A Summary for Elementary Numerical Methods

Ya Yan Lu

Department of Mathematics
City University of Hong Kong

1 Floating Point Arithmetic

The **floating point numbers** used in a programming language such as C, C++ or FORTRAN usually have a single or double precision. A single precision floating point number (i.e. float in C) has 4 bytes or 32-bits, arranged in the following

$$s \; e_7 e_6 ... e_1 e_0 \; m_1 m_2 ... m_{23},$$

where s, e_0, ..., e_7, m_1, ..., m_{23} are 1 or 0. This is defined (in the so-called IEEE standard 754) as

$$x = (-1)^s \times \left(1 + \frac{m_1}{2} + \frac{m_2}{2^2} + ... + \frac{m_{23}}{2^{23}}\right) \times 2^q \tag{1}$$

where q is an integer. More precisely

$$q = (e_7 e_6 ... e_0)_2 - 127 = e_0 + 2e_1 + 2^2 e_2 + ... + 2^7 e_7 - 127.$$

We can also use the binary number notation:

$$(1.m_1 m_2 ... m_{23})_2 = 1 + \frac{m_1}{2} + \frac{m_2}{2^2} + ... + \frac{m_{23}}{2^{23}}.$$

Thus,

$$x = \pm (1.m_1 m_2 ... m_{23})_2 \times 2^q. \tag{2}$$

Let x be a real number, we denote by $fl(x)$ the nearest floating point number. The process from x to $fl(x)$ is called **rounding**. When $fl(x)$ is used as an approximation to x, we have some small error. This error is called the **round-off** error. We define

$$\text{absolute error} = |x - fl(x)|, \quad \text{relative error} = \left|\frac{x - fl(x)}{x}\right|.$$

For the single precision floating point number system defined in (1), if x is a real number and $fl(x)$ is the nearest floating point number, then

$$\frac{|x - fl(x)|}{|x|} \le \frac{1}{2^{24}} = \epsilon_M. \tag{3}$$

Note: We usually call $\epsilon_M = 1/2^{24}$ the machine epsilon, or unit round-off. A alternative form for the above result is:

$$fl(x) = x(1+\epsilon) \quad \text{for} \quad |\epsilon| \leq \epsilon_M. \tag{4}$$

When we try to add two floating point numbers together, the exact result is usually not a floating point number. More precisely, assume x and y are floating point numbers, thus, $x = fl(x)$ and $y = fl(y)$, then $x+y$ usually is not a floating point number. We assume that our computer will do its best. That is, it will give us $fl(x+y)$. We have the following fundamental assumption on floating point arithmetics: *Let x and y be floating point numbers, $\odot$ denotes $+$, $-$, $\times$ or $\div$, then*

$$\textit{Computer Result of } x \odot y = fl(x \odot y).$$

In other words, the computer result is the nearest floating point number of the exact answer. Therefore, after every arithmetic operation, we have a round-off error.

2 Nonlinear Equations

In this section, we study some numerical methods for solving

$$f(x) = 0$$

where f is a function of x. The **bisection method** starts with two numbers a and b, assuming $f(a)$ and $f(b)$ have opposite signs. Under the assumption that f is continuous, we know that the interval (a, b) must contain a zero of f. Now, we consider the mid-point c of the interval (a, b),

$$c = \frac{a+b}{2},$$

and find $f(c)$. If $f(c) = 0$, we find the solution. Otherwise, either (a, c) or (c, b) must contain the zero of f. This can be decided by comparing the sign of $f(c)$ with that of $f(a)$ or $f(b)$. If $f(c)$ has the same sign as $f(a)$, then $f(c)$ must have an opposite sign as $f(b)$, therefore, the interval (c, b) contains a zero of f. In this case, we define the new a as c (and keep b unchanged), then repeat the process. Similarly, if $f(c)$ has the same sign as $f(b)$, we will replace the original interval (a, b) by (a, c), because (a, c) contains a zero of f. In this case, we will define the new b as c and repeat the process. In each step, we have an interval containing the zero of f, then determins an interval of half size that still contains the zero of f. The process is repeated, untill the length of the interval is small enough. The mid-point of the intervals serve as our approximate solutions.

