N/2}^{N-1} F_{s-N} e^{i2\pi sn/N} + \sum_{k=0}^{N/2-1} F_k e^{i2\pi kn/N}.$$

We make the coefficients periodic with period N, so that $F_s = F_{s-N}$. Thus

$$f_n = \sum_{s=N/2}^{N-1} F_s e^{i2\pi sn/N} + \sum_{k=0}^{N/2-1} F_k e^{i2\pi kn/N} = \sum_{k=0}^{N-1} F_k e^{i2\pi kn/N}.$$

Therefore the centered interpolation problem is equivalently a system of N equations in the N unknowns $F_0, F_1, \ldots, F_{N-1}$.

Figure 17 illustrates why we often shift the coefficients before plotting them: the negative frequencies stored near the right end of the array move to the left of zero.

Definition 3.11 A **primitive Nth root of unity** is

$$W_N = e^{i2\pi/N}.$$

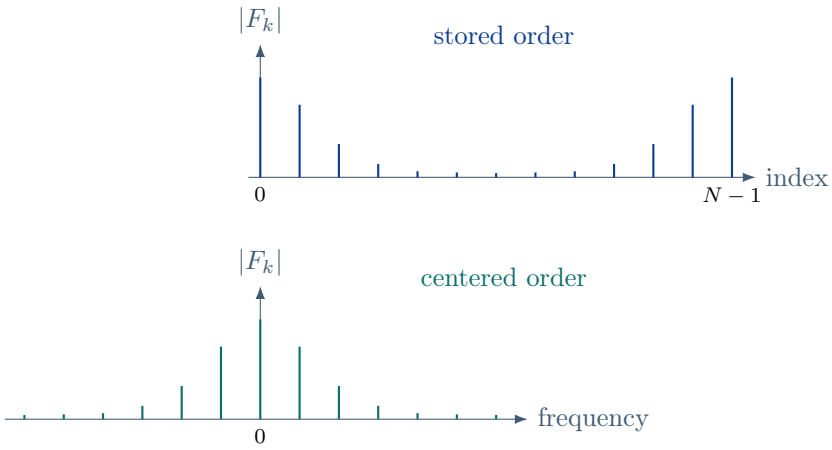


FIGURE 17

Its integer powers satisfy

$$W_N^N = 1, \qquad W_N^{-k} = W_N^{N-k}, \qquad \text{and} \qquad W_N^k = W_N^{k \bmod N}.$$

For $N = 6$, the powers of W_6 are the six equally spaced points on the unit circle shown in Figure 18.

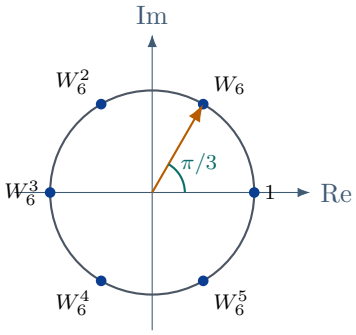


FIGURE 18

Definition 3.12 With the normalization used here, the discrete Fourier transform is

$$F_k = \frac{1}{N} \sum_{n=0}^{N-1} f_n W_N^{-kn}, \quad k = 0, 1, \ldots, N-1.$$

In matrix form,

$$\boldsymbol{F} = \frac{1}{N}\mathbf{M}\boldsymbol{f},$$

where

$$\mathbf{M} = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & W_N^{-1} & W_N^{-2} & \cdots & W_N^{-(N-1)} \\ 1 & W_N^{-2} & W_N^{-4} & \cdots & W_N^{-2(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & W_N^{-(N-1)} & W_N^{-2(N-1)} & \cdots & W_N^{-(N-1)^2} \end{pmatrix}.$$

Direct multiplication by this matrix requires $O(N^2)$ operations.

Example 3.13 Consider the data

$$(0,1), \left(\frac{1}{6}, \frac{1}{2}\right), \left(\frac{2}{6}, -\frac{1}{2}\right), \left(\frac{3}{6}, -1\right), \left(\frac{4}{6}, -\frac{1}{2}\right), \left(\frac{5}{6}, \frac{1}{2}\right).$$

These are the samples of $\cos(2\pi t)$ at the six equally spaced points $t_n = n/6$, so

$$\boldsymbol{f} = \left(1, \frac{1}{2}, -\frac{1}{2}, -1, -\frac{1}{2}, \frac{1}{2}\right).$$

Here

$$W_6 = e^{i\pi/3} = \frac{1}{2} + \frac{\sqrt{3}}{2}i.$$

The discrete Fourier transform coefficients are

$$F_k = \frac{1}{6}\sum_{n=0}^{5} f_n W_6^{-kn}.$$

Carrying out the sums gives

$$F_0 = 0, \qquad F_1 = \frac{1}{2}, \qquad F_2 = F_3 = F_4 = 0, \qquad \text{and} \qquad F_5 = \frac{1}{2}.$$

The stored index 5 represents frequency -1 modulo 6, because at the sample points $t_n = n/6$,

$$e^{i2\pi(5)t_n} = e^{-i2\pi t_n}.$$

The centered trigonometric interpolant is therefore

$$f(t) = \frac{1}{2}e^{i2\pi t} + \frac{1}{2}e^{-i2\pi t} = \cos(2\pi t).$$

Inverse Discrete Fourier Transform

Once the frequency coefficients are known, we can compress the data by setting very small coefficients or high-frequency coefficients to zero. The inverse discrete Fourier transform then reconstructs an approximation of the original signal.

Proposition 3.14 Let $\mathbf{M} = (W_N^{-kn})_{0\le k,n\le N-1}$ be the discrete Fourier transform matrix. Then

$$\mathbf{M}^*\mathbf{M} = N\mathbf{I},$$

where $\mathbf{M}^*$ denotes the conjugate transpose of $\mathbf{M}$. Consequently,

$$\mathbf{M}^{-1} = \frac{1}{N}\mathbf{M}^*,$$

and the inverse discrete Fourier transform is

$$f_n = \sum_{k=0}^{N-1} F_k W_N^{nk}, \quad n = 0, 1, \ldots, N-1.$$

Proof. The (r, s) entry of $\mathbf{M}^*\mathbf{M}$ is

$$\sum_{n=0}^{N-1} \overline{W_N^{-rn}}\, W_N^{-sn} = \sum_{n=0}^{N-1} W_N^{(r-s)n}.$$

If $r = s$, then every term is 1, so the sum is N. If $r \neq s$, this is a finite geometric series with ratio $W_N^{r-s} \neq 1$, hence

$$\sum_{n=0}^{N-1} W_N^{(r-s)n} = \frac{1 - W_N^{(r-s)N}}{1 - W_N^{r-s}} = \frac{1-1}{1 - W_N^{r-s}} = 0.$$

Thus $\mathbf{M}^*\mathbf{M} = N\mathbf{I}$, which implies

$$M^{-1} = \frac{1}{N}M^*.$$

Since $F = 1/NMf$, multiplying by M^* gives

$$f = M^*F,$$

which is precisely

$$f_n = \sum_{k=0}^{N-1} F_k W_N^{nk}.$$

□

Remark 3.15 MATLAB places the factor $1/N$ in the inverse transform instead of the forward transform. With MATLAB's convention,

$$F_n = \sum_{k=0}^{N-1} f_k W_N^{-kn} \qquad \text{and} \qquad f_n = \frac{1}{N} \sum_{k=0}^{N-1} F_k W_N^{nk}.$$

The two conventions are mathematically equivalent; they only differ by normalization.

Proposition 3.16 For discrete Fourier transform coefficients indexed modulo N,

$$F_{k+N} = F_k.$$

Equivalently,

$$F_{N-k} = F_{-k}.$$

If the sampled data f_n are real-valued, then

$$F_{N-k} = \overline{F_k}.$$

Proof. Using $W_N^{-Nn} = 1$,

$$F_{k+N} = \frac{1}{N} \sum_{n=0}^{N-1} f_n W_N^{-(k+N)n} = \frac{1}{N} \sum_{n=0}^{N-1} f_n W_N^{-kn} W_N^{-Nn} = F_k.$$

If f_n are real, then

$$\overline{F_k} = \frac{1}{N} \sum_{n=0}^{N-1} f_n \overline{W_N^{-kn}} = \frac{1}{N} \sum_{n=0}^{N-1} f_n W_N^{kn} = \frac{1}{N} \sum_{n=0}^{N-1} f_n W_N^{-(N-k)n} = F_{N-k}.$$

□

Aliasing

Example 3.17 Consider the two continuous signals

$$f_1(t) = \cos(2\pi \cdot 2 \cdot t) \qquad \text{and} \qquad f_2(t) = \cos(2\pi \cdot 4 \cdot t),$$

sampled at the six points

$$0, \frac{1}{6}, \frac{2}{6}, \frac{3}{6}, \frac{4}{6}, \frac{5}{6}.$$

For both functions the sampled values are

$$\left(1, -\frac{1}{2}, -\frac{1}{2}, 1, -\frac{1}{2}, -\frac{1}{2}\right).$$

Thus the discrete Fourier transform cannot distinguish the two original signals from these samples alone. In both cases the nonzero coefficients occur at the same discrete frequencies, namely

$$F_2 = F_4 = \frac{1}{2}.$$

Figure 19 shows the aliasing mechanism: the two different continuous cosines agree at the six sampled points.

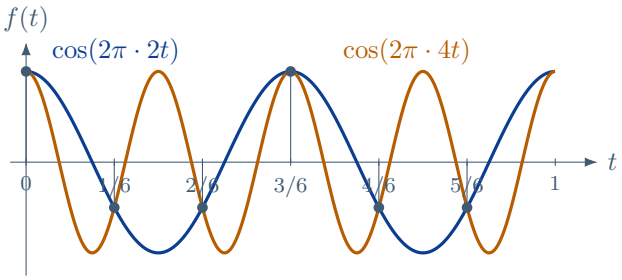


FIGURE 19

Aliasing occurs when the sampling rate is too low to distinguish high-frequency components from lower-frequency ones. In the reconstructed signal, a high frequency can masquerade as a different, lower frequency.

Theorem 3.18 (Nyquist–Shannon sampling theorem) If a continuous-time signal is bandlimited to frequencies $|\nu| \leq B$ and is sampled uniformly at a frequency strictly greater

than $2B$, then the samples determine the signal uniquely under the usual finite-energy hypotheses. Sampling below this rate permits distinct frequency components to alias. The boundary case at exactly $2B$ requires additional care, because a component at the Nyquist frequency can be lost for an unfortunate sampling phase.

Remark 3.19 Human hearing is roughly confined to the range from 20 Hz to 20 kHz, so audio CDs use a sampling rate of approximately 44 kHz, which is slightly more than twice the upper end of the audible range.

A grayscale image can be represented by an $N \times M$ array

$$\mathbf{f} = \begin{pmatrix} f_{00} & f_{01} & \cdots & f_{0,M-1} \\ f_{10} & f_{11} & \cdots & f_{1,M-1} \\ \vdots & \vdots & \ddots & \vdots \\ f_{N-1,0} & f_{N-1,1} & \cdots & f_{N-1,M-1} \end{pmatrix}.$$

Definition 3.20 Using MATLAB's normalization convention from the preceding remark, the **two-dimensional discrete Fourier transform** is

$$F_{k\ell} = \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} f_{nm} W_N^{-nk} W_M^{-\ell m},$$

and the **inverse two-dimensional discrete Fourier transform** is

$$f_{nm} = \frac{1}{NM} \sum_{k=0}^{N-1} \sum_{\ell=0}^{M-1} F_{k\ell} W_N^{nk} W_M^{\ell m}.$$

Remark 3.21 In practice, the two-dimensional transform is computed by first applying the one-dimensional discrete Fourier transform to each row and then applying it to each column of the resulting matrix. MATLAB provides the commands `fft2` and `ifft2` for this purpose.

Figure 20 shows the row-then-column structure of the two-dimensional transform.

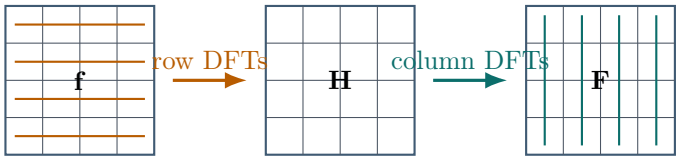


Figure 20

Fast Fourier Transform

The discrete Fourier transform is only practically useful because it can be computed much faster than a naive $O(N^2)$ matrix–vector multiplication.

Example 3.22 Suppose a machine performs approximately 5×10^{12} floating-point operations per second. If

$$N = 2^{20} = 1\,048\,576,$$

then an $O(N^2)$ algorithm takes roughly

$$\frac{N^2}{5 \times 10^{12}} \approx 0.22 \text{ seconds},$$

while an $O(N \log_2 N)$ algorithm takes roughly

$$\frac{N \log_2 N}{5 \times 10^{12}} \approx 4.2 \times 10^{-6} \text{ seconds}.$$

In one second, an $O(N^2)$ algorithm can handle a problem size of only about

$$N \approx \sqrt{5 \times 10^{12}} \approx 2.24 \times 10^6,$$

whereas an $O(N \log_2 N)$ algorithm can handle a problem size on the order of

$$N \approx 1.35 \times 10^{11}.$$

Figure 21 gives the qualitative reason that the fast Fourier transform is so important: the gap between quadratic growth and $N \log N$ growth quickly becomes enormous.

Proposition 3.23 Assume $N = 2^m$. Define the discrete Fourier transform by

$$F_k = \frac{1}{N} \sum_{n=0}^{N-1} f_n W_N^{-kn}.$$

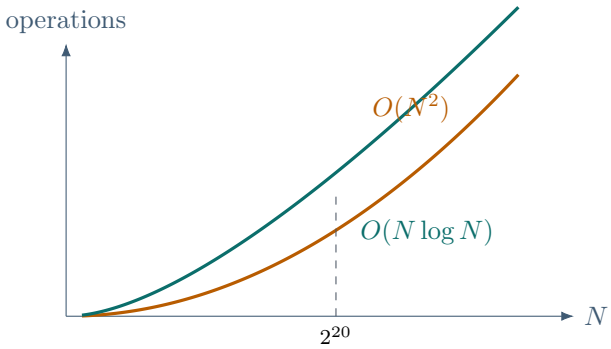


FIGURE 21

For $n = 0, 1, \ldots, N/2 - 1$, set

$$g_n = \frac{f_n + f_{n+N/2}}{2} \qquad \text{and} \qquad h_n = \frac{(f_n - f_{n+N/2})W_N^{-n}}{2}.$$

Then for $r = 0, 1, \ldots, N/2 - 1$,

$$F_{2r} = \frac{1}{N/2} \sum_{n=0}^{N/2-1} g_n W_{N/2}^{-rn},$$

and

$$F_{2r+1} = \frac{1}{N/2} \sum_{n=0}^{N/2-1} h_n W_{N/2}^{-rn}.$$

Proof. For even indices,

$$F_{2r} = \frac{1}{N} \sum_{n=0}^{N-1} f_n W_N^{-2rn}.$$

Split the sum into the first and second halves:

$$F_{2r} = \frac{1}{N} \sum_{n=0}^{N/2-1} f_n W_N^{-2rn} + \frac{1}{N} \sum_{n=0}^{N/2-1} f_{n+N/2} W_N^{-2r(n+N/2)}.$$

Since

$$W_N^{-2rN/2} = W_N^{-rN} = 1 \qquad \text{and} \qquad W_N^2 = W_{N/2},$$

this becomes

$$F_{2r} = \frac{1}{N} \sum_{n=0}^{N/2-1} (f_n + f_{n+N/2}) W_{N/2}^{-rn} = \frac{1}{N/2} \sum_{n=0}^{N/2-1} g_n W_{N/2}^{-rn}.$$

For odd indices,

$$F_{2r+1} = \frac{1}{N} \sum_{n=0}^{N-1} f_n W_N^{-(2r+1)n}.$$

Again splitting into two halves gives

$$F_{2r+1} = \frac{1}{N} \sum_{n=0}^{N/2-1} f_n W_N^{-n} W_{N/2}^{-rn} + \frac{1}{N} \sum_{n=0}^{N/2-1} f_{n+N/2} W_N^{-(n+N/2)} W_{N/2}^{-rn}.$$

Because

$$W_N^{-N/2} = -1,$$

the second sum becomes

$$-\frac{1}{N} \sum_{n=0}^{N/2-1} f_{n+N/2} W_N^{-n} W_{N/2}^{-rn}.$$

Combining the two parts yields

$$F_{2r+1} = \frac{1}{N} \sum_{n=0}^{N/2-1} (f_n - f_{n+N/2}) W_N^{-n} W_{N/2}^{-rn} = \frac{1}{N/2} \sum_{n=0}^{N/2-1} h_n W_{N/2}^{-rn}.$$

□

This recursion leads to the following algorithm.

```
function {Fk} = FastFourierTransform({fn}, N)
    if N == 1
        F0 = f0
    else
        for n = 0 to N/2 - 1
            gn = (fn + fn+N/2) / 2
            hn = ((fn - fn+N/2) WN^(-n)) / 2
        end
        {F2k}   = FastFourierTransform({gn}, N/2)
        {F2k+1} = FastFourierTransform({hn}, N/2)
    end
```

```
end
```

The first split in the radix-2 fast Fourier transform pairs entries separated by $N/2$, as shown in Figure 22.

