Lecture 1: Introductory material

Michael S. Floater

September 19, 2018

These notes are based on Chapter 1 of the book.

1 Floating-point arithmetic

We represent a number in floating point, with respect to some base β, using p digits and exponent e. For example, we can represent 0.1 in base 10 as

$$1.0000 \times 10^{-1}.$$

So here, the base is $\beta = 10$, there are $p = 5$ digits, and the exponent is $e = -1$. In general the representation is

$$\pm d_0.d_1 \cdots d_{p-1} \times \beta^e, \qquad 0 \le d_i < \beta,$$

which stands for the number

$$\pm(d_0 + d_1\beta^{-1} + \cdots + d_{p-1}\beta^{-(p-1)}) \times \beta^e.$$

The number is assumed to be *normalized* i.e., we assume $d_0 \neq 0$. The number $d_0.d_1 \cdots d_{p-1}$ is called the *mantissa*.

The largest and smallest allowable exponents are denoted by $e_{\max}$ and $e_{\min}$. If the absolute value of a number is larger than or equal to $\beta \times \beta^{e_{\max}}$ or smaller than $1.0 \times \beta^{e_{\min}}$ then it is *out of range*. We use the terms *overflow* and *underflow* for these numbers.

2 Overflow and underflow

Single-precision floating point format is a representation based on 32 bits. It uses 1 bit for the sign, 8 bits for the exponent, and 23 bits for the (24) digits. The first digit is always 1 and does not need to be stored. The 8 bits for the exponent provide all the integers from 0 and 255, but to obtain negative exponents, we substract 127 so that e is in the range from -127 to 128. In addition, the largest and smallest of these values are reserved for special numbers such as 'NaN' (not a number). So, finally, the exponent e is in the range from $e_{\min}$ to $e_{\max}$ where $e_{\max} = 127$ and $e_{\min} = -126$. In short: $e_{\max} = 2^{8-1} - 1$ and $e_{\min} = -e_{\max} + 1$.

Double-precision floating point format is similar but uses 64 bits instead of 32. It uses 1 bit for the sign, 11 bits for the exponent, and 52 bits for the digits. The maximum and minimum allowable values for the exponent are now $e_{\max} = 2^{11-1} - 1$ and $e_{\min} = -e_{\max} + 1$, i.e., $e_{\max} = 1023$ and $e_{\min} = -1022$.

In matlab:

Find the value of the constant realmax:

```
ndouble = realmax
nsingle = realmax('single')

ndouble =

  1.7977e+308


nsingle =

  3.4028e+38
```

Explanation:

$$3.4028 \times 10^{38} = 1.999\cdots \times 2^{127},$$
$$1.7977 \times 10^{308} = 1.999\cdots \times 2^{1023}.$$

Find the value of the constant realmin:

```
ndouble = realmin
nsingle = realmin('single')
```

```
ndouble =

  2.2251e-308


nsingle =

  1.1755e-38
```

Explanation:

$$1.1755 \times 10^{-38} = 1.0 \cdots \times 2^{-126},$$
$$2.2251 \times 10^{-308} = 1.0 \cdots \times 2^{-1022}.$$

Overflow is caused by any operation whose result is too large in absolute value to be represented. For example, suppose $\beta = 10$, $p = 3$, and $e_{\max} = 100$. We want to compute $\sqrt{x^2 + y^2}$, where

$$x = 3.00 \times 10^{60}, \qquad y = 4.00 \times 10^{60}.$$

Then x^2, y^2, and $x^2 + y^2$ will all overflow.

Exercise 1.1 *Write a routine that implements the calculation of* $\sqrt{x^2 + y^2}$ *in a way which avoids overflow.*

Underflow is caused by any calculation where the result is too small to be distinguished from zero. In some algorithms it is safe to treat an underflowing value as zero, but not always. For example, if δ_t is a variable time step within a loop that terminates after some fixed time has elapsed, then if δ_t underflows, the calculation may go into an infinite loop. So we need to be aware of such problems when designing algorithms.