Newton's method is a very efficient method for solving $f(x) = 0$. But, it is necessary to know the derivative of $f(x)$. You also need to provide an initial guess of the exact solution. If the initial guess is not good, the method may fail. Starting from an initial guess x_0, Newton's method obtains x_1, x_2, ..., recursively by the formula:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}. \tag{5}$$

This gives rise to a sequence of approximate solutions and we hope $\lim x_k = x_*$ as $k \to \infty$. The fast convergence of Newton's method is a result of its quadratic convergence. That is:

$$e_{k+1} \approx \lambda\, e_k^2, \qquad \text{where}$$

$$e_k = |x_k - x_*|, \quad e_{k+1} = |x_{k+1} - x_*|, \quad \lambda = \left| \frac{f''(x_*)}{2f'(x_*)} \right|.$$

The **secant method** can be applied when the derivative of $f(x)$ is not available. However, it needs two initial guesses. If the initial guesses are not good, the method may fail. Starting from x_0, x_1, to find x_* such that $f(x_*) = 0$, we obtain the sequence (of approximate solutions) by:

$$x_{k+1} = x_k - \frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})} f(x_k). \tag{6}$$

For the secant method, using Taylor expansion at x_*, the following relationship can be obtained:

$$x_{k+1} - x_* \approx \frac{f''(x_*)}{2f'(x_*)}(x_k - x_*)(x_{k-1} - x_*).$$

This leads to:

$$|x_{k+1} - x_*| \approx \lambda |x_k - x_*|^\alpha,$$

where $\alpha = (1 + \sqrt{5})/2 \approx 1.618$ and λ is some constant related to $f''(x_*)$ and $f'(x_*)$.

3 Polynomial and Spline Interpolation

The problem of **polynomial interpolation** is: Given $n+1$ points (x_j, y_j), $j = 0, 1, ..., n$, find a polynomial $P_n(x)$, with the degree of $P_n \le n$, such that

$$P_n(x_j) = y_j, \quad \text{for} \quad j = 0, 1, ..., n.$$

We must require that the x_j values are different for different j. Since we require that the degree of P_n is n or less, there is only one polynomial satisfying the conditions. This is the so-called **uniqueness**. The polynomial P_n has an explicit formula called **Lagrange interpolation formula**.

$$P_n(x) = \sum_{j=0}^{n} y_j \prod_{k=0..n}^{k \neq j} \frac{(x - x_k)}{(x_j - x_k)}.$$

For the given $n+1$ points, we can also look for a function $S(x)$ which is a cubic polynomial of x on each interval $[x_j, x_{j+1}]$ for $j = 0, 1, .., n-1$. In other words:

$$S(x) = \begin{cases} S_1(x) & x_0 \le x < x_1 \\ S_2(x) & x_1 \le x < x_2 \\ ... & \\ S_n(x) & x_{n-1} \le x \le x_n \end{cases} \tag{7}$$

where $S_j(x)$ is a cubic polynomial. The function S and its derivatives S', S'' are required to be continuous. We can write down the conditions:

$$S_j(x_{j-1}) = y_{j-1}, \quad S_j(x_j) = y_j, \quad j = 1, 2, ..., n.$$
$$S_j'(x_j) = S_{j+1}'(x_j), \quad S_j''(x_j) = S_{j+1}''(x_j), \quad j = 1, 2, ..., n-1.$$

The total number of conditions is $4n-2$. For the so-called **natural cubic spline**, we add two more conditions:

$$S''(x_0) = S''(x_n) = 0.$$

For the cubic polynomial S_j, we already know its values at x_{j-1} and x_j. If we also know its second derivative at x_{j-1} and x_j, then we can determine S_j completely. Let $S_j(x)$ be a cubic polynomial of x satisfying $S_j(x_{j-1}) = y_{j-1}$, $S_j(x_j) = y_j$, $S_j''(x_{j-1}) = y_{j-1}''$ and $S_j''(x_j) = y_j''$, then

$$S_j(x) = Ay_{j-1} + By_j + \frac{h_j^2}{6}\left[(A^3 - A)y_{j-1}'' + (B^3 - B)y_j''\right] \tag{8}$$

where

$$h_j = x_j - x_{j-1}, A = \frac{x - x_j}{x_{j-1} - x_j} = -\frac{x - x_j}{h_j}, B = \frac{x - x_{j-1}}{x_j - x_{j-1}} = \frac{x - x_{j-1}}{h_j} = 1 - A.$$

In reality, we do not know its second derivative, but we can solve y_1'', ..., y_{n-1}'' first. The condition $S_j'(x_j) = S_{j+1}'(x_j)$ (for $j = 1, 2, ..., n-1$) can be reduced to

$$h_j y_{j-1}'' + 2(h_j + h_{j+1})y_j'' + h_{j+1}y_{j+1}'' = 6\left[\frac{y_{j+1} - y_j}{h_{j+1}} - \frac{y_j - y_{j-1}}{h_j}\right]. \tag{9}$$

Since $y_0'' = y_n'' = 0$, when the above $n-1$ equations are written down, we have a linear system for the $n-1$ unknowns: y_1'', ..., y_{n-1}''. Once they are solved, we can use the earlier formula for the cubic polynomials.