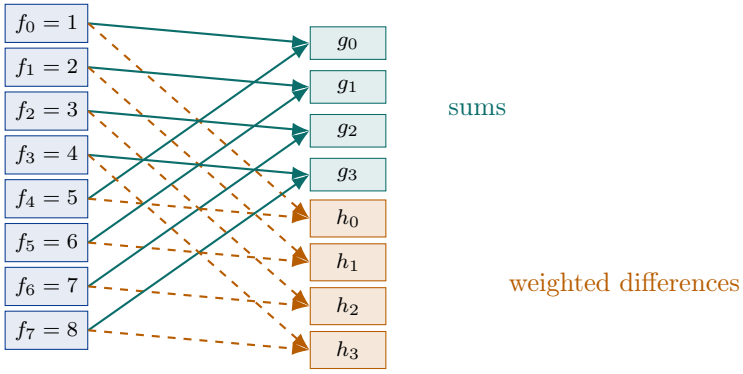


FIGURE 22

Example 3.24 Let

$$\boldsymbol{f} = (1, 2, 3, 4, 5, 6, 7, 8) \qquad \text{and} \qquad N = 8.$$

Then

$$g_0 = \frac{1+5}{2} = 3, \qquad g_1 = \frac{2+6}{2} = 4, \qquad g_2 = \frac{3+7}{2} = 5, \qquad \text{and} \qquad g_3 = \frac{4+8}{2} = 6.$$

The h_n values require

$$W_8 = e^{i\pi/4} = \frac{1+i}{\sqrt{2}} \qquad \text{and} \qquad W_8^{-1} = \frac{1-i}{\sqrt{2}}.$$

Thus

$$h_0 = \frac{1-5}{2} W_8^0 = -2,$$

$$h_1 = \frac{2-6}{2} W_8^{-1} = -2 \cdot \frac{1-i}{\sqrt{2}} = -\sqrt{2} + i\sqrt{2},$$

$$h_2 = \frac{3-7}{2} W_8^{-2} = -2(-i) = 2i,$$

and

$$h_3 = \frac{4-8}{2} W_8^{-3} = -2 \cdot \frac{-1-i}{\sqrt{2}} = \sqrt{2} + i\sqrt{2}.$$

The algorithm then recursively computes the discrete Fourier transforms of $\boldsymbol{g} = (3, 4, 5, 6)$ and $\boldsymbol{h} = (-2, -\sqrt{2} + i\sqrt{2}, 2i, \sqrt{2} + i\sqrt{2})$.

The final discrete Fourier transform coefficients are

$$\begin{aligned}
F_0 &= \frac{9}{2}, \\
F_1 &= -\frac{1}{2} + \frac{1+\sqrt{2}}{2} i, \\
F_2 &= -\frac{1}{2} + \frac{1}{2} i, \\
F_3 &= -\frac{1}{2} + \frac{\sqrt{2}-1}{2} i, \\
F_4 &= -\frac{1}{2}, \\
F_5 &= -\frac{1}{2} - \frac{\sqrt{2}-1}{2} i, \\
F_6 &= -\frac{1}{2} - \frac{1}{2} i, \\
F_7 &= -\frac{1}{2} - \frac{1+\sqrt{2}}{2} i.
\end{aligned}$$

Figure 23 shows the same computation: the recursion repeatedly combines pairs of values, and the even and odd output coefficients are interleaved at the end.

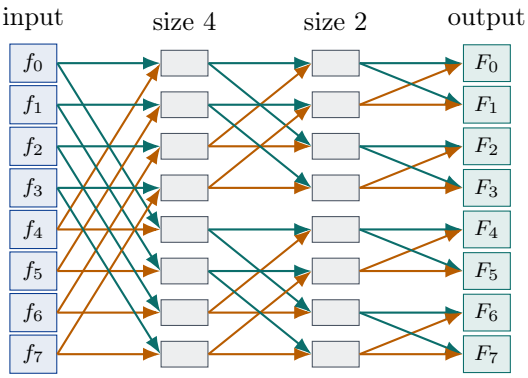


FIGURE 23

Proposition 3.25 If $T(N)$ denotes the number of operations required by the recursive fast Fourier transform when $N = 2^m$, then

$$T(N) = 2T\left(\frac{N}{2}\right) + O(N).$$

Therefore,

$$T(N) = O(N \log N).$$

Proof. At each level of the recursion, the algorithm forms the vectors $\boldsymbol{g}$ and $\boldsymbol{h}$, which takes $O(N)$ operations, and then solves two subproblems of size $N/2$. There are $\log_2 N$ levels in the recursion tree, and each level performs a total of $O(N)$ work. Hence the overall work is $O(N \log N)$. □

Figure 24 summarizes the proof of the running time: each level of the recursion tree has total work proportional to N, and there are $\log_2 N$ levels.

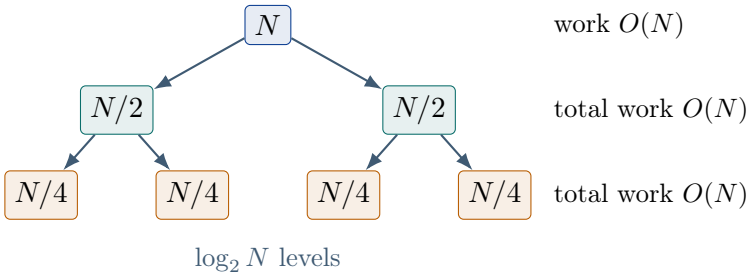


Figure 24

The fast Fourier transform presented here is a radix-2 **Cooley–Tukey algorithm**, popularized in 1965, although related ideas go back much earlier (to Gauss, in fact). General fast Fourier transform algorithms do not require the problem size to be a power of 2, but the power-of-2 case already shows the main idea clearly.

Lecture 4 — Thursday, February 11, 2016

4. Numerical Linear Algebra

PageRank and Eigenvalue Problems

Numerical linear algebra appears in many applications, and a particularly famous one is the **PageRank algorithm**. Larry Page and Sergey Brin developed it in 1996 while at Stanford University, and it eventually led to the creation of Google. The basic task of a search engine is not merely to locate pages containing a search term, but also to *rank* those pages by importance. A useful heuristic is that a page is important if important pages link to it.

Example 4.1 Imagine a random surfer who repeatedly visits web pages by clicking links. If the surfer is currently on a page, then the next page is chosen randomly from among the outlinks of the current page. Pages visited more often by this surfer should be regarded as more important.

To model this, let $\mathbf{G}$ be the adjacency matrix of a directed web graph:

$$\mathbf{G}_{ij} = \begin{cases} 1, & \text{if page } j \text{ links to page } i, \\ 0, & \text{otherwise.} \end{cases}$$

If page j has $\deg(j)$ outgoing links, then we define the probability matrix

$$\mathbf{P}_{ij} = \begin{cases} \dfrac{1}{\deg(j)}, & \text{if page } j \text{ links to page } i, \\ 0, & \text{otherwise.} \end{cases}$$

Thus the jth column of $\mathbf{P}$ describes the probabilities of moving away from page j.

Example 4.2 Consider the graph in Figure 25, with links

$$1 \to 2, \qquad 2 \to 3,\ 2 \to 4, \qquad 3 \to 4, \qquad \text{and} \qquad 4 \to 2,\ 4 \to 3.$$

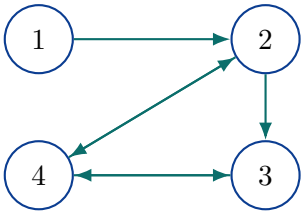


Figure 25

Its adjacency matrix is

$$\mathbf{G} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix},$$

and the corresponding probability matrix is

$$\mathbf{P} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & \frac{1}{2} \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & \frac{1}{2} & 1 & 0 \end{pmatrix}.$$

Each column sums to 1.

Dead-end pages create a problem: if a page has no outlinks, then the random surfer gets stuck. A second problem arises from closed cycles: even if every page has an outgoing link, the surfer may become trapped in part of the web and never reach the rest.

Let

$$\mathbf{1} = (1, 1, \ldots, 1)^T,$$

and let $\boldsymbol{d}$ be the indicator vector of dead-end pages:

$$d_j = \begin{cases} 1, & \text{if page } j \text{ has no outlinks,} \\ 0, & \text{otherwise.} \end{cases}$$

If there are n pages, then the dead-end correction is

$$\widetilde{\mathbf{P}} = \mathbf{P} + \frac{1}{n}\mathbf{1}\boldsymbol{d}^T.$$

If $0 \le \alpha \le 1$, the fully teleported transition matrix is

$$\mathbf{M} = \alpha \widetilde{\mathbf{P}} + \frac{1-\alpha}{n} \mathbf{1}\mathbf{1}^T.$$

The interpretation is simple. With probability α, the surfer follows a link, using the dead-end correction when necessary. With probability $1-\alpha$, the surfer jumps to a uniformly random page.

Definition 4.3 A matrix $\mathbf{A}$ is a **Markov matrix** if

$$0 \le \mathbf{A}_{ij} \le 1$$

for all i, j, and each column sums to 1. A vector $\boldsymbol{x}$ is a probability vector if

$$0 \le x_i \le 1$$

for all i and

$$\sum_i x_i = 1.$$

Proposition 4.4 If $\mathbf{A}$ is a Markov matrix and $\boldsymbol{x}$ is a probability vector, then $\mathbf{A}\boldsymbol{x}$ is again a probability vector.

Proof. Since $\mathbf{A}_{ij} \ge 0$ and $x_j \ge 0$, each entry of $\mathbf{A}\boldsymbol{x}$ is nonnegative. Also,

$$\sum_i (\mathbf{A}\boldsymbol{x})_i = \sum_i \sum_j \mathbf{A}_{ij} x_j = \sum_j x_j \sum_i \mathbf{A}_{ij} = \sum_j x_j = 1.$$

□

Let x_k denote the importance of page k. If $\boldsymbol{x}$ is the probability vector describing the fraction of surfers on each page, then after one click the distribution becomes

$$\boldsymbol{x}^{(m+1)} = \mathbf{M}\boldsymbol{x}^{(m)}.$$

The steady-state ranking is therefore a fixed point satisfying

$$\boldsymbol{x} = \mathbf{M}\boldsymbol{x},$$

or equivalently

$$(\mathbf{I} - \mathbf{M})\boldsymbol{x} = \mathbf{0}.$$

This is an eigenvector problem with eigenvalue 1.

Recall that if $\mathbf{A}$ is a square matrix, then a nonzero vector $\boldsymbol{x}$ is an eigenvector of $\mathbf{A}$ with eigenvalue λ if

$$\mathbf{A}\boldsymbol{x} = \lambda\boldsymbol{x}.$$

Nonzero eigenvectors exist only when

$$\det(\lambda\mathbf{I} - \mathbf{A}) = 0.$$

For PageRank, we want the eigenvector of $\mathbf{M}$ associated with $\lambda = 1$. Since the matrix is enormous in practice, direct matrix factorizations are not realistic. Instead, we use an iterative method.

Theorem 4.5 Suppose $\mathbf{B}$ is diagonalizable, with eigenvalues

$$|\lambda_1| > |\lambda_2| \geq \cdots \geq |\lambda_n|,$$

and let $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n$ be an eigenvector basis. If the initial vector $\boldsymbol{x}^{(0)}$ has a nonzero component in the $\boldsymbol{v}_1$ direction, then repeated multiplication by $\mathbf{B}$ tends to align the iterates with $\boldsymbol{v}_1$, up to the scalar sign or phase of λ_1^m.

This is the basis of the **power method**. If

$$\boldsymbol{x}^{(0)} = c_1\boldsymbol{v}_1 + \cdots + c_n\boldsymbol{v}_n$$

with $c_1 \neq 0$, then

$$\mathbf{B}^m\boldsymbol{x}^{(0)} = c_1\lambda_1^m\boldsymbol{v}_1 + \cdots + c_n\lambda_n^m\boldsymbol{v}_n.$$

After factoring out λ_1^m, the remaining ratios $(\lambda_j/\lambda_1)^m$ tend to zero for $j \geq 2$. In actual computation we normalize the vector after each multiplication, because the direction is what matters and the length may grow or decay rapidly.

Theorem 4.6 (Perron–Frobenius theorem) If $\mathbf{A}$ is a Markov matrix, then 1 is an eigenvalue of $\mathbf{A}$, and there is a corresponding nonnegative eigenvector. If all entries of $\mathbf{A}$ are positive, then the eigenvalue 1 is simple, every other eigenvalue satisfies $|\lambda| < 1$, and the corresponding eigenvector is strictly positive.

For the PageRank matrix $\mathbf{M}$, the Perron–Frobenius theorem is relevant: because all entries of $\mathbf{M}$ are positive when $\alpha \neq 1$, the eigenvalue 1 is dominant and the remaining eigenvalues satisfy

$|\lambda| < 1$. This justifies the iteration

$$\boldsymbol{x}^{(m+1)} = \mathbf{M}\boldsymbol{x}^{(m)}.$$

In practice, we do not multiply by the dense matrix $\mathbf{M}$ directly. Instead,

$$\mathbf{M}\boldsymbol{x} = \alpha\mathbf{P}\boldsymbol{x} + \alpha\frac{\boldsymbol{d}^T\boldsymbol{x}}{n}\mathbf{1} + (1-\alpha)\frac{\mathbf{1}^T\boldsymbol{x}}{n}\mathbf{1},$$

and $\mathbf{P}$ is sparse, so the multiplication can be organized efficiently. A sparse matrix is stored by recording only its nonzero entries, for example through equal-length lists of row indices, column indices, and values. This is essential for PageRank, since the number of possible entries is enormous while the number of links from any one page is comparatively small.

Linear Systems

The basic problem is to solve a linear system

$$\mathbf{A}\boldsymbol{x} = \boldsymbol{b},$$

where

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}, \qquad \boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}, \qquad \text{and} \qquad \boldsymbol{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix}.$$

If $m < n$, the system is underdetermined. If $m > n$, it is overdetermined. The square case with a unique solution is the starting point.

There are several ways to solve $\mathbf{A}\boldsymbol{x} = \boldsymbol{b}$:

$$\boldsymbol{x} = \mathbf{A}^{-1}\boldsymbol{b} \qquad \text{and} \qquad \boldsymbol{x} = \mathbf{A}\backslash\boldsymbol{b},$$

or by factorizing $\mathbf{A}$ into simpler pieces such as

$$\mathbf{A} = \mathbf{LU} \qquad \text{or} \qquad \mathbf{A} = \mathbf{QR}.$$

Iterative methods form another major class of techniques. In these, we start with a guess $\boldsymbol{x}^{(0)}$ and produce a sequence of approximations $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots$ that (hopefully) converge to the solution.

Lower and upper triangular systems are especially easy to solve.

Definition 4.7 A square matrix $\mathbf{L}$ is **lower triangular** if

$$\mathbf{L}_{ij} = 0 \quad \text{whenever } i < j.$$

A square matrix $\mathbf{U}$ is **upper triangular** if

$$\mathbf{U}_{ij} = 0 \quad \text{whenever } i > j.$$

Proposition 4.8 If $\mathbf{L}\boldsymbol{x} = \boldsymbol{b}$ with $\mathbf{L}$ lower triangular and all diagonal entries nonzero, then the solution is determined by

$$x_1 = \frac{b_1}{\ell_{11}},$$

and for $k = 2, \ldots, n$,

$$x_k = \frac{1}{\ell_{kk}} \left(b_k - \sum_{j=1}^{k-1} \ell_{kj} x_j \right).$$

Proof. The first equation is

$$\ell_{11} x_1 = b_1,$$

so $x_1 = b_1/\ell_{11}$. The kth equation has the form

$$\ell_{k1} x_1 + \cdots + \ell_{k,k-1} x_{k-1} + \ell_{kk} x_k = b_k.$$

Since $x_1, \ldots, x_{k-1}$ are already known, we solve for x_k directly. □

Proposition 4.9 If $\mathbf{U}\boldsymbol{x} = \boldsymbol{b}$ with $\mathbf{U}$ upper triangular and all diagonal entries nonzero, then

$$x_n = \frac{b_n}{u_{nn}},$$

and for $k = n-1, n-2, \ldots, 1$,

$$x_k = \frac{1}{u_{kk}} \left(b_k - \sum_{j=k+1}^{n} u_{kj} x_j \right).$$

Proof. The proof is the same idea as forward substitution, but we start from the last row and

move upward. □

Both forward and backward substitution require on the order of n^2 floating-point operations.

Gaussian Elimination and LU Factorization

Since triangular systems are easy to solve, the key idea is to transform a general matrix into triangular form by eliminating entries below the diagonal. This is **Gaussian elimination**.

Example 4.10 Consider

$$\mathbf{A} = \begin{pmatrix} 2 & 2 & 3 \\ 4 & 6 & 5 \\ 6 & 7 & 8 \end{pmatrix} \qquad \text{and} \qquad \boldsymbol{b} = \begin{pmatrix} 1 \\ 3 \\ 0 \end{pmatrix}.$$

Subtracting 2 times row 1 from row 2 and 3 times row 1 from row 3 eliminates the entries below the first pivot. Repeating this in the second column produces an upper triangular system.

If the multipliers are stored systematically, the elimination process becomes a matrix factorization.

Proposition 4.11 Suppose Gaussian elimination can be performed on $\mathbf{A}$ without row exchanges. Then

$$\mathbf{A} = \mathbf{LU},$$

where $\mathbf{L}$ is unit lower triangular and $\mathbf{U}$ is upper triangular.

Proof. At the first elimination step, define the multipliers

$$m_{j1} = \frac{a_{j1}}{a_{11}}, \quad j = 2, \dots, n,$$

and form the multiplier matrix

$$\mathbf{M}_1 = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ -m_{21} & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ -m_{n1} & 0 & \cdots & 1 \end{pmatrix}.$$

Then $\mathbf{M}_1\mathbf{A}$ has zeros below the first pivot. Repeating this construction gives matrices $\mathbf{M}_1, \dots, \mathbf{M}_{n-1}$ such that

$$\mathbf{M}_{n-1} \cdots \mathbf{M}_2 \mathbf{M}_1 \mathbf{A} = \mathbf{U}.$$

Hence

$$\mathbf{A} = (\mathbf{M}_1^{-1} \mathbf{M}_2^{-1} \cdots \mathbf{M}_{n-1}^{-1}) \mathbf{U}.$$

Each inverse $\mathbf{M}_k^{-1}$ is unit lower triangular, and the product of unit lower triangular matrices is again unit lower triangular. Therefore

$$\mathbf{L} = \mathbf{M}_1^{-1} \mathbf{M}_2^{-1} \cdots \mathbf{M}_{n-1}^{-1}$$

is unit lower triangular, so $\mathbf{A} = \mathbf{L}\mathbf{U}$. □

Remark 4.12 Once the factorization $\mathbf{A} = \mathbf{L}\mathbf{U}$ is known, every new right-hand side can be solved by first computing $\boldsymbol{y}$ from

$$\mathbf{L}\boldsymbol{y} = \boldsymbol{b}$$

by forward substitution, and then computing $\boldsymbol{x}$ from

$$\mathbf{U}\boldsymbol{x} = \boldsymbol{y}$$

by backward substitution.