3 Absolute error and relative error

Suppose that x, y are real numbers far from overflow or underflow. Let x^* denote the floating-point representation of x. We define the *absolute error* ϵ as

$$\epsilon = x^* - x,$$

and the relative error δ as

$$\delta = \epsilon/x,$$

assuming $x \neq 0$.

The absolute and relative errors are zero if and only if x can be represented exactly in the chosen floating-point representation.

In floating-point arithmetic, relative error is a better measure of error than absolute error. For example, suppose $\beta = 10$ and $p = 3$ and let

$$x = 1.001 \times 10^3 \quad \text{and} \quad y = 1.001 \times 10^0.$$

Then

$$x^* = 1.00 \times 10^3 \quad \text{and} \quad y^* = 1.00 \times 10^0.$$

The absolute errors of x^* and y^* are quite different:

$$\epsilon_x = 0.001 \times 10^3 = 1 \quad \text{and} \quad \epsilon_y = 0.001 \times 10^0 = 0.001.$$

Howevever, the relative errors are the same:

$$\delta_x = \frac{\epsilon_x}{x} \approx 0.999 \times 10^{-3} \quad \text{and} \quad \delta_y = \frac{\epsilon_y}{y} \approx 0.999 \times 10^{-3}.$$

4 Forward and backward error analysis

Forward error analysis examines how perturbations of the input propogate. For example, consider the function $f(x) = x^2$. Let $x^* = x(1 + \delta)$ be the representation of x, where δ is the error of x^* relative to x. Then

$$\begin{aligned} f(x^*) = (x^*)^2 &= x^2(1 + \delta)^2 \\ &= x^2(1 + 2\delta + \delta^2) \\ &\approx x^2(1 + 2\delta), \end{aligned}$$

Thus the error of $(x^*)^2$ is approximately double the error of x^*.

Forward error analysis often leads to pessimistic overestimates of the error, especially when a sequence of calculations is performed. Sometimes errors average out, since an error in one calculation can get reduced by an error of opposite sign in the next calculation. This is one reason to prefer backward error analysis. Another is that it is often easier than forward error analysis.

Backward error analysis examines the question: How much error in input would be required to explain all output error? It assumes that an approximate solution to a problem is good if it the exact solution to a nearby problem. Consider the previous example. The output error can be written as

$$[f(x)]^* = (x^2)^* = x^2(1+\rho),$$

where ρ is the relative error of the output. Since ρ is small, there is some $\tilde{\rho}$ such that $(1+\tilde{\rho})^2 = 1+\rho$, and $|\tilde{\rho}| \approx |\rho|/2$. Now

$$\begin{aligned}[f(x)]^* &= x^2(1+\tilde{\rho})^2 \\ &= f[x(1+\tilde{\rho})].\end{aligned}$$

5 Loss of significance

Suppose

$$\begin{aligned}x_1^* &= x_1(1+\rho_1), \\ x_2^* &= x_2(1+\rho_2).\end{aligned}$$

Then

$$\begin{aligned}x_1^* \times x_2^* &\approx (x_1 \times x_2)(1+\rho_1+\rho_2), \\ x_1^*/x_2^* &\approx (x_1/x_2)(1+\rho_1-\rho_2),\end{aligned}$$

and thus the relative errors of multiplication and division are no worse than $|\rho_1|+|\rho_2|$. Similarly, the relative error of computing x_1^n for some integer n is no worse than $n|\rho_1|$.