4 Numerical Integration

In this section, we will consider the definite integral

$$I = \int_a^b f(x)\,dx,$$

where a and b are constants, f is a given function. For an integer n, we define

$$h = \frac{b-a}{n}, \quad x_0 = a, \quad x_j = a + jh.$$

The composite trapezoidal rule is

$$\int_a^b f(x)dx \approx h\left[\frac{1}{2}f(x_0) + f(x_1) + ... + f(x_{n-1}) + \frac{1}{2}f(x_n)\right]$$

To have some idea about the accuracy of this method, we assume f has continuous second order derivative on $[a, b]$, then there is a number $\xi \in (a, b)$, such that

$$\int_a^b f(x)dx - h\left[\frac{1}{2}f(x_0) + f(x_1) + ... + f(x_{n-1}) + \frac{1}{2}f(x_n)\right] = -\frac{(b-a)^3}{12n^2}f''(\xi),$$

where $h = (b-a)/n$, $x_j = a + jh$. The **composite Simpson's rule** requires an even integer n. It is given as

$$\int_a^b f(x)dx \approx h\left[\frac{1}{3}f(x_0) + \frac{4}{3}f(x_1) + \frac{2}{3}f(x_2) + ... + \frac{2}{3}f(x_{N-2}) + \frac{4}{3}f(x_{N-1}) + \frac{1}{3}f(x_N)\right]$$

If f has a continuous fourth order derivative on (a,b), then there is a number $\xi \in (a,b)$, such that

$$\int_a^b f(x)dx - \frac{h}{3}[f(x_0) + 4f(x_1) + ... + 4f(x_{n-1}) + f(x_n)] = -\frac{(b-a)^5}{180n^4} f^{(4)}(\xi),$$

where n is even, $h = (b-a)/n$ and $x_j = a + jh$ for $j = 0, 1, ..., n$. Notice that the Simpson's method has a fourth order of accuracy. That is, the error decreases at $1/n^4$ as n increases. The trapezoidal rule has a second order of accuracy.

The n-point **Gauss-Legendre quadrature** formula is:

$$\int_{-1}^{1} f(x) \approx \sum_{k=1}^{n} c_k f(x_k), \tag{10}$$

where $x_1, x_2, ..., x_n$ are zeros of the Legendre polynomial of degree n, c_k is given by

$$c_k = \int_{-1}^{1} \prod_{j=1..n}^{j\neq k} \frac{(x - x_j)}{(x_k - x_j)} dx.$$

The Legendre polynomials are defined by:

$$L_0 = 1, \quad L_1 = x, \quad (n+1)L_{n+1} = (2n+1)xL_n - nL_{n-1}, \quad \text{for} \quad n \geq 1.$$

We have

$$L_2 = \frac{3\,x^2}{2} - 1/2, \quad L_3 = \frac{5\,x^3}{2} - \frac{3\,x}{2}, \quad L_4 = \frac{35\,x^4}{8} - \frac{15\,x^2}{4} + 3/8.$$

For $n = 1$, the Gauss-Legendre formula is

$$\int_{-1}^{1} f(x)dx \approx 2f(0).$$

For $n = 2$, we have

$$\int_{-1}^{1} f(x)dx \approx f\left(\frac{1}{\sqrt{3}}\right) + f\left(-\frac{1}{\sqrt{3}}\right).$$

The 3-point Gauss-Legendre formula is

$$\int_{-1}^{1} f(x)dx \approx \frac{5}{9} f(-\sqrt{0.6}) + \frac{8}{9} f(0) + \frac{5}{9} f(\sqrt{0.6}).$$

The Gauss-Legendre quadrature formula above is only for integrals on $[-1, 1]$. In general, we want the integral on $[a, b]$. This can be achieved by a transformation. We will use t for the variable on $[a, b]$, thus we let

$$t = a + \frac{b-a}{2}(x+1)$$

and then

$$\int_a^b f(t)\ dt = \frac{b-a}{2} \int_{-1}^{1} f\left(a + \frac{b-a}{2}(x+1)\right) dx \approx \frac{b-a}{2} \sum_{k=1}^{n} c_k f\left(a + \frac{b-a}{2}(x_k+1)\right)$$

Since the n-point Gauss-Legendre formula is exact for any polynomial of degree $\leq 2n-1$, as a special case, we have

$$\int_{-1}^{1} x^m dx = \sum_{j=1}^{n} c_j x_j^m, \quad \text{for} \quad m = 0, 1, ..., 2n-1. \tag{11}$$

The above equations gives an alternative approach to obtain the nodes $\{x_k\}$ and coefficients $\{c_k\}$. These are $2n$ equations and we have solve the $2n$ parameters $\{x_k, c_k\}$.