An efficient implementation overwrites the original matrix with the entries of $\mathbf{L}$ below the diagonal and the entries of $\mathbf{U}$ on and above the diagonal. In MATLAB we may recover them as

`U = triu(A)` and `L = tril(A,-1) + eye(size(A))`.

The factorization step for a dense $n \times n$ matrix requires roughly $2n^3/3$ floating-point operations, while the two triangular solves require only $O(n^2)$ operations. This is why an already computed $\mathbf{L}\mathbf{U}$ factorization is valuable when many right-hand sides must be solved. Partial pivoting adds row searches and swaps, but it does not change the leading $O(n^3)$ cost.

Remark 4.13 At pivot step p in the usual overwrite implementation, the multiplier for row $r > p$ is

$$m_{rp} = \frac{u_{rp}}{u_{pp}},$$

and the entries to the right of the pivot column are updated by

$$u_{rc} \leftarrow u_{rc} - m_{rp}u_{pc}, \quad c = p+1, \ldots, n.$$

The number m_{rp} is then stored in the (r,p)-entry, where it becomes part of $\mathbf{L}$. Thus the computed matrix stores $\mathbf{U}$ on and above the diagonal and the strictly lower triangular part of $\mathbf{L}$ below the diagonal.

Pivoting and Stability

If a pivot is zero, elimination breaks down. Even if the pivot is only very small, the resulting multipliers can become very large, which leads to severe roundoff problems.

Example 4.14 Consider the perturbed system

$$10^{-4}x_1 + x_2 = 1 \qquad \text{and} \qquad x_1 + x_2 = 2.$$

The exact solution is

$$x_2 = \frac{9998}{9999} \approx 0.99989998 \qquad \text{and} \qquad x_1 = \frac{10000}{9999} \approx 1.00010001.$$

This is a small perturbation of the system

$$x_2 = 1 \qquad \text{and} \qquad x_1 + x_2 = 2,$$

so the problem itself is well-conditioned.

If elimination is performed without pivoting, the first pivot is 10^{-4} and the multiplier is 10^4. In the floating-point system $\mathrm{FL}(10,3,1)$ with rounding, the eliminated second equation becomes

$$\mathrm{fl}(-9999)x_2 = \mathrm{fl}(-9998),$$

or

$$-10^4 x_2 = -10^4.$$

Thus $x_2 = 1$, and substitution into the first equation gives $x_1 = 0$, so the first component has relative error 100%. By switching the rows first, the pivot becomes 1 instead, the multiplier is only 10^{-4}, and the computation is stable.

Definition 4.15 At elimination step k, partial pivoting chooses as the pivot the entry of largest absolute value in column k among rows $k, k+1, \ldots, n$, and swaps that row into position k before performing the elimination.

Proposition 4.16 Gaussian elimination with partial pivoting produces permutation matrices and multipliers such that

$$\mathbf{PA} = \mathbf{LU},$$

where $\mathbf{P}$ is a permutation matrix, $\mathbf{L}$ is unit lower triangular, and $\mathbf{U}$ is upper triangular.

Proof. Maintain the invariant

$$\mathbf{P}^{(k)}\mathbf{A} = \mathbf{L}^{(k)}\mathbf{U}^{(k)}$$

after the first k pivot columns have been eliminated. At step $k+1$, the pivot search permutes rows $k+1, \ldots, n$ of $\mathbf{U}^{(k)}$. Applying the same permutation to the already stored entries in the first k columns of $\mathbf{L}^{(k)}$ preserves the invariant. The new elimination multipliers are then stored in column $k+1$ of $\mathbf{L}^{(k+1)}$, while the corresponding row operations update $\mathbf{U}^{(k+1)}$. Thus $\mathbf{L}^{(k+1)}$ remains unit lower triangular and the first $k+1$ columns of $\mathbf{U}^{(k+1)}$ have the required zeros. Induction through the last pivot gives $\mathbf{PA} = \mathbf{LU}$. □

Example 4.17 Consider

$$\mathbf{A} = \begin{pmatrix} 2 & 1 & 1 & 0 \\ 4 & 3 & 3 & 1 \\ 8 & 7 & 9 & 5 \\ 6 & 7 & 9 & 8 \end{pmatrix}.$$

Partial pivoting first chooses the entry 8 in the first column and swaps rows 1 and 3. Hence

$$\mathbf{P}_1 = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

and

$$\mathbf{P}_1\mathbf{A} = \begin{pmatrix} 8 & 7 & 9 & 5 \\ 4 & 3 & 3 & 1 \\ 2 & 1 & 1 & 0 \\ 6 & 7 & 9 & 8 \end{pmatrix}.$$

Eliminating the entries below the first pivot gives

$$\mathbf{M}_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ -\frac{1}{2} & 1 & 0 & 0 \\ -\frac{1}{4} & 0 & 1 & 0 \\ -\frac{3}{4} & 0 & 0 & 1 \end{pmatrix},$$

so

$$\mathbf{M}_1\mathbf{P}_1\mathbf{A} = \begin{pmatrix} 8 & 7 & 9 & 5 \\ 0 & -\frac{1}{2} & -\frac{3}{2} & -\frac{3}{2} \\ 0 & -\frac{3}{4} & -\frac{5}{4} & -\frac{5}{4} \\ 0 & \frac{7}{4} & \frac{9}{4} & \frac{17}{4} \end{pmatrix}.$$

In the second column, the largest remaining entry in absolute value is 7/4, so rows 2 and 4 are swapped:

$$\mathbf{P}_2 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}.$$

Thus

$$\mathbf{P}_2\mathbf{M}_1\mathbf{P}_1\mathbf{A} = \begin{pmatrix} 8 & 7 & 9 & 5 \\ 0 & \frac{7}{4} & \frac{9}{4} & \frac{17}{4} \\ 0 & -\frac{3}{4} & -\frac{5}{4} & -\frac{5}{4} \\ 0 & -\frac{1}{2} & -\frac{3}{2} & -\frac{3}{2} \end{pmatrix}.$$

Eliminating the entries below the second pivot gives

$$\mathbf{M}_2 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & \frac{3}{7} & 1 & 0 \\ 0 & \frac{2}{7} & 0 & 1 \end{pmatrix},$$

and hence

$$\mathbf{M}_2\mathbf{P}_2\mathbf{M}_1\mathbf{P}_1\mathbf{A} = \begin{pmatrix} 8 & 7 & 9 & 5 \\ 0 & \frac{7}{4} & \frac{9}{4} & \frac{17}{4} \\ 0 & 0 & -\frac{2}{7} & \frac{4}{7} \\ 0 & 0 & -\frac{6}{7} & -\frac{2}{7} \end{pmatrix}.$$

In the third column, the largest remaining entry in absolute value is $-6/7$, so rows 3 and 4 are swapped:

$$\mathbf{P}_3 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}.$$

After this swap,

$$\mathbf{P}_3\mathbf{M}_2\mathbf{P}_2\mathbf{M}_1\mathbf{P}_1\mathbf{A} = \begin{pmatrix} 8 & 7 & 9 & 5 \\ 0 & \frac{7}{4} & \frac{9}{4} & \frac{17}{4} \\ 0 & 0 & -\frac{6}{7} & -\frac{2}{7} \\ 0 & 0 & -\frac{2}{7} & \frac{4}{7} \end{pmatrix}.$$

The last elimination matrix is

$$\mathbf{M}_3 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -\frac{1}{3} & 1 \end{pmatrix},$$

and therefore

$$\mathbf{U} = \mathbf{M}_3\mathbf{P}_3\mathbf{M}_2\mathbf{P}_2\mathbf{M}_1\mathbf{P}_1\mathbf{A} = \begin{pmatrix} 8 & 7 & 9 & 5 \\ 0 & \frac{7}{4} & \frac{9}{4} & \frac{17}{4} \\ 0 & 0 & -\frac{6}{7} & -\frac{2}{7} \\ 0 & 0 & 0 & \frac{2}{3} \end{pmatrix}.$$

The final row order is $3, 4, 2, 1$, so

$$\mathbf{P} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}.$$

The corresponding unit lower triangular factor is

$$\mathbf{L} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ \frac{3}{4} & 1 & 0 & 0 \\ \frac{1}{2} & -\frac{2}{7} & 1 & 0 \\ \frac{1}{4} & -\frac{3}{7} & \frac{1}{3} & 1 \end{pmatrix},$$

and these matrices satisfy

$$\mathbf{PA} = \mathbf{LU}.$$

Remark 4.18 When partial pivoting gives $\mathbf{PA} = \mathbf{LU}$, the system $\mathbf{A}\boldsymbol{x} = \boldsymbol{b}$ is solved by first applying the same row permutation to the right-hand side:

$$\mathbf{PA}\boldsymbol{x} = \mathbf{P}\boldsymbol{b}.$$

Since $\mathbf{PA} = \mathbf{LU}$, this becomes

$$\mathbf{LU}\boldsymbol{x} = \mathbf{P}\boldsymbol{b}.$$

Thus we solve

$$\mathbf{L}\boldsymbol{y} = \mathbf{P}\boldsymbol{b}$$

by forward substitution and then solve

$$\mathbf{U}\boldsymbol{x} = \boldsymbol{y}$$

by backward substitution.

Example 4.19 Partial pivoting does not fix every scaling problem. Consider

$$2x_1 + 20000x_2 = 20000 \qquad \text{and} \qquad x_1 + x_2 = 2.$$

The exact solution is

$$x_1 = \frac{10000}{9999} \qquad \text{and} \qquad x_2 = \frac{9998}{9999}.$$

No row exchange is triggered by partial pivoting in the first column. Eliminating x_1 gives

$$(1 - 10^4)x_2 = 2 - 10^4.$$

In the floating-point system $\mathrm{FL}(10, 3, 1)$ with rounding, this again becomes

$$-10^4 x_2 = -10^4,$$

so $x_2 = 1$ and then $x_1 = 0$.

If the columns are swapped first, then the first equation is

$$20000x_2 + 2x_1 = 20000.$$

Eliminating x_2 from the second equation gives

$$\left(1 - \frac{2}{20000}\right) x_1 = 2 - \frac{20000}{20000},$$

so $x_1 \approx 1$ and then $x_2 \approx 19998/20000$. This illustrates the reason behind complete pivoting: the largest available entry in the remaining coefficient matrix is used as the next pivot.

Remark 4.20 Complete pivoting searches the entire remaining submatrix for the largest available pivot and therefore introduces both row and column permutations. This leads to a factorization

$$\mathbf{PAQ} = \mathbf{LU}.$$

To solve $\mathbf{A}\boldsymbol{x} = \boldsymbol{b}$ from this factorization, set $\boldsymbol{z} = \mathbf{Q}^T\boldsymbol{x}$, solve

$$\mathbf{LU}\boldsymbol{z} = \mathbf{P}\boldsymbol{b},$$

and then recover $\boldsymbol{x} = \mathbf{Q}\boldsymbol{z}$. Complete pivoting is rarely worth the extra work in practice, since partial pivoting is usually sufficient.

Conditioning of Linear Systems

The sensitivity of the problem $\mathbf{A}\boldsymbol{x} = \boldsymbol{b}$ depends primarily on the matrix $\mathbf{A}$.

Definition 4.21 For a nonsingular matrix $\mathbf{A}$ and a natural matrix norm $\|\cdot\|$, the **condition number** is

$$\kappa(\mathbf{A}) = \|\mathbf{A}\| \, \|\mathbf{A}^{-1}\|.$$

Proposition 4.22 Suppose

$$\mathbf{A}(\boldsymbol{x} + \Delta\boldsymbol{x}) = \boldsymbol{b} + \Delta\boldsymbol{b},$$

where $\mathbf{A}$ is nonsingular and $\boldsymbol{b} \neq \mathbf{0}$. Then

$$\frac{\|\Delta\boldsymbol{x}\|}{\|\boldsymbol{x}\|} \leq \kappa(\mathbf{A}) \frac{\|\Delta\boldsymbol{b}\|}{\|\boldsymbol{b}\|}.$$

Proof. Subtracting $\mathbf{A}\boldsymbol{x} = \boldsymbol{b}$ from

$$\mathbf{A}(\boldsymbol{x} + \Delta\boldsymbol{x}) = \boldsymbol{b} + \Delta\boldsymbol{b}$$

gives

$$\mathbf{A}\Delta\boldsymbol{x} = \Delta\boldsymbol{b}.$$

Hence

$$\Delta\boldsymbol{x} = \mathbf{A}^{-1}\Delta\boldsymbol{b},$$

so

$$\|\Delta\boldsymbol{x}\| \leq \|\mathbf{A}^{-1}\| \, \|\Delta\boldsymbol{b}\|.$$

Also,

$$\boldsymbol{b} = \mathbf{A}\boldsymbol{x},$$

so

$$\|\boldsymbol{b}\| \leq \|\mathbf{A}\| \, \|\boldsymbol{x}\|, \qquad \text{which implies} \qquad \|\boldsymbol{x}\| \geq \frac{\|\boldsymbol{b}\|}{\|\mathbf{A}\|}.$$

Dividing the first inequality by this lower bound for $\|\boldsymbol{x}\|$ gives

$$\frac{\|\Delta\boldsymbol{x}\|}{\|\boldsymbol{x}\|} \leq \|\mathbf{A}^{-1}\| \, \|\Delta\boldsymbol{b}\| \cdot \frac{\|\mathbf{A}\|}{\|\boldsymbol{b}\|} = \kappa(\mathbf{A}) \frac{\|\Delta\boldsymbol{b}\|}{\|\boldsymbol{b}\|}.$$

□

Definition 4.23 The **matrix 1-norm**, **2-norm**, and **∞-norm** are

$$\|\mathbf{A}\|_1 = \max_j \sum_i |a_{ij}|,$$

$$\|\mathbf{A}\|_2 = \sqrt{\lambda_{\max}(\mathbf{A}^T\mathbf{A})},$$

and

$$\|\mathbf{A}\|_\infty = \max_i \sum_j |a_{ij}|.$$

Example 4.24 Let

$$\mathbf{A} = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 1 & 3 \\ 3 & 2 & 1 \end{pmatrix}.$$

Its inverse is

$$\mathbf{A}^{-1} = \begin{pmatrix} -\frac{5}{12} & \frac{1}{3} & \frac{1}{4} \\ \frac{7}{12} & -\frac{2}{3} & \frac{1}{4} \\ \frac{1}{12} & \frac{1}{3} & -\frac{1}{4} \end{pmatrix}.$$

Therefore

$$\|\mathbf{A}\|_1 = 7, \qquad \|\mathbf{A}^{-1}\|_1 = \frac{4}{3}, \qquad \text{and} \qquad \kappa_1(\mathbf{A}) = \frac{28}{3} \approx 9.33.$$

Similarly,

$$\|\mathbf{A}\|_\infty = 6, \qquad \|\mathbf{A}^{-1}\|_\infty = \frac{3}{2}, \qquad \text{and} \qquad \kappa_\infty(\mathbf{A}) = 9.$$

Also,

$$\mathbf{A}^T\mathbf{A} = \begin{pmatrix} 14 & 10 & 12 \\ 10 & 9 & 11 \\ 12 & 11 & 19 \end{pmatrix}$$

has largest eigenvalue approximately 36.71 and smallest eigenvalue approximately 0.8927, so

$$\|\mathbf{A}\|_2 = \sqrt{36.71} \approx 6.06 \qquad \text{and} \qquad \|\mathbf{A}^{-1}\|_2 = \frac{1}{\sqrt{0.8927}} \approx 1.06,$$

and

$$\kappa_2(\mathbf{A}) \approx 6.42.$$

This is a well-conditioned system.

Example 4.25 The **Hilbert matrix**

$$\mathbf{H} = \begin{pmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} \\ \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} \end{pmatrix}$$

has

$$\kappa_2(\mathbf{H}) \approx 15513.74.$$

Thus $\mathbf{H}\boldsymbol{x} = \boldsymbol{b}$ is generally an ill-conditioned problem: even a stable algorithm may produce a solution with noticeable relative error if the data are perturbed.

If $\widehat{\boldsymbol{x}}$ is a computed solution, then two different quantities matter:

$$\text{residual error} = \boldsymbol{b} - \mathbf{A}\widehat{\boldsymbol{x}},$$

and

$$\text{relative solution error} = \frac{\|\boldsymbol{x} - \widehat{\boldsymbol{x}}\|}{\|\boldsymbol{x}\|}.$$

These are not the same thing! A small residual does not automatically imply a small relative solution error, especially if the problem is ill-conditioned.

Example 4.26 Consider

$$\mathbf{A} = \begin{pmatrix} 0.780 & 0.563 \\ 0.913 & 0.659 \end{pmatrix} \qquad \text{and} \qquad \boldsymbol{x} = \begin{pmatrix} 1 \\ -1 \end{pmatrix}.$$

Then

$$\boldsymbol{b} = \mathbf{A}\boldsymbol{x} = \begin{pmatrix} 0.217 \\ 0.254 \end{pmatrix}.$$

Now compare the two approximations

$$\widehat{\boldsymbol{x}}_1 = \begin{pmatrix} 0.341 \\ -0.087 \end{pmatrix} \qquad \text{and} \qquad \widehat{\boldsymbol{x}}_2 = \begin{pmatrix} 0.999 \\ -1.00 \end{pmatrix}.$$

The residuals are

$$\|\boldsymbol{b} - \mathbf{A}\widehat{\boldsymbol{x}}_1\|_1 \approx 1.0 \times 10^{-6} \qquad \text{and} \qquad \|\boldsymbol{b} - \mathbf{A}\widehat{\boldsymbol{x}}_2\|_1 \approx 1.693 \times 10^{-3},$$

so $\widehat{\boldsymbol{x}}_1$ appears much better if we only look at the residual. However, the relative solution errors behave in the opposite way:

$$\frac{\|\boldsymbol{x} - \widehat{\boldsymbol{x}}_1\|_2}{\|\boldsymbol{x}\|_2} \approx 0.8 \qquad \text{and} \qquad \frac{\|\boldsymbol{x} - \widehat{\boldsymbol{x}}_2\|_2}{\|\boldsymbol{x}\|_2} \approx 7.1 \times 10^{-4}.$$

Thus a small residual does not automatically imply a good approximation to the true solution.