On the other hand, the relative error of addition and subtraction can be significant. Since

$$\begin{aligned}x_1^* + x_2^* &= x_1 + x_2 + (\rho_1+\rho_2), \\ x_1^* - x_2^* &= x_1 - x_2 + (\rho_1-\rho_2),\end{aligned}$$

the absolute errors are bounded by $|\rho_1|+|\rho_2|$. However, consider the relative error of the addition,

$$\frac{\rho_1+\rho_2}{x_1+x_2}.$$

If x_1 and x_2 are two similar numbers of opposite sign, the relative error could be very large compared to the size of ρ_1 and ρ_2. This problem is known as *loss of significance*. The same thing occurs with subtraction when x_1 and x_2 are similar numbers of the same sign, since the relative error is then

$$\frac{\rho_1 - \rho_2}{x_1 - x_2}.$$

6 Robustness

An algorithm is *robust* if for any valid data input which is reasonably representable, it completes successfully. This is often achieved at the expense of time. For example, consider computing the roots of the quadratic equation

$$ax^2 + bx + c = 0.$$

The formula for one of the roots is

$$x = \frac{-b + \sqrt{b^2 - 4ac}}{2a}.$$

This will not be a robust algorithm (formula) if, for example, b is much larger than a and c, due to loss of significance in the numerator. In this case, a robust alternative formula is

$$x = \frac{-2c}{b + \sqrt{b^2 - 4ac}}.$$

7 Error testing and rate of convergence

Many algorithms find an approximation to the solution by iteration. Consider a sequence of iterations $x_0, x_1, x_2, \ldots$ to the true solution x. We define the absolute error of the n-th iteration as

$$\epsilon_n = x_n - x.$$

The sequence *converges* to the limit x if

$$\lim_{n \to \infty} \epsilon_n = 0.$$

It is typical to use either an absolute target accuracy ϵ_t and terminate the iteration when

$$|\epsilon_n| \leq \epsilon_t,$$

or a relative target accuracy δ_t and terminate the iteration when

$$\frac{|\epsilon_n|}{|x_n|} \leq \delta_t.$$

Alternatively, one could use a combination of the two stopping citeria and terminate when

$$|\epsilon_n| \leq \eta_t(1 + |x_n|),$$

for some target accuracy η_t.

However, x is unknown, and so we can only estimate ϵ_n. A simple approach is to estmate ϵ_n by

$$\epsilon_n \approx x_n - x_{n-1},$$

although it might be far from the real error. An improvement might be

$$\epsilon_n \approx |x_n - x_{n-1}| + |x_{n-1} - x_{n-2}|.$$

However, in many problems, convergence can be tested independently, for example, when the inverse of a function can be easily calculated (calculating the square instead of the square root, and so on).

The *rate of convergence* is p, $p \geq 1$, if there is a constant C such that

$$\lim_{n\to\infty} \left| \frac{\epsilon_{n+1}}{\epsilon_n^p} \right| = C.$$

This is often expressed as

$$|\epsilon_{n+1}| = O(|\epsilon_n|^p),$$

or, in other words,

$$|\epsilon_{n+1}| \approx C|\epsilon_n|^p,$$

for sufficiently large n. Thus the error in the next iteration is approximately the p-th power of the current iteration error times a constant C. For $p = 1$ we need to have $C < 1$ for convergence. For $p > 1$, this is not necessary. Convergence with $p = 1$ is known as *linear convergence*. Convergence with $p = 2$ is known as *quadratic convergence*. Convergence with $1 < p < 2$ is known as *superlinear convergence*.

8 Computational complexity

We would prefer an algorithm that is not only robust and have a fast rate of convergence, but it should also have a reasonable *computational complexity*, so that the algorithm does not become too slow for large problems.

For example, consider matrix calculations. Suppose $A = (A_{ij})$ and $B = (B_{ij})$ are two $n \times n$ matrices and we want to compute their matrix product $C = (C_{ij})$. For each element in C, we have to calculate

$$C_{ij} = \sum_{k=1}^{n} A_{ik} B_{kj},$$

which requires n multiplications and $n-1$ additions. Since there are n^2 elements in C, the total number of operations required is n^3 multiplications and $n^2(n-1)$ additions. Counting multiplications and additions (and divisions and subtractions) as flops (floating-point operations) we have a total of

$$n^3 + n^2(n-1) \quad \text{flops.}$$

We are mainly concerned here with the computational complexity when n is large. Since

$$n^3 + n^2(n-1) \leq 2n^3$$

it follows that the computational complexity is $O(n^3)$ as $n \to \infty$.

