COMP0043 Numerical Methods for Finance

1.2 Numerical errors: truncation, discretisation and rounding; stability and conditioning; grids and histograms

Guido Germano

Department of Computer Science, University College London

3 October 2024

Truncation, discretisation and grids

Many mathematical problems are continuous, i.e. expressed with real numbers $x \in \mathbb{R}$ which cannot be represented exactly on a digital computer.

We must **truncate** to a finite interval $[a, b] \subset \mathbb{R}$ introducing a lower (or left) truncation limit a and an upper (or right) truncation limit b.

We must then **discretise** introducing N intervals with step

$$\Delta x = \frac{b - a}{N}. \tag{1}$$

This results in a **grid** with N points on the left of each interval,

$$x_n = a + n\Delta x, \quad n = 0, \dots, N - 1. \tag{2}$$

The grid points can be collected in the grid vector (a one-dimensional array)

$$\mathbf{x} = (x_0, x_1, \dots, x_{N-1}). \tag{3}$$

Sometimes one may wish to use a grid with $N + 1$ points including $x_N = b$.

These are **equispaced** grids where the step Δx is constant.

There are also **adaptive** grids where the step Δx_n is variable, but we will not use them.

Building a grid

Each programming language provides commands to build a grid. In Matlab there are e.g.

1. The colon operator `:`, which takes as inputs a, b, Δx.
2. The `linspace` function, which takes as inputs a, b, N.

Typically, our intervals $[a, b]$ are centred around zero, i.e. $a = -b$, and have 0 as a grid point. Alternatively, the grid may be shifted by $\Delta x/2$ with respect to 0.

Truncation, discretisation and rounding errors

Truncation and discretisation cause errors. Numerical methods are characterised by how these errors behave (or scale) for $N \to \infty$ with

1. Δx fixed: **truncation error**, where $a, b \to \infty$, or
2. a, b fixed: **discretisation error**, where $\Delta x \to 0$.

If a problem is naturally expressed on a finite and discrete set of numbers to start with, it will not be affected by either error.

Whereas truncation and discretisation errors are caused by the mathematical method used to approximate a continuous problem with a discrete one, a further source of error is the finite precision with which a digital computer stores a real number in **floating-point representation**,

$$x = (-1)^S \times M \times b^{E-e}, \tag{4}$$

where S is the sign bit, M is an integer mantissa, b is an integer basis (usually 2), E is an integer exponent, and e is a fixed integer “bias” of the exponent so that the smallest non-zero representable number has $E = 1$. Except for $x = 0$, the mantissa is normalised so that its highest-order bit is one, so the leading bit is not stored and called also “hidden”, “phantom” or “implicit”; the other bits are called the fraction F, i.e. $M = 1F$.

Floating-point arithmetic is not exact and results in a **rounding error**.

Machine accuracy, stability and conditioning

The **machine accuracy** ϵ_m (or machine epsilon) is the smallest positive floating-point number that, when added to the floating-point number 1.0, produces a floating-point result different from 1.0.

Roughly speaking, the machine accuracy is the fractional accuracy to which floating-point numbers are represented, corresponding to a change of one in the least significant bit of the mantissa. An arithmetic operation among numbers in floating-point representation introduces a fractional rounding (or roundoff) error of at least ϵ_m.

The smallest floating-point number that can be represented on a machine depends on how many bits there are in the exponent. The machine epsilon ϵ_m is different and depends on how many bits there are in the mantissa.

Rounding errors accumulate with an increasing amount of calculations. If in doing so they cancel each other, the algorithm is **stable**; if they magnify, it is **unstable**. In the latter case, a small change in the input produces a large change in the output.

Even with a stable algorithm, a small change in the input may amplify to an inaccurate solution if the underlying mathematical problem is **ill conditioned**. The (relative) condition number κ of a problem is the ratio of the relative change in the solution to the relative change in the input. A problem is well conditioned if $\kappa \approx 1$ and ill conditioned if $\kappa \gg 1$.

Floating-point representations

All modern processors share the same floating-point representation specified by the IEEE 754 standard of 1985. Minor revisions were done in 2008 and 2019.

For 32-bit “single precision” float values, 1 bit is used for the sign, 8 bits for the exponent (with bias $e = 127$), and 23 bits for the fraction; the machine accuracy is $\epsilon_\mathrm{m} = 2^{-23} = 10^{-23 \log_{10} 2} \approx 10^{-23\times 0.301} \approx 1.19 \times 10^{-7} = 1.19\mathrm{E}{-7}$.

For 64-bit “double precision” values, E has 11 bits (with bias $e = 1023$) and F has 52 bits; $\epsilon_\mathrm{m} = 2^{-52} = 10^{52 \log_{10} 2} \approx 10^{-52\times 0.301} \approx 2.22 \times 10^{-16} = 2.22\mathrm{E}{-16}$.

There is also quadruple precision and, since 2008, half precision, which was introduced mainly for the reduced accuracy requirements of machine learning.

The IEEE 754 standard includes ± 0, which are computationally equivalent, $\pm\infty$, and NaN, i.e. “not a number”, which results from an undefined operation like 0/0 or $\infty - \infty$.

There are several proposals to improve IEEE 754, e.g. John Gustafson’s universal number (unum) format, where the sizes of the exponent and the mantissa are adaptive. Besides two additional fields that give the number of bits of E and F, there is a “u-bit” that marks whether a number is exact or in between exact values. However, the ubiquitous adoption of IEEE 754 for almost 40 years makes it difficult to reach a consensus on how to replace it.

For further reading see Section 1.1 (pages 8–12) of Numerical Recipes.

Goal of numerical analysis

Truncation/discretisation errors and ill conditioning are mathematical and would exist even on a perfect computer with an infinitely accurate representation of real numbers and thus no rounding error and no instability.

In some cases the condition number, especially of a matrix, can be reduced with a transformation called preconditioner.

Usually truncation/discretisation errors do not interact much with the rounding error. A calculation can be imagined as being affected by the truncation/discretisation error and the conditioning that it would have on an infinite-precision computer, "plus" the rounding error associated with the number of performed operations and the stability of the algorithm.

The goal of numerical analysis is to design stable algorithms that minimise the truncation and discretisation errors and the condition number with a parsimonious use of computational resources, i.e. CPU time, memory and, for distributed computing, communication.

Histograms

A histogram gives an approximate representation of the empirical distribution of data.

Its name and current meaning were introduced by the statistician Karl Pearson (UCL) in 1891–1895. Before that, histograms had been used by the Belgian statistician and social scientist Adolphe Quetelet aroud 1846 and by the Scottish engineer and economist William Playfair in a book of 1786 where he also introduced bar and pie charts.

Given N_{sample} data, they are sorted into $N \ll N_{\text{sample}}$ bins (or buckets or classes). The bins form a grid where the bin width corresponds to the grid step.

Several formulas have been worked out to determine the optimal number of bins, e.g. the Sturges rule,

$$N = 1 + \log_2 N_{\text{sample}}$$

or the optimal bin width, e.g. the Scott rule,

$$\Delta x = \frac{3.49\sigma}{\sqrt[3]{N_{\text{sample}}}}.$$

Another important yet easy decision is how to normalise a histogram.

In Matlab, a histogram can be computed and plotted with the function `histogram`.