For large sparse systems, iterative methods are often preferable to direct factorizations.

Definition 4.27 Write the equations of $\mathbf{A}\boldsymbol{x} = \boldsymbol{b}$ as

$$\sum_{j=1}^{n} a_{ij} x_j = b_i.$$

Assuming $a_{ii} \neq 0$, the **Jacobi iteration** is

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} \right).$$

Definition 4.28 The **Gauss–Seidel method** uses the newest available values:

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} \right).$$

Typical stopping rules include performing a fixed number of iterations, requiring the residual norm

$$\|\boldsymbol{b} - \mathbf{A}\boldsymbol{x}^{(k)}\|$$

to fall below a tolerance, or requiring consecutive iterates to satisfy

$$\|\boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)}\|$$

below a tolerance.

Definition 4.29 A square matrix $\mathbf{A}$ is **strictly diagonally dominant** if for every i,

$$|a_{ii}| > \sum_{j \neq i} |a_{ij}|.$$

Theorem 4.30 If $\mathbf{A}$ is **strictly diagonally dominant**, then for any initial guess $\boldsymbol{x}^{(0)}$, both the Jacobi iteration and the Gauss–Seidel iteration converge to the unique solution of $\mathbf{A}\boldsymbol{x} = \boldsymbol{b}$.

Proof. Strict diagonal dominance implies in particular that each diagonal entry is nonzero, so both iterations are well-defined. It also implies that $\mathbf{A}$ is nonsingular, so the system has a unique solution $\boldsymbol{x}$. For the Jacobi method, let $\boldsymbol{e}^{(k)} = \boldsymbol{x} - \boldsymbol{x}^{(k)}$ be the error after k steps. Since $\boldsymbol{x}$ satisfies

$$x_i = \frac{1}{a_{ii}} \left(b_i - \sum_{j \neq i} a_{ij} x_j \right),$$

subtracting the Jacobi update from this identity gives

$$e_i^{(k+1)} = -\frac{1}{a_{ii}} \sum_{j \neq i} a_{ij} e_j^{(k)}.$$

Hence

$$\left| e_i^{(k+1)} \right| \leq \frac{1}{|a_{ii}|} \sum_{j \neq i} |a_{ij}| \, \|\boldsymbol{e}^{(k)}\|_\infty.$$

If

$$q = \max_{1 \leq i \leq n} \frac{1}{|a_{ii}|} \sum_{j \neq i} |a_{ij}|,$$

then strict diagonal dominance gives $0 \leq q < 1$, and therefore

$$\|\boldsymbol{e}^{(k+1)}\|_\infty \leq q \|\boldsymbol{e}^{(k)}\|_\infty.$$

Thus the Jacobi errors converge to 0 geometrically. For the Gauss–Seidel method, subtracting the iteration formula from the exact equation gives

$$e_i^{(k+1)} = -\frac{1}{a_{ii}} \left(\sum_{j=1}^{i-1} a_{ij} e_j^{(k+1)} + \sum_{j=i+1}^{n} a_{ij} e_j^{(k)} \right).$$

Define

$$r_i = \frac{1}{|a_{ii}|}\sum_{j=1}^{i-1}|a_{ij}| \qquad \text{and} \qquad s_i = \frac{1}{|a_{ii}|}\sum_{j=i+1}^{n}|a_{ij}|.$$

Then $r_i + s_i < 1$ for every i. Let

$$E_{k+1} = \|\boldsymbol{e}^{(k+1)}\|_\infty \qquad \text{and} \qquad E_k = \|\boldsymbol{e}^{(k)}\|_\infty,$$

and choose p such that $\left|e_p^{(k+1)}\right| = E_{k+1}$. Using the error equation for $i = p$ gives

$$E_{k+1} \le r_p E_{k+1} + s_p E_k,$$

so

$$E_{k+1} \le \frac{s_p}{1 - r_p} E_k.$$

Since $s_i < 1 - r_i$, each factor $s_i/(1 - r_i)$ is strictly less than 1. If

$$\widetilde{q} = \max_{1 \le i \le n} \frac{s_i}{1 - r_i},$$

then $0 \le \widetilde{q} < 1$, and hence

$$\|\boldsymbol{e}^{(k+1)}\|_\infty \le \widetilde{q}\|\boldsymbol{e}^{(k)}\|_\infty.$$

Thus the Gauss–Seidel iteration also converges to $\boldsymbol{x}$. □

Least Squares and QR Factorization

If $m > n$, then the system $\mathbf{A}\boldsymbol{x} = \boldsymbol{b}$ is overdetermined. There may be no exact solution, one exact solution, or infinitely many exact solutions. A standard alternative is to choose $\boldsymbol{x}$ to minimize the residual norm:

$$\min_{\boldsymbol{x}} \|\boldsymbol{b} - \mathbf{A}\boldsymbol{x}\|_2^2.$$

This is the **linear least squares problem**.

Example 4.31 Each of the following systems is overdetermined. The system

$$x_1 + x_2 = 2, \qquad x_1 - x_2 = 0, \qquad \text{and} \qquad x_1 + 2x_2 = 5$$

has no solution, because the first two equations imply $x_1 = x_2 = 1$, which does not satisfy the third equation. The system

$$x_1 + x_2 = 2, \qquad x_1 - x_2 = 0, \qquad \text{and} \qquad x_1 + 2x_2 = 3$$

has the unique solution $x_1 = x_2 = 1$. The system

$$x_1 + x_2 = 2, \qquad 2x_1 + 2x_2 = 4, \qquad \text{and} \qquad 3x_1 + 3x_2 = 6$$

has infinitely many solutions, since all three equations reduce to the single condition $x_1 + x_2 = 2$.

Theorem 4.32 If $\mathbf{A}$ has full column rank, then the linear least-squares problem has a unique minimizer, and it is the unique solution of the normal equations

$$\mathbf{A}^T\mathbf{A}\boldsymbol{x} = \mathbf{A}^T\boldsymbol{b}.$$

Proof. Define

$$\phi(\boldsymbol{x}) = \|\boldsymbol{b} - \mathbf{A}\boldsymbol{x}\|_2^2 = (\boldsymbol{b} - \mathbf{A}\boldsymbol{x})^T(\boldsymbol{b} - \mathbf{A}\boldsymbol{x}).$$

Expanding gives

$$\phi(\boldsymbol{x}) = \boldsymbol{b}^T\boldsymbol{b} - 2\boldsymbol{x}^T\mathbf{A}^T\boldsymbol{b} + \boldsymbol{x}^T\mathbf{A}^T\mathbf{A}\boldsymbol{x}.$$

Differentiating with respect to $\boldsymbol{x}$ shows that every minimizer must satisfy

$$-2\mathbf{A}^T\boldsymbol{b} + 2\mathbf{A}^T\mathbf{A}\boldsymbol{x} = 0,$$

so

$$\mathbf{A}^T\mathbf{A}\boldsymbol{x} = \mathbf{A}^T\boldsymbol{b}.$$

Because $\mathbf{A}$ has full column rank, $\mathbf{A}^T\mathbf{A}$ is positive definite. Hence ϕ is strictly convex, the normal equations have a unique solution, and that stationary point is the unique global minimizer. □

Example 4.33 Let

$$\mathbf{A} = \begin{pmatrix} 1 & 1 & 1 \\ 2 & 1 & 2 \\ 3 & 2 & 0 \\ 2 & -2 & -1 \\ -1 & 1 & 2 \end{pmatrix} \quad \text{and} \quad \boldsymbol{b} = \begin{pmatrix} 5 \\ 4 \\ 5 \\ 0 \\ 3 \end{pmatrix}.$$

Then

$$\mathbf{A}^T\mathbf{A} = \begin{pmatrix} 19 & 4 & 1 \\ 4 & 11 & 7 \\ 1 & 7 & 10 \end{pmatrix} \quad \text{and} \quad \mathbf{A}^T\boldsymbol{b} = \begin{pmatrix} 25 \\ 22 \\ 19 \end{pmatrix}.$$

Solving the normal equations produces the least squares solution

$$\boldsymbol{x} \approx \begin{pmatrix} 1.0747 \\ 0.8448 \\ 1.2011 \end{pmatrix}.$$

If $\mathbf{A}$ has full column rank, then $\mathbf{A}^T\mathbf{A}$ is symmetric positive definite, so the normal equations can be solved with a Cholesky factorization.

Definition 4.34 If $\mathbf{M}$ is symmetric positive definite, then there exists a lower triangular matrix $\mathbf{L}$ with positive diagonal entries such that

$$\mathbf{M} = \mathbf{L}\mathbf{L}^T.$$

Proposition 4.35 Let $\mathbf{M} = (m_{ij})$ be symmetric positive definite, and suppose

$$\mathbf{M} = \mathbf{L}\mathbf{L}^T$$

with $\mathbf{L} = (\ell_{ij})$ lower triangular. Then for $k = 1, \ldots, n$,

$$\ell_{kk} = \sqrt{m_{kk} - \sum_{j=1}^{k-1} \ell_{kj}^2},$$

and for $i = k+1, \ldots, n$,

$$\ell_{ik} = \frac{m_{ik} - \sum_{j=1}^{k-1} \ell_{ij}\ell_{kj}}{\ell_{kk}}.$$

Proof. Comparing entries of $\mathbf{L}\mathbf{L}^T = \mathbf{M}$ gives

$$m_{kk} = \ell_{k1}^2 + \cdots + \ell_{k,k-1}^2 + \ell_{kk}^2,$$

which yields the diagonal formula. For $i > k$,

$$m_{ik} = \ell_{i1}\ell_{k1} + \cdots + \ell_{i,k-1}\ell_{k,k-1} + \ell_{ik}\ell_{kk},$$

and solving for ℓ_{ik} gives the second formula. □

Example 4.36 For

$$\mathbf{A}^T\mathbf{A} = \begin{pmatrix} 19 & 4 & 1 \\ 4 & 11 & 7 \\ 1 & 7 & 10 \end{pmatrix},$$

the Cholesky factor is

$$\mathbf{L} = \begin{pmatrix} 4.3589 & 0 & 0 \\ 0.9177 & 3.1871 & 0 \\ 0.2294 & 2.1303 & 2.3258 \end{pmatrix}.$$

Solving

$$\mathbf{L}\boldsymbol{y} = \mathbf{A}^T\boldsymbol{b}$$

by forward substitution gives

$$\boldsymbol{y} \approx \begin{pmatrix} 5.7354 \\ 5.2514 \\ 2.7936 \end{pmatrix},$$

and solving

$$\mathbf{L}^T\boldsymbol{x} = \boldsymbol{y}$$

by backward substitution gives again

$$\boldsymbol{x} \approx \begin{pmatrix} 1.0747 \\ 0.8448 \\ 1.2011 \end{pmatrix}.$$

The Cholesky-based least squares solve requires about

$$mn^2 + \frac{n^3}{3}$$

floating-point operations after the normal equations are formed.

MATLAB's `chol` command returns an upper triangular factor $\mathbf{R}$ satisfying $\mathbf{M} = \mathbf{R}^T\mathbf{R}$ by default. The command `chol(M,'lower')` returns the lower triangular factor $\mathbf{L}$ instead.

An alternative that is usually more stable is the **QR factorization**.

Definition 4.37 If $\mathbf{A}$ is an $m \times n$ matrix with full column rank, then there exist an $m \times n$ matrix $\mathbf{Q}$ with orthonormal columns and an $n \times n$ upper triangular matrix $\mathbf{R}$ with positive diagonal entries such that

$$\mathbf{A} = \mathbf{QR}.$$

If the columns of $\mathbf{A}$ are $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n$ and the columns of $\mathbf{Q}$ are $\boldsymbol{q}_1, \ldots, \boldsymbol{q}_n$, then

$$\boldsymbol{a}_k = \sum_{j=1}^{k} r_{jk}\boldsymbol{q}_j.$$

This leads to the classical Gram–Schmidt construction.

Proposition 4.38 (Gram–Schmidt QR factorization) Assume $\mathbf{A}$ has full column rank. Then

$$r_{11} = \|\boldsymbol{a}_1\| \qquad \text{and} \qquad \boldsymbol{q}_1 = \frac{\boldsymbol{a}_1}{r_{11}},$$

and for $k \geq 2$,

$$r_{jk} = \boldsymbol{q}_j^T\boldsymbol{a}_k, \quad 1 \leq j \leq k-1,$$

$$\widehat{\boldsymbol{a}}_k = \boldsymbol{a}_k - \sum_{j=1}^{k-1} r_{jk}\boldsymbol{q}_j, \qquad r_{kk} = \|\widehat{\boldsymbol{a}}_k\|, \qquad \text{and} \qquad \boldsymbol{q}_k = \frac{\widehat{\boldsymbol{a}}_k}{r_{kk}}.$$

Proof. The relation

$$\boldsymbol{a}_k = \sum_{j=1}^{k} r_{jk}\boldsymbol{q}_j$$

and the orthonormality of the columns of $\mathbf{Q}$ imply

$$\boldsymbol{q}_j^T\boldsymbol{a}_k = r_{jk} \quad (j < k).$$

Subtracting the already constructed components gives

$$\widehat{\boldsymbol{a}}_k = \boldsymbol{a}_k - \sum_{j=1}^{k-1} r_{jk}\boldsymbol{q}_j = r_{kk}\boldsymbol{q}_k.$$

Taking norms yields

$$r_{kk} = \|\widehat{\boldsymbol{a}}_k\|,$$

and normalizing gives $\boldsymbol{q}_k$. □

Example 4.39 For

$$\mathbf{A} = \begin{pmatrix} 1 & 1 & 1 \\ 2 & 1 & 2 \\ 3 & 2 & 0 \\ 2 & -2 & -1 \\ -1 & 1 & 2 \end{pmatrix},$$

the reduced QR factorization is approximately

$$\mathbf{Q} = \begin{pmatrix} 0.23 & 0.25 & 0.18 \\ 0.46 & 0.18 & 0.65 \\ 0.69 & 0.43 & -0.46 \\ 0.46 & -0.76 & 0.22 \\ -0.23 & 0.38 & 0.53 \end{pmatrix} \quad \text{and} \quad \mathbf{R} = \begin{pmatrix} 4.36 & 0.92 & 0.23 \\ 0 & 3.19 & 2.13 \\ 0 & 0 & 2.33 \end{pmatrix}.$$

Then

$$\mathbf{R}\boldsymbol{x} = \mathbf{Q}^T\boldsymbol{b}$$

becomes

$$\begin{pmatrix} 4.36 & 0.92 & 0.23 \\ 0 & 3.19 & 2.13 \\ 0 & 0 & 2.33 \end{pmatrix} \boldsymbol{x} = \begin{pmatrix} 5.74 \\ 5.25 \\ 2.79 \end{pmatrix},$$

and backward substitution again yields

$$\boldsymbol{x} \approx \begin{pmatrix} 1.07 \\ 0.84 \\ 1.20 \end{pmatrix}.$$

The Gram–Schmidt QR factorization requires roughly

$$2mn^2$$

floating-point operations.

Example 4.40 Consider the overdetermined system

$$\begin{pmatrix} 1 & -1 \\ 0 & 10^{-5} \\ 0 & 0 \end{pmatrix} \boldsymbol{x} = \begin{pmatrix} 0 \\ 10^{-5} \\ 1 \end{pmatrix}.$$

The exact least squares solution is

$$x_1 = x_2 = 1.$$

The normal equations are

$$\begin{pmatrix} 1 & -1 \\ -1 & 1 + 10^{-10} \end{pmatrix} \boldsymbol{x} = \begin{pmatrix} 0 \\ 10^{-10} \end{pmatrix}.$$

In low precision, the matrix $\mathbf{A}^T\mathbf{A}$ rounds to

$$\begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix},$$

which is singular, so Cholesky fails.

By contrast, a QR factorization is

$$\mathbf{A} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} 1 & -1 \\ 0 & 10^{-5} \end{pmatrix},$$

and

$$\mathbf{Q}^T\boldsymbol{b} = \begin{pmatrix} 0 \\ 10^{-5} \end{pmatrix}.$$

Solving

$$\mathbf{R}\boldsymbol{x} = \mathbf{Q}^T\boldsymbol{b}$$

gives

$$x_1 = x_2 = 1.$$

Thus QR is the more stable method in this example, and usually in general.

Remark 4.41 Cholesky is usually faster when it applies, especially for large sparse matrices, but QR is usually preferred for general dense least squares problems because of its superior numerical stability.

The reduced factorization above is also called the **economy-size QR factorization**. The full QR factorization writes

$$\mathbf{A} = \begin{pmatrix} \mathbf{Q}_1 & \mathbf{Q}_2 \end{pmatrix} \begin{pmatrix} \mathbf{R}_1 \\ \mathbf{0} \end{pmatrix} = \mathbf{Q}_1 \mathbf{R}_1,$$

where $\mathbf{Q}_1$ has the same columns as the reduced factor $\mathbf{Q}$. Other standard ways to compute QR factors include **Householder transformations** and **Givens rotations**. In MATLAB, `qr(A,0)` computes the economy-size factorization, and pivoted QR can be requested by including a permutation output.