Exercise 1.3 *Let A be an $n \times n$ nonsingular band matrix that satisfies the condition $A_{ij} = 0$ for $|i-j| > r$, where r is small, and suppose Gauss elimination is used to solve $A\mathbf{x} = \mathbf{b}$. Deduce that the total number of additions and multiplications required is bounded by a constant multiple of nr^2.*

9 Condition

The condition of a problem is inherent to the problem independent of the algorithm used to solve it. The *condition number* of a numerical problem measures the asymptotically worst case of how much the outcome can change in proportion to small pertubations in the input data.

- A problem with a low condition number is *well-conditioned*.
- A problem with a high condition number is *ill-conditioned*.

9.1 Function evaluation

Consider the problem of evaluating a function f at a point x, and let $\hat{x}$ be a point close to x. The relative change in x is

$$\frac{\hat{x} - x}{x},$$

and the corresponding relative change in $f(x)$ is

$$\frac{f(\hat{x}) - f(x)}{f(x)}.$$

The condition number of the evaluation is the limit of the relative change in $f(x)$ divided by the relative change in x as $\hat{x}$ tends to x. If f is differentiable we find

$$K(x) = \lim_{\hat{x} \to x} \left| \frac{f(\hat{x}) - f(x)}{f(x)} \right| \Big/ \left| \frac{\hat{x} - x}{x} \right| = \left| \frac{x f'(x)}{f(x)} \right|.$$

For example, for $f(x) = \sqrt{x}$, we find $K(x) = 1/2$ for all x, and so the conditioning is the same for all x: the perturbation of the output is half of the perturbation of the input. On the other hand, if $f(x) = 1/(1-x)$ then

$$K(x) = \left| \frac{x}{1-x} \right|,$$

and so the evaluation is ill-conditioned for x close to 1. For example, if $x = 1.000001$ then $K(x) = 1.000001 \times 10^6$.

Exercise 1.5 *Examine the condition of evaluating* $\cos x$.

9.2 Solving a linear system

Suppose next we want to solve the linear system $A\mathbf{x} = \mathbf{b}$ where $A = (A_{ij})$ is an $n \times n$ matrix. We examine the condition number of A associated with this system. A large condition number means that a small perturbation in the right hand side $\mathbf{b}$ may cause a large error in $\mathbf{x}$.

The condition number is defined to be the maximum ratio of the relative change in $\mathbf{x}$ divided by the relative change in $\mathbf{b}$ over all possible $\mathbf{b}$. Let $\mathbf{e}$ be a perturbation in $\mathbf{b}$. Then the relative change in $\mathbf{b}$ is

$$\|\mathbf{e}\|/\|\mathbf{b}\| = \|\mathbf{e}\|/\|A\mathbf{x}\|,$$

where $\|\cdot\|$ denotes some *vector norm*. We will see below how different vector norms lead to different condition numbers.

Assuming that A is invertible, then $\mathbf{x} = A^{-1}\mathbf{b}$ and the perturbation in $\mathbf{x}$ is

$$A^{-1}(\mathbf{b} + \mathbf{e}) - A^{-1}\mathbf{b} = A^{-1}\mathbf{e}.$$

Thus the relative change in $\mathbf{x}$ is

$$\|A^{-1}\mathbf{e}\|/\|\mathbf{x}\|.$$

Hence the condition number is

$$K(A) = \max_{\mathbf{x},\mathbf{e}\neq 0} \frac{\|A^{-1}\mathbf{e}\|}{\|\mathbf{x}\|} \Big/ \frac{\|\mathbf{e}\|}{\|A\mathbf{x}\|} = \max_{\mathbf{e}\neq 0} \frac{\|A^{-1}\mathbf{e}\|}{\|\mathbf{e}\|} \times \max_{\mathbf{x}\neq 0} \frac{\|A\mathbf{x}\|}{\|\mathbf{x}\|}. \tag{1}$$

Recall that the *matrix norm* $\|\cdot\|$ derived from the vector norm $\|\cdot\|$ is defined to be

$$\|A\| = \max_{\mathbf{v}\neq 0} \frac{\|A\mathbf{v}\|}{\|\mathbf{v}\|}.$$

Hence, we conclude that

$$K(A) = \|A^{-1}\| \times \|A\|.$$

The smallest possible value of the condition number is $K(A) = 1$ which occurs when A is the identity matrix. In general, $K(A) \geq 1$, and the conditioning of A is worse the larger $K(A)$ is.