5 Fast Fourier Transform

For a given integer $N > 0$, we consider the relationship:

$$f_k = \sum_{j=0}^{N-1} \hat{f}_j e^{i2\pi jk/N}, \quad \text{for} \quad k = 0, 1, 2, ..., N-1. \tag{12}$$

This is the **discrete Fourier transform** (DFT). Then, we will prove that the following is true:

$$\hat{f}_j = \frac{1}{N} \sum_{k=0}^{N-1} f_k e^{-i2\pi jk/N}. \tag{13}$$

Fast Fourier Transform (FFT) is a fast method to evaluate the discrete Fourier transform or its inverse. For a given N vector $\hat{f}$, we first calculate the discrete Fourier transform of these two vectors:

$$(\hat{f}_0, \hat{f}_2, ..., \hat{f}_{N-2}), \quad \text{and} \quad (\hat{f}_1, \hat{f}_3, ..., \hat{f}_{N-1}).$$

If the results are denoted as α_j, β_j for $j = 0, 1, ..., N/2 - 1$, then

$$\begin{aligned} f_j &= \alpha_j + \omega^j \beta_j \\ f_{N/2+j} &= \alpha_j - \omega^j \beta_j \end{aligned}$$

where $\omega = \omega_N = e^{i2\pi/N}$. If we denote by T_N the total required number of (real) operations for FFT of an N-vector, we have $T_N = 2T_{N/2} + 5N$ and $T_1 = 0$. This lead to $T_N = 5N \log_2 N$.

6 Linear System of Equations

Consider the linear system of equations $Ax = b$, where A is an invertible square matrix, b is a given vector and x is the unknown vector. Our method is a variant of Gaussian elimination. It corresponds to $PA = LU$, where P is a permutation matrix, L is a lower triangular matrix and U is an upper triangular matrix. We call this LU **decomposition with partial pivoting**. The algorithm is as follows:

For $k = 1, 2, ..., n-1$,

Find p_k, such that

$$|a_{p_k,k}| = \max\{|a_{kk}|, |a_{k+1,k}|, ..., |a_{nk}|\}.$$

If $p_k > k$, swap the k-th row with the p_k-th row.

Reset the k-th column for matrix L.

$$a_{ik} := a_{ik}/a_{kk} \quad \text{for} \quad i = k+1, ..., n.$$

Update the trailing $(n-k) \times (n-k)$ matrix:

$$a_{ij} := a_{ij} - a_{ik}a_{kj} \quad \text{for} \quad i, j = k+1, ..., n.$$

Once the decomposition $PA = LU$ is found, the equation $Ax = b$ can be easily solved. The following algorithm overwrites b with the solution x:

Overwrite b by Pb:

$$\text{If } \ p(i) > i, \ \text{swap } \ b_i \text{ with } \ b_{p_i}, \ \text{for } \ i = 1, 2, ..., n-1,$$

Overwrite Pb by y from $Ly = Pb$ using forward substitution.

$$b_i := b_i - a_{i1}b_1 - a_{i2}b_2 - ... - a_{i,i-1}b_{i-1}, \quad \text{for } \ i = 2, 3, ..., n.$$

Overwrite y by x from $Ux = y$ using backward substitution.

$$b_i := [b_i - a_{i,i+1}b_{i+1} - ... - a_{in}b_n]/a_{ii} \quad \text{for } \ i = n, n-1, ..., 1.$$

7 QR Factorization and Least Squares Problems

An orthogonal matrix Q is a real matrix (no complex entries), such that $Q^{-1} = Q^T$. **Householder reflection** is a special orthogonal matrix. For a given column vector $x = (x_1, x_2, ..., x_m)^T$, the Householder reflection is

$$H = I - \frac{2}{v^T v} v v^T, \quad \text{where} \quad v = \begin{bmatrix} x_1 - \sigma \\ x_2 \\ \vdots \\ x_m \end{bmatrix} \quad \text{and} \quad \sigma = \pm ||x||.$$