If $\text{rank}(\mathbf{A}) = k < n$, then the least squares solution is not unique. A useful approach is to permute the columns so that the important columns are handled first. This leads to a pivoted QR factorization of the form

$$\mathbf{AP} = \mathbf{QR},$$

where

$$\mathbf{R} = \begin{pmatrix} \mathbf{R}_{11} & \mathbf{S} \\ 0 & 0 \end{pmatrix},$$

with $\mathbf{R}_{11}$ a $k \times k$ upper triangular matrix. We then solve the reduced system for the first k variables and set the remaining ones to zero to obtain a particular least squares solution.

Singular Value Decomposition

Definition 4.42 If $\mathbf{A}$ is an $m \times n$ matrix of full column rank, then there exist matrices

$$\mathbf{U} \in \mathbb{R}^{m \times n}, \qquad \mathbf{\Sigma} \in \mathbb{R}^{n \times n}, \qquad \text{and} \qquad \mathbf{V} \in \mathbb{R}^{n \times n},$$

such that

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T.$$

Here the columns of $\mathbf{U}$ are orthonormal, $\mathbf{V}$ is orthogonal, and

$$\mathbf{\Sigma} = \begin{pmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_n \end{pmatrix}, \quad \sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n > 0.$$

The numbers σ_i are the **singular values** of $\mathbf{A}$.

The singular values are the square roots of the eigenvalues of $\mathbf{A}^T\mathbf{A}$. The columns of $\mathbf{V}$ are eigen-

vectors of $\mathbf{A}^T\mathbf{A}$, and the columns of $\mathbf{U}$ are the corresponding left singular vectors. In particular,

$$\|\mathbf{A}\|_2 = \sigma_1,$$

and, when $\mathbf{A}$ has full column rank,

$$\kappa_2(\mathbf{A}) = \frac{\sigma_1}{\sigma_n}.$$

Proposition 4.43 If

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$$

with full column rank, then the least squares solution is

$$\boldsymbol{x} = \mathbf{V}\mathbf{\Sigma}^{-1}\mathbf{U}^T\boldsymbol{b}.$$

Proof. The normal equations are

$$\mathbf{A}^T\mathbf{A}\boldsymbol{x} = \mathbf{A}^T\boldsymbol{b}.$$

Using the singular value decomposition,

$$\mathbf{A}^T\mathbf{A} = (\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T)^T(\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T) = \mathbf{V}\mathbf{\Sigma}^T\mathbf{U}^T\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T = \mathbf{V}\mathbf{\Sigma}^2\mathbf{V}^T.$$

Also,

$$\mathbf{A}^T\boldsymbol{b} = \mathbf{V}\mathbf{\Sigma}\mathbf{U}^T\boldsymbol{b}.$$

Thus the normal equations become

$$\mathbf{V}\mathbf{\Sigma}^2\mathbf{V}^T\boldsymbol{x} = \mathbf{V}\mathbf{\Sigma}\mathbf{U}^T\boldsymbol{b}.$$

Multiplying by $\mathbf{V}^T$ on the left and then by $\mathbf{\Sigma}^{-1}$ gives

$$\mathbf{\Sigma}\mathbf{V}^T\boldsymbol{x} = \mathbf{U}^T\boldsymbol{b},$$

so

$$\mathbf{V}^T\boldsymbol{x} = \mathbf{\Sigma}^{-1}\mathbf{U}^T\boldsymbol{b},$$

and therefore

$$\boldsymbol{x} = \mathbf{V}\mathbf{\Sigma}^{-1}\mathbf{U}^T\boldsymbol{b}.$$

□

The singular value decomposition has many uses beyond least squares. If we keep only the first k

singular values and the corresponding singular vectors, then

$$\mathbf{A}_k = \mathbf{U}_k \mathbf{\Sigma}_k \mathbf{V}_k^T$$

is a **low-rank approximation** of $\mathbf{A}$. This idea is central in **principal component analysis** (**PCA**), **latent semantic analysis** (**LSA**), and image compression.

A grayscale image can be viewed as a matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ whose entries record pixel intensities. Instead of storing all mn entries of $\mathbf{A}$, we may store a low-rank approximation

$$\mathbf{A}_k = \mathbf{U}_k \mathbf{\Sigma}_k \mathbf{V}_k^T,$$

where $\mathbf{U}_k$ contains the first k left singular vectors, $\mathbf{V}_k$ contains the first k right singular vectors, and

$$\mathbf{\Sigma}_k = \operatorname{diag}(\sigma_1, \ldots, \sigma_k).$$

When k is much smaller than $\min\{m, n\}$, this can greatly reduce storage while preserving the main visual features of the image.

Indeed, storing $\mathbf{A}$ directly requires mn numbers, whereas storing the truncated singular value decomposition requires

$$mk + k + nk = k(m + n + 1)$$

numbers. Compression is therefore achieved whenever

$$k(m + n + 1) < mn.$$

Large singular values capture the dominant structure of the image, while the smaller singular values often encode fine detail and noise. Keeping only the first few singular values therefore produces a blurred but recognizable approximation.

This image compression interpretation is shown in Figure 26. A very small value of k preserves only the broad intensity pattern, while larger values of k restore sharper edges and finer texture.

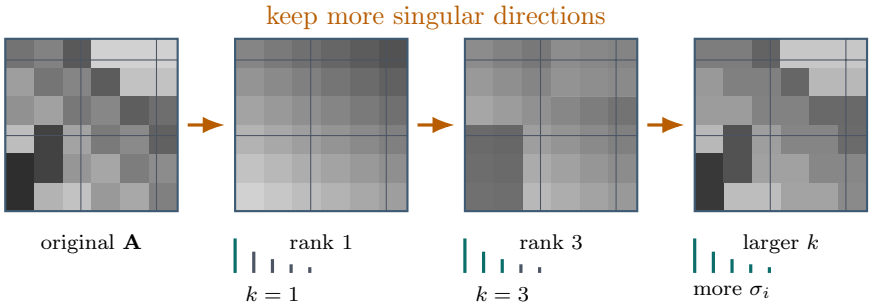


Figure 26

Lecture 5 — Thursday, March 10, 2016

5. Numerical Integration and Root Finding

Numerical Integration

Given a continuous function f on an interval $[a, b]$, the definite integral

$$I = \int_a^b f(x)\,\mathrm{d}x$$

measures the signed area under the graph of f. When an antiderivative F is known, the Fundamental Theorem of Calculus gives

$$I = F(b) - F(a).$$

In many applications, however, numerical integration is needed because f is known only through observations (x_k, f_k), because no closed form for an antiderivative F is available, or because evaluating an available antiderivative is more expensive than estimating the integral directly.

A first geometric approximation is obtained by placing a grid over the region under the curve. The completely enclosed grid squares give a lower bound, while all grid squares that meet the region give an upper bound. Refining the grid improves these bounds, but it is not an efficient way to compute accurate integrals.

The grid bounds in Figure 27 motivate more systematic approximations. Instead of counting squares, we sample f at selected points, interpolate those data by a simpler function, and integrate

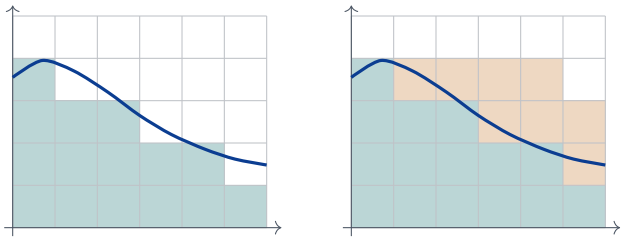


Figure 27

that simpler function exactly. This leads to the problem of numerical integration, also called numerical quadrature.

Definition 5.1 A **quadrature rule** on $[a, b]$ is an approximation of the form

$$\int_a^b f(x)\,\mathrm{d}x \approx \sum_{j=0}^{n} c_j f(x_j),$$

where the nodes $x_j \in [a, b]$ and the weights c_j are prescribed numbers.

One way to construct such rules is to interpolate f by a polynomial p and then integrate p exactly:

$$\int_a^b f(x)\,\mathrm{d}x \approx \int_a^b p(x)\,\mathrm{d}x.$$

If p is written in the Lagrange form for the sampled values $f(x_0), \ldots, f(x_n)$, then integrating the Lagrange basis polynomials produces weights c_j, so the approximation has the quadrature form

$$\sum_{j=0}^{n} c_j f(x_j).$$

Different choices of nodes and different polynomial degrees produce different rules and different error terms. When the interpolation nodes are equally spaced, the resulting formulas are called **Newton–Cotes rules**.

Newton–Cotes Rules

Figure 28 shows the three basic Newton–Cotes rules on a single interval. The midpoint rule replaces f by a constant, the trapezoid rule replaces f by a line, and Simpson's rule replaces f by a quadratic.

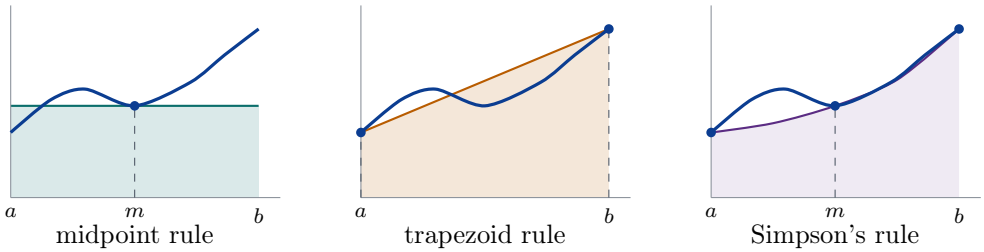


FIGURE 28

Definition 5.2 Let

$$m = \frac{a+b}{2}.$$

Approximating f by the constant polynomial $p_0(x) = f(m)$ gives the **midpoint rule**

$$Q_M = (b-a)f(m).$$

Definition 5.3 Approximating f by the linear interpolating polynomial through $(a, f(a))$ and $(b, f(b))$ gives the **trapezoid rule**

$$Q_T = \frac{b-a}{2}\big(f(a) + f(b)\big).$$

Proof. The degree-1 Lagrange interpolating polynomial is

$$p_1(x) = f(a)\frac{x-b}{a-b} + f(b)\frac{x-a}{b-a}.$$

Integrating over $[a, b]$ gives

$$\int_a^b p_1(x)\,\mathrm{d}x = f(a)\int_a^b \frac{x-b}{a-b}\,\mathrm{d}x + f(b)\int_a^b \frac{x-a}{b-a}\,\mathrm{d}x.$$

Each basis polynomial has integral $(b-a)/2$, so

$$\int_a^b p_1(x)\,\mathrm{d}x = \frac{b-a}{2}\big(f(a) + f(b)\big).$$

□

Definition 5.4 Let

$$m = \frac{a+b}{2}.$$

Approximating f by the quadratic interpolating polynomial through $(a, f(a))$, $(m, f(m))$, and $(b, f(b))$ gives Simpson's rule

$$Q_S = \frac{b-a}{6}\big(f(a) + 4f(m) + f(b)\big).$$

Proof. Write the quadratic interpolant in Lagrange form:

$$p_2(x) = f(a)\ell_0(x) + f(m)\ell_1(x) + f(b)\ell_2(x),$$

where

$$\ell_0(x) = \frac{(x-m)(x-b)}{(a-m)(a-b)}, \qquad \ell_1(x) = \frac{(x-a)(x-b)}{(m-a)(m-b)}, \qquad \text{and} \qquad \ell_2(x) = \frac{(x-a)(x-m)}{(b-a)(b-m)}.$$

Set $h = (b-a)/2$, so that $a = m-h$ and $b = m+h$. A direct calculation gives

$$\int_a^b \ell_0(x)\,\mathrm{d}x = \frac{h}{3}, \qquad \int_a^b \ell_1(x)\,\mathrm{d}x = \frac{4h}{3}, \qquad \text{and} \qquad \int_a^b \ell_2(x)\,\mathrm{d}x = \frac{h}{3}.$$

Therefore

$$\int_a^b p_2(x)\,\mathrm{d}x = \frac{h}{3}f(a) + \frac{4h}{3}f(m) + \frac{h}{3}f(b) = \frac{b-a}{6}\big(f(a) + 4f(m) + f(b)\big).$$

□

Remark 5.5 Since

$$Q_M = (b-a)f(m) \qquad \text{and} \qquad Q_T = \frac{b-a}{2}\big(f(a) + f(b)\big),$$

we may rewrite Simpson's rule as

$$Q_S = \frac{2Q_M + Q_T}{3}.$$

Example 5.6 Consider

$$I = \int_0^1 x \sin x \, \mathrm{d}x.$$

Integration by parts gives

$$I = \left[-x \cos x + \sin x\right]_0^1 = -\cos 1 + \sin 1 \approx 0.3012.$$

The midpoint rule gives

$$Q_M = f\left(\frac{1}{2}\right) = \frac{1}{2} \sin\left(\frac{1}{2}\right) \approx 0.2397.$$

The trapezoid rule gives

$$Q_T = \frac{f(0) + f(1)}{2} = \frac{\sin 1}{2} \approx 0.4207.$$

Simpson's rule gives

$$Q_S = \frac{1}{6}\left(f(0) + 4f\left(\frac{1}{2}\right) + f(1)\right) \approx 0.3001,$$

which is much more accurate in this case.

Proposition 5.7 Let $x_0, \ldots, x_n$ be distinct points in $[a, b]$, let p_n be the Lagrange interpolating polynomial for f at these nodes, and let $\ell_0, \ldots, \ell_n$ be the associated Lagrange basis polynomials. If $f \in C^{n+1}[a, b]$, then

$$\int_a^b f(x) \, \mathrm{d}x = \sum_{j=0}^{n} c_j f(x_j) + E_n[f],$$

where

$$c_j = \int_a^b \ell_j(x) \, \mathrm{d}x$$

and

$$E_n[f] = \int_a^b \big(f(x) - p_n(x)\big) \, \mathrm{d}x.$$

Moreover, if

$$\omega(x) = \prod_{k=0}^{n} (x - x_k),$$

then the pointwise interpolation remainder gives the rigorous error bound

$$|E_n[f]| \le \frac{\|f^{(n+1)}\|_\infty}{(n+1)!} \int_a^b |\omega(x)|\,\mathrm{d}x.$$

Proof. Since

$$p_n(x) = \sum_{j=0}^{n} f(x_j)\ell_j(x),$$

integrating both sides gives

$$\int_a^b p_n(x)\,\mathrm{d}x = \sum_{j=0}^{n} f(x_j) \int_a^b \ell_j(x)\,\mathrm{d}x = \sum_{j=0}^{n} c_j f(x_j).$$

Now write

$$\int_a^b f(x)\,\mathrm{d}x = \int_a^b p_n(x)\,\mathrm{d}x + \int_a^b \big(f(x) - p_n(x)\big)\,\mathrm{d}x.$$

This gives the quadrature formula with error term $E_n[f]$. By Theorem 2.12, for each x there is a point $\xi_x \in [a,b]$ such that

$$f(x) - p_n(x) = \frac{f^{(n+1)}(\xi_x)}{(n+1)!}\omega(x).$$

Taking absolute values, bounding $\left|f^{(n+1)}(\xi_x)\right|$ by its supremum, and integrating gives the stated error bound. This avoids treating the unspecified points ξ_x as though they formed a continuous function of x. □

Theorem 5.8 Suppose g and ϕ are continuous on $[a,b]$, and suppose ϕ does not change sign on $[a,b]$. Then there exists $\tau \in [a,b]$ such that

$$\int_a^b g(x)\phi(x)\,\mathrm{d}x = g(\tau)\int_a^b \phi(x)\,\mathrm{d}x.$$

Proof. Assume first that $\phi(x) \ge 0$ on $[a,b]$. Since g is continuous on a compact interval, it attains its minimum m and maximum M. Therefore

$$m\phi(x) \le g(x)\phi(x) \le M\phi(x)$$

for every $x \in [a, b]$. Integrating gives

$$m \int_a^b \phi(x)\,\mathrm{d}x \le \int_a^b g(x)\phi(x)\,\mathrm{d}x \le M \int_a^b \phi(x)\,\mathrm{d}x.$$

If

$$\int_a^b \phi(x)\,\mathrm{d}x = 0,$$

then the conclusion is immediate. Otherwise, dividing by the positive number

$$\int_a^b \phi(x)\,\mathrm{d}x$$

gives

$$m \le \frac{\int_a^b g(x)\phi(x)\,\mathrm{d}x}{\int_a^b \phi(x)\,\mathrm{d}x} \le M.$$

Because g takes every value between m and M, there exists $\tau \in [a, b]$ such that

$$g(\tau) = \frac{\int_a^b g(x)\phi(x)\,\mathrm{d}x}{\int_a^b \phi(x)\,\mathrm{d}x}.$$

Rearranging proves the result. If $\phi \le 0$, apply the same argument to $-\phi$. □

Error Bounds

Proposition 5.9 If $f \in C^2[a, b]$, then there exists $\xi_T \in (a, b)$ such that

$$\int_a^b f(x)\,\mathrm{d}x - Q_T = -\frac{(b-a)^3}{12} f''(\xi_T).$$

Proof. Let $E_T(f) = \int_a^b f(x)\,\mathrm{d}x - Q_T$. Twice integrating the Taylor formula with integral remainder, or equivalently integrating by parts, gives the Peano-kernel identity

$$E_T(f) = \int_a^b K_T(t) f''(t)\,\mathrm{d}t, \qquad K_T(t) = \frac{(t-a)(t-b)}{2}.$$

The kernel is nonpositive on $[a, b]$, so Theorem 5.8 gives a point $\xi_T \in (a, b)$ such that

$$E_T(f) = f''(\xi_T) \int_a^b K_T(t)\,\mathrm{d}t.$$

Finally,

$$\int_a^b K_T(t)\,\mathrm{d}t = -\frac{(b-a)^3}{12},$$

which proves the formula. □

Proposition 5.10 If $f \in C^2[a, b]$, then there exists $\xi_M \in (a, b)$ such that

$$\int_a^b f(x)\,\mathrm{d}x - Q_M = \frac{(b-a)^3}{24} f''(\xi_M).$$

Proof. Let $m = (a + b)/2$ and $E_M(f) = \int_a^b f(x)\,\mathrm{d}x - Q_M$. The corresponding Peano-kernel identity is

$$E_M(f) = \int_a^b K_M(t) f''(t)\,\mathrm{d}t, \qquad K_M(t) = \begin{cases} \dfrac{(t-a)^2}{2}, & a \le t \le m, \\ \dfrac{(b-t)^2}{2}, & m \le t \le b. \end{cases}$$

This identity follows directly by inserting the integral-remainder form of Taylor's theorem on the two sides of m and changing the order of integration. Since K_M is nonnegative, Theorem 5.8 gives a point $\xi_M \in (a, b)$ such that

$$E_M(f) = f''(\xi_M) \int_a^b K_M(t)\,\mathrm{d}t.$$

Finally,

$$\int_a^b K_M(t)\,\mathrm{d}t = \frac{(b-a)^3}{24},$$

which proves the formula. □

Proposition 5.11 If $f \in C^4[a, b]$, then there exists $\xi_S \in (a, b)$ such that

$$\int_a^b f(x)\,\mathrm{d}x - Q_S = -\frac{(b-a)^5}{2880} f^{(4)}(\xi_S).$$

Proof. Let $m = (a+b)/2$ and $h = (b-a)/2$, and define the error functional

$$L(F) = \int_{-h}^{h} F(t)\,\mathrm{d}t - \frac{h}{3}\big(F(-h) + 4F(0) + F(h)\big).$$

This functional annihilates every polynomial of degree at most 3. Taylor's formula with integral remainder therefore gives the Peano kernel representation

$$L(F) = \int_{-h}^{h} K(s) F^{(4)}(s)\,\mathrm{d}s,$$

where a direct calculation gives

$$K(s) = -\frac{(h-|s|)^3(h+3|s|)}{72} \leq 0.$$

Since $-K(s) \geq 0$, the weighted mean value theorem gives some $\theta \in (-h, h)$ such that

$$L(F) = F^{(4)}(\theta) \int_{-h}^{h} K(s)\,\mathrm{d}s.$$

We can find the integral of the kernel by applying L to $F(t) = t^4$:

$$\int_{-h}^{h} K(s)\,\mathrm{d}s = \frac{L(t^4)}{4!} = \frac{1}{24}\left(\frac{2h^5}{5} - \frac{2h^5}{3}\right) = -\frac{h^5}{90}.$$

Taking $F(t) = f(m+t)$ and $\xi_S = m + \theta$, and then using $h = (b-a)/2$, gives

$$\int_a^b f(x)\,\mathrm{d}x - Q_S = -\frac{(b-a)^5}{2880} f^{(4)}(\xi_S).$$

□

Remark 5.12 The midpoint and trapezoid rules are exact for all polynomials of degree at most 1, while Simpson's rule is exact for all polynomials of degree at most 3.