9.3 The 2-norm

Recall that the *2-norm* or *Euclidean norm* of a vector $\mathbf{x} = (x_1, \ldots, x_n)$ is

$$\|\mathbf{x}\|_2 = \left(\sum_{i=1}^{n} x_i^2 \right)^{1/2}.$$

Then the induced matrix norm is

$$\|A\|_2 = \sigma_{\max}(A),$$

and

$$K_2(A) = \frac{\sigma_{\max}(A)}{\sigma_{\min}(A)},$$

where $\sigma_{\max}(A)$ and $\sigma_{\min}(A)$ are the maximal and minimal singular values of A, respectively. Recall that the singular values of a matrix A are the square roots of the eigenvalues of the matrix A^*A where A^* denotes the complex conjugate transpose of A. In particular, A is a *normal matrix* if $A^*A = AA^*$. For example, a symmetric matrix is normal. If A is normal then

$$K_2(A) = \frac{\lambda_{\max}(A)}{\lambda_{\min}(A)},$$

where $\lambda_{\max}(A)$ and $\lambda_{\min}(A)$ are the maximal and minimum modulus eigenvalues of A. The matrix A is *unitary* if $A^*A = I$, in which case $K_2(A) = 1$.

9.4 The ∞-norm

The ∞-*norm* of the vector $\mathbf{x}$ is

$$\|\mathbf{x}\|_\infty = \max_{i=1,\ldots,n} |x_i|,$$

and the induced matrix norm is

$$\|A\|_\infty = \max_{i=1,\ldots,n} \sum_{j=1}^{n} |A_{ij}|,$$

the maximum row sum of A.

9.5 Example

Consider the linear system

$$\begin{bmatrix} 404 & 60 \\ 60 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}.$$

The matrix has eigenvalues $\lambda_1 \approx 412.8$ and $\lambda_2 \approx -4.8$. Suppose that $\mathbf{e}$ in (1) is the eigenvector belonging to λ_2. Then

$$\frac{\|A^{-1}\mathbf{e}\|}{\|\mathbf{e}\|} = \frac{\|\lambda_2^{-1}\mathbf{e}\|}{\|\mathbf{e}\|} = |\lambda_2|^{-1}.$$

Conversely, suppose that $\mathbf{x}$ in (1) is the eigenvector belonging to λ_1. Then

$$\frac{\|A\mathbf{x}\|}{\|\mathbf{x}\|} = \frac{\|\lambda_1\mathbf{x}\|}{\|\mathbf{x}\|} = |\lambda_1|.$$

This is the worst-case scenario and we find

$$K(A) = \frac{|\lambda_1|}{|\lambda_2|} \approx 85.9.$$

Exercise 1.7 *Let*

$$A = \begin{bmatrix} 1000 & 999 \\ 999 & 998 \end{bmatrix}.$$

Calculate A^{-1}*, the eigenvalues and eigenvectors of* A*,* $K_2(A)$*, and* $K_\infty(A)$*. What is special about the vectors* $[1,1]^T$ *and* $[-1,1]^T$*?*

An example of a notoriously ill-conditioned matrix is the Hilbert matrix $H = (H_{ij})$, whose entries are

$$H_{ij} = \frac{1}{i+j-1}.$$

If n is the dimension of the matrix, the condition number is of order $O((1+\sqrt{2})^{4n}/\sqrt{n})$ as $n \to \infty$.