The single-interval formulas and local errors may be summarized as follows:

rule	approximation	error $\int_a^b f(x)\,dx - Q$	degree of exactness
midpoint	$(b-a)f(m)$	$\frac{(b-a)^3}{24}f''(\xi_M)$	1
trapezoid	$\frac{b-a}{2}\big(f(a)+f(b)\big)$	$-\frac{(b-a)^3}{12}f''(\xi_T)$	1
Simpson	$\frac{b-a}{6}\big(f(a)+4f(m)+f(b)\big)$	$-\frac{(b-a)^5}{2880}f^{(4)}(\xi_S)$	3

Example 5.13 Consider

$$I = \int_2^6 x^2 \ln x \, dx.$$

The antiderivative is

$$\int x^2 \ln x \, dx = \frac{x^3(3\ln x - 1)}{9},$$

so

$$I = \left[\frac{x^3(3\ln x - 1)}{9}\right]_2^6 \approx 104.0472.$$

Figure 29 shows the same integrand with the constant, linear, and quadratic approximations used by the three single-interval rules.

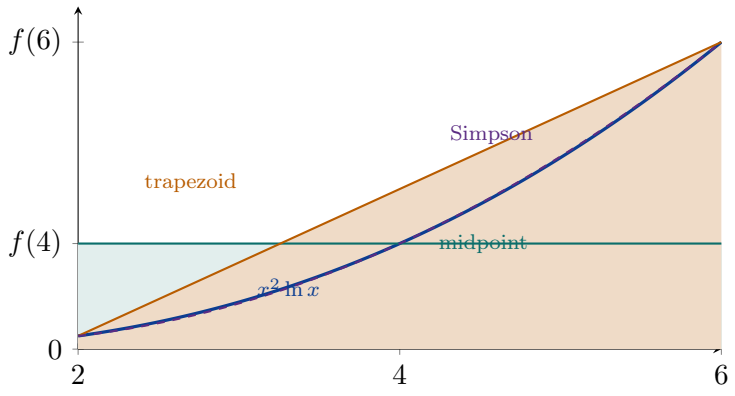


FIGURE 29

The midpoint rule gives

$$Q_M = 4f(4) = 4(22.1807) \approx 88.7228,$$

with absolute error approximately 15.3243. The trapezoid rule gives

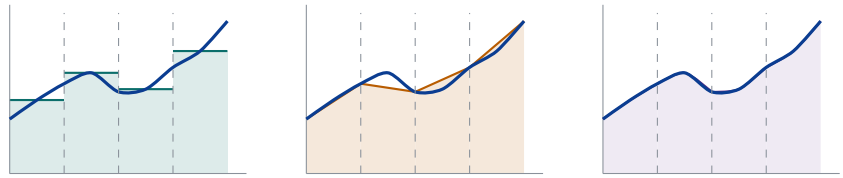
$$Q_T = \frac{4}{2}\big(f(2) + f(6)\big) = 2(2.7726 + 64.5033) \approx 134.5519,$$

with absolute error approximately 30.5047. Simpson's rule gives

$$Q_S = \frac{4}{6}\big(f(2) + 4f(4) + f(6)\big) \approx 103.9992,$$

with absolute error approximately 0.0480.

When the interval is large or the integrand is not well approximated by a low-degree polynomial on all of $[a, b]$, we divide the interval into smaller pieces and apply the basic rule on each subinterval. The composite rules in Figure 30 use the same local approximations as the single-interval rules, but repeat them across a partition.



composite midpoint composite trapezoid composite Simpson

Figure 30

Let

$$h = \frac{b-a}{n} \qquad \text{and} \qquad x_i = a + ih, \quad i = 0, 1, \ldots, n.$$

Also let

$$m_i = \frac{x_{i-1} + x_i}{2} \quad (i = 1, \ldots, n).$$

Definition 5.14 The **composite midpoint rule** is

$$M_n = h \sum_{i=1}^{n} f(m_i).$$

Definition 5.15 The **composite trapezoid rule** is

$$T_n = \frac{h}{2}\left(f(x_0) + 2\sum_{i=1}^{n-1} f(x_i) + f(x_n)\right).$$

Definition 5.16 The **composite Simpson's rule** is

$$S_n = \frac{h}{6}\sum_{i=1}^{n}\big(f(x_{i-1}) + 4f(m_i) + f(x_i)\big).$$

Equivalently,

$$S_n = \frac{h}{6}\left(f(x_0) + 4\sum_{i=1}^{n} f(m_i) + 2\sum_{i=1}^{n-1} f(x_i) + f(x_n)\right).$$

Proposition 5.17 If $f \in C^2[a,b]$, then there exists $\xi_M \in (a,b)$ such that

$$\int_a^b f(x)\,\mathrm{d}x - M_n = \frac{(b-a)}{24}h^2 f''(\xi_M).$$

Proof. By Proposition 5.10, the midpoint rule error on each subinterval $[x_{i-1}, x_i]$ is

$$\int_{x_{i-1}}^{x_i} f(x)\,\mathrm{d}x - hf(m_i) = \frac{h^3}{24}f''(\xi_i)$$

for some $\xi_i \in (x_{i-1}, x_i)$. Summing over i gives

$$\int_a^b f(x)\,\mathrm{d}x - M_n = \frac{h^3}{24}\sum_{i=1}^{n} f''(\xi_i).$$

Let

$$m = \min_{x\in[a,b]} f''(x) \qquad \text{and} \qquad M = \max_{x\in[a,b]} f''(x).$$

Then

$$nm \le \sum_{i=1}^{n} f''(\xi_i) \le nM.$$

By continuity of f'', there exists $\xi_M \in (a,b)$ such that

$$\sum_{i=1}^{n} f''(\xi_i) = nf''(\xi_M).$$

Therefore

$$\int_a^b f(x)\,\mathrm{d}x - M_n = \frac{nh^3}{24} f''(\xi_M) = \frac{(b-a)}{24} h^2 f''(\xi_M).$$

□

Proposition 5.18 If $f \in C^2[a,b]$, then there exists $\xi_T \in (a,b)$ such that

$$\int_a^b f(x)\,\mathrm{d}x - T_n = -\frac{(b-a)}{12} h^2 f''(\xi_T).$$

Proof. Applying Proposition 5.9 on each subinterval gives

$$\int_{x_{i-1}}^{x_i} f(x)\,\mathrm{d}x - \frac{h}{2}\big(f(x_{i-1}) + f(x_i)\big) = -\frac{h^3}{12} f''(\eta_i)$$

for some $\eta_i \in (x_{i-1}, x_i)$. Summing over i and using the same continuity argument yields

$$\int_a^b f(x)\,\mathrm{d}x - T_n = -\frac{nh^3}{12} f''(\xi_T) = -\frac{(b-a)}{12} h^2 f''(\xi_T)$$

for some $\xi_T \in (a,b)$. □

Proposition 5.19 If $f \in C^4[a,b]$, then there exists $\xi_S \in (a,b)$ such that

$$\int_a^b f(x)\,\mathrm{d}x - S_n = -\frac{(b-a)}{2880} h^4 f^{(4)}(\xi_S).$$

Proof. Applying Proposition 5.11 on each subinterval gives

$$\int_{x_{i-1}}^{x_i} f(x)\,\mathrm{d}x - \frac{h}{6}\big(f(x_{i-1}) + 4f(m_i) + f(x_i)\big) = -\frac{h^5}{2880} f^{(4)}(\zeta_i)$$

for some $\zeta_i \in (x_{i-1}, x_i)$. Summing over i yields

$$\int_a^b f(x)\,\mathrm{d}x - S_n = -\frac{h^5}{2880}\sum_{i=1}^{n} f^{(4)}(\zeta_i).$$

As before, continuity of $f^{(4)}$ implies that there exists $\xi_S \in (a, b)$ such that

$$\sum_{i=1}^{n} f^{(4)}(\zeta_i) = n f^{(4)}(\xi_S).$$

Hence

$$\int_a^b f(x)\,\mathrm{d}x - S_n = -\frac{nh^5}{2880} f^{(4)}(\xi_S) = -\frac{(b-a)}{2880} h^4 f^{(4)}(\xi_S).$$

□

Gaussian Quadrature

The Newton–Cotes rules fix the nodes first and then determine the weights. **Gaussian quadrature** takes a different approach: both the nodes and the weights are chosen to maximize the degree of exactness.

Figure 31 illustrates the difference. Newton–Cotes rules sample at prescribed equally spaced nodes, while Gaussian quadrature moves the nodes and changes the weights so that the rule integrates a larger class of polynomials exactly.

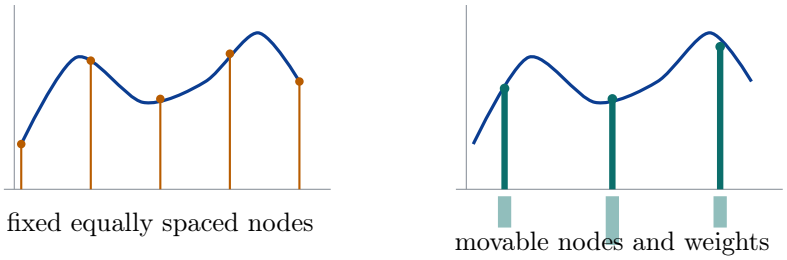


FIGURE 31

Definition 5.20 A quadrature rule

$$\int_a^b f(x)\,\mathrm{d}x \approx \sum_{j=0}^{n-1} c_j f(x_j)$$

has **degree of exactness** d if it is exact for every polynomial of degree at most d, but not for every polynomial of degree $d+1$.

With n nodes, an unrestricted quadrature rule has $2n$ unknowns: the n nodes and the n weights. Requiring exactness for the polynomial basis $1, x, x^2, \ldots, x^{2n-1}$ gives $2n$ conditions, so the best possible Gaussian rule is expected to have degree of exactness $2n-1$.

Proposition 5.21 A symmetric three-point quadrature rule on $[-1,1]$ that is exact for all polynomials of degree at most 5 is

$$\int_{-1}^{1} f(x)\,\mathrm{d}x \approx \frac{5}{9} f\left(-\sqrt{\frac{3}{5}}\right) + \frac{8}{9} f(0) + \frac{5}{9} f\left(\sqrt{\frac{3}{5}}\right).$$

Proof. Symmetry suggests that the nodes should have the form

$$x_0 = -t, \qquad x_1 = 0, \qquad \text{and} \qquad x_2 = t,$$

and that the outer weights should match:

$$c_0 = c_2 = A \qquad \text{and} \qquad c_1 = B.$$

Exactness for odd powers x and x^3 is then automatic. It remains to enforce exactness for 1, x^2, and x^4. For the constant polynomial 1,

$$\int_{-1}^{1} 1\,\mathrm{d}x = 2,$$

so

$$2A + B = 2.$$

For x^2,

$$\int_{-1}^{1} x^2\,\mathrm{d}x = \frac{2}{3},$$

so

$$2At^2 = \frac{2}{3}.$$

For x^4,

$$\int_{-1}^{1} x^4\,\mathrm{d}x = \frac{2}{5},$$

so

$$2At^4 = \frac{2}{5}.$$

Dividing the last equation by the previous one gives

$$t^2 = \frac{3}{5},$$

hence

$$t = \sqrt{\frac{3}{5}}.$$

Then

$$2A \cdot \frac{3}{5} = \frac{2}{3},$$

so

$$A = \frac{5}{9}.$$

Finally, the first equation gives

$$B = 2 - 2A = 2 - \frac{10}{9} = \frac{8}{9}.$$

This produces the stated rule. □

Example 5.22 Consider

$$I = \int_{-1}^{1} x \sin x \, \mathrm{d}x.$$

Since

$$\int x \sin x \, \mathrm{d}x = -x \cos x + \sin x,$$

we have

$$I = 2(\sin 1 - \cos 1) \approx 0.6023.$$

Simpson's rule on $[-1, 1]$ gives

$$Q_S = \frac{2}{6}\big(f(-1) + 4f(0) + f(1)\big) = \frac{2 \sin 1}{3} \approx 0.5610,$$

whose relative error is approximately 6.87%. The three-point Gaussian quadrature rule gives

$$Q_G = \frac{5}{9} f\left(-\sqrt{\frac{3}{5}}\right) + \frac{8}{9} f(0) + \frac{5}{9} f\left(\sqrt{\frac{3}{5}}\right) = \frac{10}{9}\sqrt{\frac{3}{5}} \sin\left(\sqrt{\frac{3}{5}}\right) \approx 0.6020,$$

whose relative error is approximately 0.061%.

Definition 5.23 The **Legendre polynomials** are defined recursively by

$$P_0(x) = 1 \qquad \text{and} \qquad P_1(x) = x,$$

and

$$P_{n+1}(x) = \frac{2n+1}{n+1} x P_n(x) - \frac{n}{n+1} P_{n-1}(x) \quad (n \geq 1).$$

The nodes of the n-point Gauss–Legendre rule on $[-1, 1]$ are the roots of P_n. For example,

$$P_2(x) = \frac{1}{2}(3x^2 - 1) \qquad \text{and} \qquad P_3(x) = \frac{1}{2}(5x^3 - 3x).$$

The roots of P_3 are

$$0 \qquad \text{and} \qquad \pm\sqrt{\frac{3}{5}},$$

which are exactly the three-point Gaussian quadrature nodes derived for this rule. For each order n, the nodes and weights depend only on n and the interval, not on the particular integrand. They are therefore usually tabulated or computed once, rather than found by solving the nonlinear exactness equations for every new integral.

Proposition 5.24 For any interval $[a, b]$, the affine change of variables

$$x = \frac{a+b}{2} + \frac{b-a}{2} t$$

maps $t \in [-1, 1]$ to $x \in [a, b]$, and

$$\int_a^b f(x)\,\mathrm{d}x = \frac{b-a}{2} \int_{-1}^{1} f\left(\frac{a+b}{2} + \frac{b-a}{2} t\right) \mathrm{d}t.$$

Proof. The substitution gives

$$\mathrm{d}x = \frac{b-a}{2}\,\mathrm{d}t.$$

When $t = -1$,

$$x = \frac{a+b}{2} - \frac{b-a}{2} = a,$$

and when $t = 1$,

$$x = \frac{a+b}{2} + \frac{b-a}{2} = b.$$

Substituting into the integral yields the formula. □

Example 5.25 Consider

$$I = \int_1^{3/2} e^{-x^2}\,\mathrm{d}x.$$

The affine change of variables from $[-1, 1]$ to $[1, 3/2]$ is

$$x = \frac{1+3/2}{2} + \frac{3/2-1}{2}t = \frac{5}{4} + \frac{t}{4} = \frac{5+t}{4}.$$

Thus

$$I = \frac{1}{4}\int_{-1}^{1} \exp\left(-\left(\frac{5+t}{4}\right)^2\right)\mathrm{d}t.$$

Applying the three-point Gaussian quadrature rule gives

$$I \approx \frac{1}{4}\left[\frac{5}{9}\exp\left(-\left(\frac{5-\sqrt{3/5}}{4}\right)^2\right) + \frac{8}{9}\exp\left(-\left(\frac{5}{4}\right)^2\right) + \frac{5}{9}\exp\left(-\left(\frac{5+\sqrt{3/5}}{4}\right)^2\right)\right].$$

This evaluates to approximately 0.1093642, while the actual value is approximately 0.1093643. Not bad!

An n-point Gauss–Legendre quadrature rule is exact for all polynomials of degree at most $2n-1$. This is better than Newton–Cotes rules using the same number of function evaluations. The error term for an n-point Gaussian quadrature rule involves a derivative of order $2n$; for example, the three-point rule has an error controlled by a sixth derivative. By comparison, the single-interval midpoint and trapezoid errors involve second derivatives, while Simpson's rule involves a fourth derivative.

Monte Carlo Integration

Classical quadrature rules become increasingly expensive in high dimensions. **Monte Carlo integration** replaces deterministic sampling by random sampling, which doesn't suffer quite so much from the curse of dimensionality. The idea is to estimate the integral by the average value of the integrand at randomly chosen points.

If $U \sim \text{Unif}(a, b)$, then

$$\mathbb{E}[f(U)] = \frac{1}{b-a} \int_a^b f(x)\,\mathrm{d}x.$$

Equivalently, Theorem 5.8 with $\phi(x) = 1$ says that on $[0, 1]$ the integral equals the average height $f(c)$ for some point $c \in [0, 1]$. Monte Carlo integration estimates this average height by evaluating f at randomly chosen points. Provided $f(U)$ is integrable, the weak law of large numbers guarantees convergence in probability to the true integral as the number of random points increases.

Definition 5.26 Let $U_1, \ldots, U_n$ be independent random variables uniformly distributed on $[a, b]$. The **Monte Carlo estimator** for

$$I = \int_a^b f(x)\,\mathrm{d}x$$

is

$$I_n = (b-a)\frac{1}{n}\sum_{k=1}^{n} f(U_k).$$

Proposition 5.27 Assume $f(U_1)$ has finite variance. Then

$$\mathbb{E}\left[I_n\right] = I$$

and

$$\text{Var}(I_n) = \frac{(b-a)^2}{n}\,\text{Var}(f(U_1)).$$

Consequently, the root mean square error is $O(n^{-1/2})$.

Proof. Since U_1 is uniform on $[a, b]$,

$$\mathbb{E}\left[f(U_1)\right] = \frac{1}{b-a}\int_a^b f(x)\,\mathrm{d}x = \frac{I}{b-a}.$$

Therefore

$$\mathbb{E}\left[I_n\right] = (b-a)\frac{1}{n}\sum_{k=1}^{n}\mathbb{E}\left[f(U_k)\right] = (b-a)\mathbb{E}\left[f(U_1)\right] = I.$$

Because the U_k are independent,

$$\text{Var}(I_n) = (b-a)^2\,\text{Var}\left(\frac{1}{n}\sum_{k=1}^{n} f(U_k)\right)$$

$$= (b-a)^2 \frac{1}{n^2} \sum_{k=1}^{n} \operatorname{Var}(f(U_k)).$$

All summands are equal, so

$$\operatorname{Var}(I_n) = \frac{(b-a)^2}{n} \operatorname{Var}(f(U_1)).$$

Taking square roots shows that the typical error size is proportional to $n^{-1/2}$. □

Example 5.28 For

$$I = \int_0^1 x \sin x \, \mathrm{d}x \approx 0.3012,$$

Monte Carlo estimates vary from one trial to another. Five independent trials for several values of n give:

n	trial 1	trial 2	trial 3	trial 4	trial 5
1	0.1715	0.0399	0.2025	0.1887	0.6316
3	0.0124	0.2382	0.1334	0.0886	0.1714
5	0.4766	0.0725	0.2886	0.3901	0.1994
10	0.2267	0.3179	0.2786	0.2165	0.3101
100	0.3307	0.2826	0.2144	0.2980	0.2845
1000	0.3105	0.3003	0.3214	0.2987	0.3034
10000	0.3036	0.3026	0.3069	0.3058	0.3058

The estimates converge, but the convergence is slow: the typical error is reduced by a factor of about 10 only after increasing n by a factor of about 100.

Example 5.29 To estimate the area of the unit quarter-circle

$$\{(x, y) \in [0, 1]^2 : x^2 + y^2 \leq 1\},$$

we may generate random points (X_k, Y_k) uniformly in the unit square and count the fraction that fall inside the quarter-circle, as shown in Figure 32.

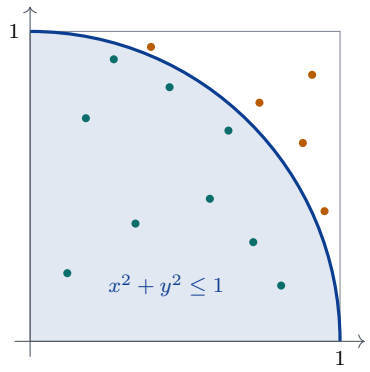


Figure 32

If d_n denotes this fraction, then

$$d_n \approx \frac{\pi/4}{1},$$

so

$$\pi \approx 4d_n.$$

This is a geometric Monte Carlo method.

Remark 5.30 Monte Carlo convergence is slower than the high-order deterministic rules in one dimension, but its cost scales much better with dimension. A tensor-product quadrature with m points per coordinate in d dimensions needs m^d function evaluations, whereas a Monte Carlo estimate still uses only n sampled points.

On the unit square and unit cube, the basic formulas are

$$\int_0^1 \int_0^1 f(x,y)\,\mathrm{d}x\,\mathrm{d}y \approx \frac{1}{n}\sum_{k=1}^{n} f(X_k, Y_k),$$

and

$$\int_0^1 \int_0^1 \int_0^1 f(x,y,z)\,\mathrm{d}x\,\mathrm{d}y\,\mathrm{d}z \approx \frac{1}{n}\sum_{k=1}^{n} f(X_k, Y_k, Z_k),$$

where the sampled points are independent and uniformly distributed in the corresponding box.

More generally, let

$$D = [a_1, b_1] \times \cdots \times [a_d, b_d] \subseteq \mathbb{R}^d$$

be a d-dimensional hyperrectangle with volume

$$\operatorname{vol}(D) = \prod_{j=1}^{d} (b_j - a_j).$$

If $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_n$ are independent random vectors uniformly distributed on D, then

$$\int_D f(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} \approx \operatorname{vol}(D) \frac{1}{n} \sum_{k=1}^{n} f(\boldsymbol{X}_k).$$

Example 5.31 Consider the three-dimensional integral

$$I = \int_0^{9/10} \int_0^{1} \int_0^{11/10} \sqrt{4 - x^2 - y^2 - z^2}\,\mathrm{d}z\,\mathrm{d}y\,\mathrm{d}x.$$

If X_k, Y_k, and Z_k are sampled uniformly from $(0, 9/10)$, $(0, 1)$, and $(0, 11/10)$ respectively, then

$$I \approx \frac{9}{10} \cdot 1 \cdot \frac{11}{10} \cdot \frac{1}{n} \sum_{k=1}^{n} \sqrt{4 - X_k^2 - Y_k^2 - Z_k^2}.$$

The volume factor is 99/100. The actual value is approximately 1.705759, and five independent Monte Carlo trials give:

n	trial 1	trial 2	trial 3	trial 4	trial 5
10	1.6721	1.8058	1.6777	1.7846	1.7235
100	1.6938	1.6910	1.6905	1.7020	1.7056
1000	1.7000	1.7084	1.7082	1.7067	1.7105
10000	1.7068	1.7050	1.7062	1.7070	1.7016
100000	1.7051	1.7049	1.7062	1.7058	1.7054
1000000	1.7057	1.7057	1.7056	1.7059	1.7056

True random numbers can be generated from physical events (such as flipping a coin, throwing dice, or measuring radioactive decay), but doing so is usually too slow for numerical integration. In practice we use **pseudorandom numbers** generated algorithmically. Such sequences are seeded, deterministic, and sufficiently unpredictable for typical Monte Carlo computations.

In MATLAB, `rand()` returns a pseudorandom value drawn approximately uniformly from $(0, 1)$. The command `rand(M,N)` returns an M-by-N matrix, while `rand(N)` returns an N-by-N matrix. The commands `rand(..., 'double')` and `rand(..., 'single')` choose the floating-point class of the returned values.

Uniform samples on a general interval $[a, b]$ can be generated by

```
r = a + (b-a) * rand(N,1);
```

Integer samples from $\{1, \ldots, 100\}$ can be generated by

```
r = randi(100,1,5);
```

The default startup state of the random number generator can be restored by

```
rng('default')
rand(1,5)
```

which makes the subsequent `rand`, `randi`, and `randn` sequences reproducible.

Lecture 6 — Tuesday, March 22, 2016

Bisection and Regula Falsi

A **root** of a continuous function $f : \mathbb{R} \to \mathbb{R}$ is a number x^* such that

$$f(x^*) = 0.$$

In floating-point arithmetic the computed value $\mathrm{fl}(x^*)$ may not equal the exact root, and the computed residual $\mathrm{fl}(f(x^*))$ may not be exactly zero. We usually settle for an approximation $\widehat{x}$ for which the residual $|f(\widehat{x})|$ is below a prescribed tolerance. Depending on the problem, the root may be real or complex; here the roots are real and the functions are continuous and real-valued.

The first two methods are bracketing methods. They begin with points $a_0 < b_0$ such that $f(a_0)f(b_0) < 0$, so the intermediate value theorem guarantees a root in $[a_0, b_0]$. In practice,

the initial bracket may come from analytic information, a graph, or a coarse search for a sign change.

Figure 33 shows bisection applied to $f(x) = \sin x$ on $[1, 6]$. The root in this interval is π, and each step keeps the half-interval whose endpoints still have opposite signs.

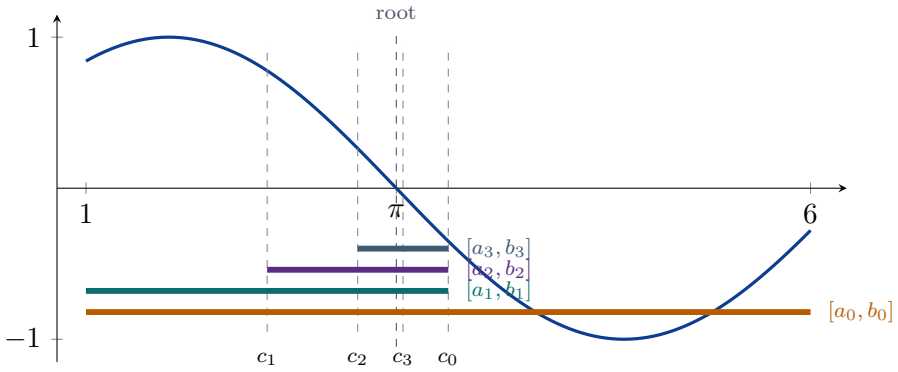


FIGURE 33

Definition 6.1 Assume f is continuous on $[a_0, b_0]$ and

$$f(a_0)f(b_0) < 0.$$

Define

$$c_k = \frac{a_k + b_k}{2}.$$

If $f(c_k) = 0$, stop. Otherwise choose

$$[a_{k+1}, b_{k+1}] = \begin{cases} [a_k, c_k], & \text{if } f(a_k)f(c_k) < 0, \\ [c_k, b_k], & \text{if } f(c_k)f(b_k) < 0. \end{cases}$$

This iteration is called the **bisection method**.

Theorem 6.2 Suppose f is continuous on $[a_0, b_0]$ and $f(a_0)f(b_0) < 0$. Then for every $k \geq 0$:

$$f(a_k)f(b_k) \leq 0 \qquad \text{and} \qquad b_k - a_k = \frac{b_0 - a_0}{2^k},$$

and there exists a root $x^* \in [a_k, b_k]$. Moreover, the midpoint approximation satisfies

$$|c_k - x^*| \leq \frac{b_0 - a_0}{2^{k+1}}.$$

Proof. The initial interval contains a root by the intermediate value theorem. At step k, either $f(c_k) = 0$, in which case the procedure stops, or one of the subintervals $[a_k, c_k]$ and $[c_k, b_k]$ has endpoints of opposite sign. The algorithm chooses such a subinterval, so the next interval again contains a root. By induction, every interval $[a_k, b_k]$ contains a root x^*. Each step halves the interval length, so

$$b_{k+1} - a_{k+1} = \frac{b_k - a_k}{2}.$$

Induction gives

$$b_k - a_k = \frac{b_0 - a_0}{2^k}.$$

Since c_k is the midpoint of $[a_k, b_k]$ and $x^* \in [a_k, b_k]$,

$$|c_k - x^*| \leq \frac{b_k - a_k}{2} = \frac{b_0 - a_0}{2^{k+1}}.$$

□

Corollary 6.3 To guarantee

$$|c_k - x^*| \leq \varepsilon,$$

it suffices to choose

$$k \geq \left\lceil \log_2 \left(\frac{b_0 - a_0}{2\varepsilon} \right) \right\rceil.$$

Proof. By Theorem 6.2, it is enough that

$$\frac{b_0 - a_0}{2^{k+1}} \leq \varepsilon.$$

Rearranging gives

$$2^{k+1} \geq \frac{b_0 - a_0}{\varepsilon},$$

which is equivalent to the stated bound after taking base-2 logarithms. □

Bisection is reliable because it preserves a bracketing interval, but it converges only linearly: each step reduces the error by roughly a factor of 1/2.

Example 6.4 Suppose bisection starts with $[a_0, b_0] = [1, 6]$. How many steps guarantee

$$b_k - a_k \leq 10^{-5}?$$

Solution. Since the interval length after k bisection steps is

$$b_k - a_k = \frac{5}{2^k},$$

we need

$$\frac{5}{2^k} \le 10^{-5}.$$

Equivalently,

$$2^k \ge 5 \cdot 10^5,$$

so

$$k \ge \lceil \log_2(5 \cdot 10^5) \rceil = 19.$$

Thus 19 bisection steps are sufficient. □

Figure 34 shows the same starting bracket for $f(x) = \sin x$ on $[1, 6]$, but the new point is the x-intercept of the secant line through the current endpoints rather than the midpoint.

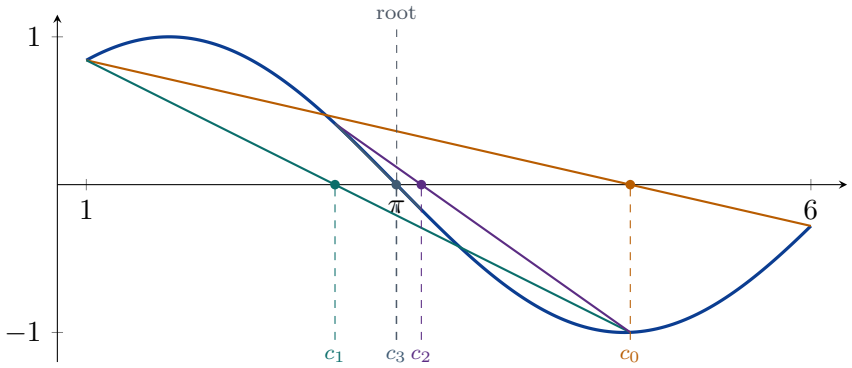


FIGURE 34

Definition 6.5 Assume $f(a_k)f(b_k) < 0$. The **regula falsi approximation** is the x-intercept of the secant line through $(a_k, f(a_k))$ and $(b_k, f(b_k))$:

$$c_k = b_k - \frac{f(b_k)(b_k - a_k)}{f(b_k) - f(a_k)} = \frac{a_k f(b_k) - b_k f(a_k)}{f(b_k) - f(a_k)}.$$

After computing c_k, we replace whichever endpoint has the same sign as $f(c_k)$.

Proposition 6.6 If f is continuous and $f(a_k)f(b_k) < 0$, then the regula falsi iterate c_k lies in (a_k, b_k), and the updated interval again brackets a root.

Proof. Because $f(a_k)$ and $f(b_k)$ have opposite signs, the secant line through the endpoints crosses the x-axis between a_k and b_k, so $c_k \in (a_k, b_k)$. If $f(c_k) = 0$, then c_k is a root. Otherwise, exactly one of the products

$$f(a_k)f(c_k) \qquad \text{and} \qquad f(c_k)f(b_k)$$

is negative. Replacing the appropriate endpoint preserves the sign change, so the new interval still contains a root by continuity. □

Regula falsi is again a bracketing method, so it is guaranteed to converge in exact arithmetic. It is often faster than bisection, but it can be slow if one endpoint barely moves.

Newton, Secant, and Fixed-Point Methods

Definition 6.7 Assume f is differentiable and choose an initial guess x_0. **Newton's method** defines

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)},$$

provided $f'(x_k) \neq 0$.

This formula comes from replacing the graph of f near x_k by its tangent line. The tangent line at x_k is

$$y = f(x_k) + f'(x_k)(x - x_k),$$

and setting $y = 0$ gives the next iterate.

Figure 35 shows Newton's method applied to $f(x) = \sin x$ on $[1, 6]$. Starting at $x_0 = 2$, the tangent line sends the next iterate past the root; the later tangents then move the iterates back toward π.

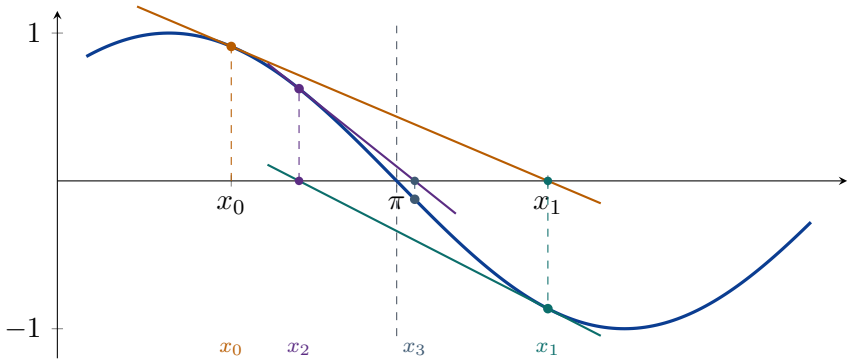


Figure 35

Example 6.8 Newton's method can cycle if the hypotheses for convergence are not met. For

$$f(x) = x^3 - 5x$$

with $x_0 = 1$, we have

$$f'(x) = 3x^2 - 5,$$

and hence

$$x_1 = 1 - \frac{1^3 - 5(1)}{3(1)^2 - 5} = -1.$$

The next step gives

$$x_2 = -1 - \frac{(-1)^3 - 5(-1)}{3(-1)^2 - 5} = 1.$$

Thus the iterates alternate between 1 and -1, even though neither point is a root. Figure 36 illustrates this two-cycle.

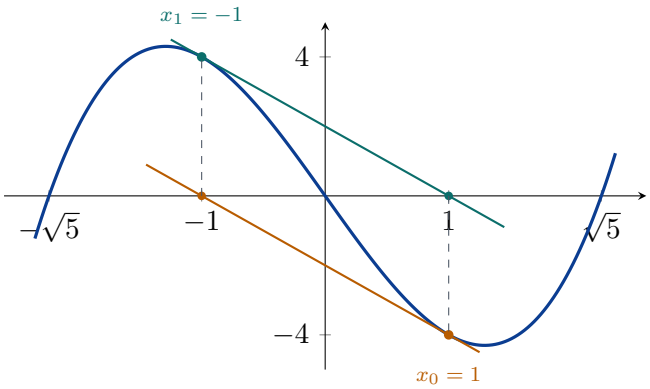


FIGURE 36

Example 6.9 To solve

$$f(x) = x^3 - 5 = 0,$$

start with $x_0 = 1$. Then

$$x_{k+1} = x_k - \frac{x_k^3 - 5}{3x_k^2}.$$

The first few steps are

$$x_1 = 1 - \frac{1 - 5}{3} = \frac{7}{3} \approx 2.3333,$$

$$x_2 = \frac{7}{3} - \frac{(7/3)^3 - 5}{3(7/3)^2} \approx 1.8623,$$

$$x_3 \approx 1.7140 \qquad \text{and} \qquad x_4 \approx 1.7100.$$

The exact root is

$$\sqrt[3]{5} \approx 1.7099759,$$

so the convergence is already very rapid once the iterates are near the root.

Newton's method is often much faster than bracketing methods, but it is not globally reliable. If the initial guess is poor, the sequence may diverge, cycle, or encounter a point where $f'(x_k)$ is very small. It also requires derivative evaluations, which may be more expensive or less convenient than function evaluations.

The secant method removes the need to evaluate f' explicitly.

Definition 6.10 Choose two initial guesses x_0 and x_1. The **secant method** defines

$$x_{k+1} = x_k - \frac{f(x_k)(x_k - x_{k-1})}{f(x_k) - f(x_{k-1})} \quad (k \geq 1),$$

provided $f(x_k) \neq f(x_{k-1})$.

This method approximates the derivative in Newton's method by the finite difference

$$f'(x_k) \approx \frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}.$$

It avoids explicit derivative evaluations, but it no longer preserves a bracketing interval.

The secant update can also suffer from catastrophic cancellation when x_k and x_{k-1} are very close or when $f(x_k)$ and $f(x_{k-1})$ are nearly equal. In those cases the finite-difference denominator may be dominated by roundoff error.

Figure 37 shows the secant method for $f(x) = \sin x$. Unlike regula falsi, the secant method uses the two most recent iterates; it does not necessarily keep a bracket around the root.

The secant method usually converges faster than bisection and regula falsi, though not as fast as Newton's method near a simple root. A more refined analysis shows that its order of convergence is the golden ratio

$$\frac{1 + \sqrt{5}}{2} \approx 1.618.$$

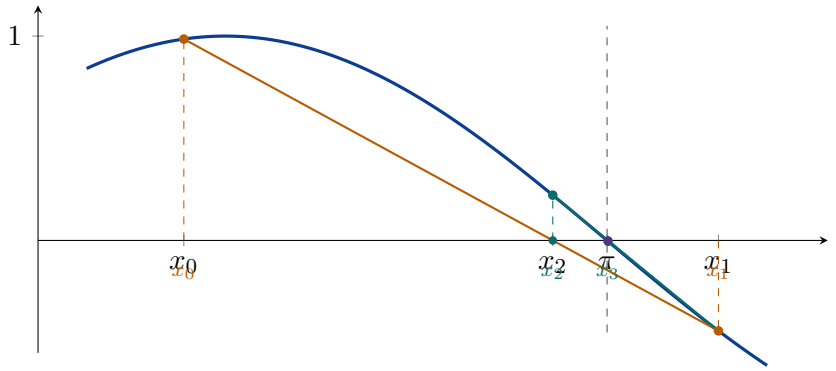


FIGURE 37

Fixed-point iteration rewrites the problem in the form $x = g(x)$.

Definition 6.11 If a root-finding problem can be rewritten as

$$x = g(x),$$

then we may generate approximations by

$$x_{k+1} = g(x_k).$$

Any solution x^* of $x^* = g(x^*)$ is called a **fixed point** of g.

For root finding, the chosen function g must be constructed so that $x^* = g(x^*)$ if and only if $f(x^*) = 0$. Otherwise, the fixed-point iteration may converge to a point that is not a root of the original function.

Example 6.12 To solve

$$\sin x = 0,$$

we may choose

$$g(x) = x - \sin x \qquad \text{or} \qquad h(x) = x + \sin x.$$

Both functions have the zeros of $\sin x$ as fixed points, but their attracting fixed points can be different. Starting from $x_0 = 2$, the iteration $x_{k+1} = g(x_k)$ gives

$$x_1 = 2 - \sin 2 \approx 1.0907, \qquad x_2 \approx 0.2038, \qquad \text{and} \qquad x_3 \approx 0.0014,$$

so the iterates converge to 0. Starting from the same $x_0 = 2$, the iteration $x_{k+1} = h(x_k)$

gives

$$x_1 = 2 + \sin 2 \approx 2.9093, \qquad x_2 \approx 3.1395, \qquad \text{and} \qquad x_3 \approx 3.1416,$$

so the iterates converge to π. The local reason is visible from the derivatives:

$$g'(x) = 1 - \cos x \qquad \text{and} \qquad h'(x) = 1 + \cos x.$$

At 0,

$$|g'(0)| = 0 < 1 \qquad \text{and} \qquad |h'(0)| = 2 > 1.$$

At π,

$$|g'(\pi)| = 2 > 1 \qquad \text{and} \qquad |h'(\pi)| = 0 < 1.$$

Thus g attracts nearby iterates to 0 and repels nearby iterates from π, while h does the opposite. Figure 38 summarizes these two iteration paths.

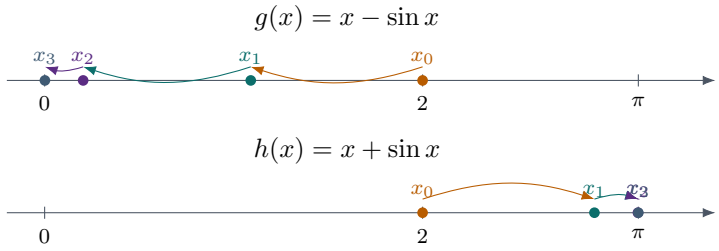


FIGURE 38

Stopping criteria are needed in practical implementations. Ideally, an algorithm would stop when $f(x_k) = 0$, but exact zero residuals are not realistic in floating-point arithmetic.

Common stopping criteria include requiring one of the following:

$$|f(x_k)| \le \varepsilon_1, \qquad |x_{k+1} - x_k| \le \varepsilon_2, \qquad \text{and} \qquad b_k - a_k \le \varepsilon_3$$

for prescribed tolerances. The first criterion checks the residual, but a small residual does not always mean that x_k is close to a root unless the scale and conditioning of the problem are understood. The second criterion detects small changes in the iterates, but an iteration can stagnate far from a root. The third criterion applies to bracketing methods and gives a direct location guarantee. Robust implementations usually combine several stopping criteria with a maximum number of iterations.

Figure 39 illustrates the distinction between residual size, step size, and bracket width.

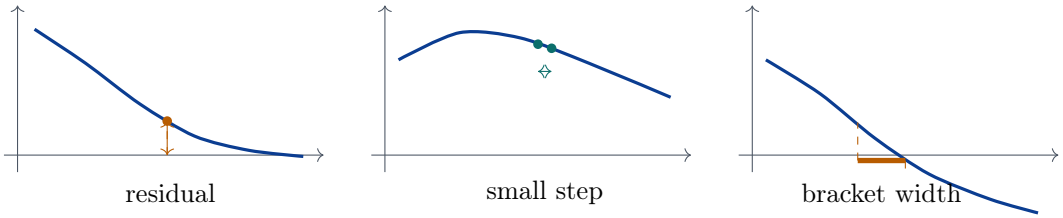


FIGURE 39

The speed of convergence can be quantified by the order of convergence.

Definition 6.13 Suppose $x_k \to x^*$ and define the errors

$$e_k = x_k - x^*.$$

If there exist numbers $q > 0$ and $\lambda > 0$ such that

$$\lim_{k\to\infty} \frac{|e_{k+1}|}{|e_k|^q} = \lambda,$$

then the sequence is said to converge to x^* with **order** q and **asymptotic error constant** λ.

When $q = 1$, the convergence is linear, so each step reduces the error by a roughly constant factor. When $q = 2$, the convergence is quadratic, so the error is roughly squared at each step, which is dramatically faster once the error is small.

For example, suppose two error sequences satisfy

$$|p_{n+1}| = \frac{1}{2}|p_n| \qquad \text{and} \qquad |q_{n+1}| = \frac{1}{2}|q_n|^2,$$

with $|p_0| = |q_0| = 1$. The difference after only a few steps is substantial:

n	linear error $\lvert p_n\rvert$	quadratic error $\lvert q_n\rvert$
1	5.0000×10^{-1}	5.0000×10^{-1}
2	2.5000×10^{-1}	1.2500×10^{-1}
3	1.2500×10^{-1}	7.8125×10^{-3}
4	6.2500×10^{-2}	3.0518×10^{-5}
5	3.1250×10^{-2}	4.6566×10^{-10}
6	1.5625×10^{-2}	1.0842×10^{-19}
7	7.8125×10^{-3}	5.8775×10^{-39}

Definition 6.14 A root x^* of f has **multiplicity** m if

$$f(x^*) = f'(x^*) = \cdots = f^{(m-1)}(x^*) = 0$$

but

$$f^{(m)}(x^*) \neq 0.$$

A **simple root** has multiplicity 1, and a **double root** has multiplicity 2.

Newton's method has quadratic convergence near simple roots.

Theorem 6.15 Suppose $f \in C^2$ on an interval containing x^*, with $f(x^*) = 0$ and $f'(x^*) \neq 0$. If x_0 is sufficiently close to x^*, then Newton's method is well-defined and converges at least quadratically to x^*. More precisely, whenever the errors are nonzero,

$$\lim_{k\to\infty} \frac{e_{k+1}}{e_k^2} = \frac{f''(x^*)}{2f'(x^*)}.$$

Thus the order is exactly 2 when $f''(x^*) \neq 0$.

Proof. Let

$$e_k = x_k - x^*.$$

Since $f(x^*) = 0$, Taylor's theorem applied at x_k gives

$$0 = f(x^*) = f(x_k) + f'(x_k)(x^* - x_k) + \frac{f''(\xi_k)}{2}(x^* - x_k)^2$$

for some ξ_k between x_k and x^*. Because $x^* - x_k = -e_k$, this becomes

$$f(x_k) = f'(x_k)e_k - \frac{f''(\xi_k)}{2}e_k^2.$$

Now Newton's iteration gives

$$e_{k+1} = x_{k+1} - x^* = x_k - \frac{f(x_k)}{f'(x_k)} - x^* = e_k - \frac{f(x_k)}{f'(x_k)}.$$

Substituting in Taylor's formula yields

$$e_{k+1} = e_k - \frac{f'(x_k)e_k - \frac{f''(\xi_k)}{2}e_k^2}{f'(x_k)} = \frac{f''(\xi_k)}{2f'(x_k)}e_k^2.$$

Because $f'(x^*) \neq 0$ and f' is continuous, there is a neighborhood of x^* on which f' stays bounded away from 0. Also, f'' is bounded there. Hence there exists a constant $C > 0$ such that

$$|e_{k+1}| \leq C|e_k|^2$$

whenever x_k lies in that neighborhood. Choosing x_0 close enough that $C|e_0| < 1$ makes the errors decrease and keeps every iterate in the same neighborhood, proving convergence and the quadratic bound. Finally, $\xi_k \to x^*$ as $x_k \to x^*$, so the exact error identity gives

$$\frac{e_{k+1}}{e_k^2} = \frac{f''(\xi_k)}{2f'(x_k)} \longrightarrow \frac{f''(x^*)}{2f'(x^*)}.$$

□

Proposition 6.16 Suppose

$$f(x) = (x - x^*)^m g(x)$$

with $m \geq 2$, where g is continuously differentiable near x^* and $g(x^*) \neq 0$. Then the standard Newton iteration converges linearly near x^*, and

$$\lim_{k\to\infty} \frac{e_{k+1}}{e_k} = \frac{m-1}{m}.$$

Proof. Differentiate:

$$f'(x) = m(x - x^*)^{m-1} g(x) + (x - x^*)^m g'(x).$$

Hence

$$\frac{f(x)}{f'(x)} = \frac{(x - x^*)^m g(x)}{m(x - x^*)^{m-1} g(x) + (x - x^*)^m g'(x)} = \frac{x - x^*}{m + (x - x^*)g'(x)/g(x)}.$$

Therefore Newton's update satisfies the exact relation

$$\frac{e_{k+1}}{e_k} = 1 - \frac{1}{m + e_k g'(x_k)/g(x_k)}.$$

As a function of e_k, the right-hand side extends continuously to $e_k = 0$ with value $(m-1)/m \in (0, 1)$. It is therefore bounded in magnitude by some $q < 1$ throughout a sufficiently small neighborhood of the root. Any iteration started in that neighborhood remains there and has $e_k \to 0$; taking the limit in the displayed identity then gives the stated asymptotic factor. □

A general way to prove fixed-point convergence is to use contractions.

Definition 6.17 A function $g : [a, b] \to [a, b]$ is a **contraction** on $[a, b]$ if there exists $L \in [0, 1)$ such that

$$|g(x) - g(y)| \leq L|x - y|$$

for all $x, y \in [a, b]$.

In other words, g is L-Lipschitz. The contraction condition means that distances between points are uniformly shortened by at least the factor L. If g is differentiable on $[a, b]$ and $|g'(x)| \leq L < 1$ throughout the interval, then the mean value theorem implies that g is a contraction. Geometrically, every secant slope of g has magnitude at most L.

Theorem 6.18 Suppose $g : [a, b] \to [a, b]$ is continuous and is a contraction with constant $L < 1$. Then g has a unique fixed point $x^* \in [a, b]$, and for any initial value $x_0 \in [a, b]$, the iteration

$$x_{k+1} = g(x_k)$$

converges to x^*.

Proof. Define

$$\phi(x) = g(x) - x.$$

Because $g([a, b]) \subseteq [a, b]$, we have

$$\phi(a) = g(a) - a \geq 0 \qquad \text{and} \qquad \phi(b) = g(b) - b \leq 0.$$

Since ϕ is continuous, the intermediate value theorem gives a point $x^* \in [a, b]$ with

$$\phi(x^*) = 0,$$

that is,

$$g(x^*) = x^*.$$

To prove uniqueness, suppose u and v are fixed points. Then

$$|u - v| = |g(u) - g(v)| \leq L|u - v|.$$

Since $L < 1$, this forces $|u - v| = 0$, so $u = v$. Finally, let $x_0 \in [a, b]$. Since x^* is a fixed point,

$$|x_{k+1} - x^*| = |g(x_k) - g(x^*)| \leq L|x_k - x^*|.$$

Iterating this estimate gives

$$|x_k - x^*| \le L^k |x_0 - x^*|.$$

Because $0 \le L < 1$, the right-hand side tends to 0, so $x_k \to x^*$. □

Corollary 6.19 Suppose $x^* = g(x^*)$ and g is differentiable in a neighborhood of x^*. If

$$|g'(x^*)| < 1,$$

then there exists $\delta > 0$ such that any iteration started with $|x_0 - x^*| < \delta$ converges to x^*. If

$$|g'(x^*)| > 1,$$

then any sufficiently small nonzero perturbation away from x^* is pushed farther away, so the iteration is locally divergent.

Proof. If $|g'(x^*)| < 1$, continuity of g' implies that there exist $\delta > 0$ and $L < 1$ such that

$$|g'(x)| \le L$$

whenever $|x - x^*| \le \delta$. By the mean value theorem, for any x, y in that neighborhood, there exists ξ between x and y such that

$$|g(x) - g(y)| \le L|x - y|$$

for all x, y in that neighborhood. In particular,

$$|g(x) - x^*| \le L|x - x^*| \le L\delta < \delta,$$

so the closed neighborhood maps into itself and g is a contraction there. Theorem 6.18 then gives convergence. If $|g'(x^*)| > 1$, continuity of g' yields a neighborhood in which

$$|g'(x)| \ge \mu > 1.$$

Applying the mean value theorem to $g(x) - g(x^*)$ shows that for x sufficiently close to x^*,

$$|g(x) - x^*| \ge \mu |x - x^*|,$$

so the error grows rather than shrinks. Hence the fixed point is locally repelling. □

The derivative test is inconclusive when $|g'(x^*)| = 1$. For example, the fixed point 0 of $g(x) = x - x^3$ is attracting from nearby starting values even though $g'(0) = 1$, while the fixed point 0 of

$h(x) = x + x^3$ is repelling even though $h'(0) = 1$.

The principal features of the standard root-finding methods are as follows.

Method	Guaranteed?	Typical speed	Needs f'?
Bisection	Yes	Linear	No
Regula falsi	Yes	Usually linear	No
Newton	No, depends on x_0	Quadratic near simple roots	Yes
Secant	No, depends on x_0, x_1	Order $\frac{1+\sqrt{5}}{2}$	No
Fixed point	Depends on g	Usually linear	No

Appendix: Lecture and Tutorial Index