Given an $m \times n$ matrix (assuming $m \geq n$), we can use Household reflections to find the QR **factorization of** A. That is $A = QR$, where Q is an $m \times m$ orthogonal matrix and R is an $m \times n$ upper triangular matrix. The basic idea is to construct a sequence of Householder reflections such that

$$H_1 A = \begin{bmatrix} * & * & ... & * \\ 0 & * & ... & * \\ \vdots & \vdots & & \vdots \\ 0 & * & ... & * \end{bmatrix}, \quad H_2 H_1 A = \begin{bmatrix} * & * & * & ... & * \\ 0 & * & * & ... & * \\ 0 & 0 & * & ... & * \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & * & ... & * \end{bmatrix},$$

and if $m > n$, we have

$$H_n ... H_2 H_1 A = R = \begin{bmatrix} * & * & ... & * \\ & * & ... & * \\ & & \ddots & \vdots \\ & & & * \\ & & & 0 \end{bmatrix}.$$

where the last 0 means a zero column. Then, $A = QR$, where

$$Q = (H_{n-1} ... H_2 H_1)^{-1} = H_1 H_2 ... H_{n-1}$$

since H_i is symmetric and orthogonal.

Using the QR factorization of A, we can solve the **least squares problem**:

$$\min_x ||Ax - b||,$$

where A is an $m \times n$ (for $m \geq n$) with $rank(A) = n$, b is a column vector of length m and $||v|| = \sqrt{v^T v}$ for a column vector v. We have

$$||Ax - b|| = ||QRx - b|| = ||Q(Rx - Q^T b)|| = ||Rx - Q^T b||.$$

If we write vector $Q^T b$ as

$$Q^T b = \begin{bmatrix} \beta \\ \gamma \end{bmatrix}$$

where β is a vector of length n, then

$$\min_x ||Rx - Q^T b|| = ||\gamma||,$$

and the minimum is obtained for x satisfying $R_1 x = \beta$, where R_1 is the $n \times n$ upper triangular matrix obtained from the first n rows of R.

8 Matrix Eigenvalue Problem

In this section, we consider the matrix eigenvalue problem

$$Ax = \lambda x,$$

where A is a $n \times n$ square matrix. We look for some special number λ, such that a non-zero vector x can be found to satisfy the above equation.

Let λ_1 be the largest eigenvalue (in absolute value) satisfying $|\lambda_1| > |\lambda_j|$ for $j \neq 1$, then the **power method** can be used to calculate the eigenvector corresponding to λ_1.

- Set x_0 as an arbitrary vector (initial guess);
- For $k = 1, 2, ...$, let $z = Ax_{k-1}$ and $x_k = z/||z||$.

If λ_1 is the smallest eigenvalue in absoluate value, i.e. $0 < |\lambda_1| < |\lambda_j|$ for $j \neq 1$, then the **inverse iteration** method can be used to find the eigenvector corresponding to λ_1. It is mathematically equivalent to the power method applied to A^{-1}. But we do not need to calculate A^{-1}.

- Set x_0 as an arbitrary non-zero vector (initial guess).
- For $k = 1, 2, 3, ...$, solve z from $Az = x_{k-1}$ and let $x_k = z/||z||$.

If x is an approximate eigenvector, we can use $s = x^T Ax/(x^T x)$ (the Rayleigh quotient) to approximate the eigenvalue. The **Rayleigh Quotient Iteration** applies the inverse power method to $A - sI$, where s is the Rayleigh quotient (approximate eigenvalue).

- Let $x_0 =$ arbitrary initial guess.

- For $k = 1, 2, 3, ...$, let $s = x_{k-1}^T A x_{k-1} / (x_{k-1}^T x_{k-1})$, solve z from $(A - \lambda I)z = x_{k-1}$, and let $x_k = z/||z||$.

The method has a very fast convergence. But for an arbitrary initial guess x_0, it is not known which eigenvalue/eigenvector pair will it converge to.

If we want to find all eigenvalues and eigenvectors of a real symmetric matrix A. We first find a symmetric tridiagonal matrix T and an orthogonal matrix Q such that $A = QTQ^T$, then find the eigenvalues and eigenvectors of T. For the **tridiagonal reduction** step $A = QTQ^T$, we use Householder reflections. For $n = 4$, we have

$$H_1 A H_1^T = \begin{bmatrix} * & * & 0 & 0 \\ * & * & * & * \\ 0 & * & * & * \\ 0 & * & * & * \end{bmatrix}, \quad H_2 H_1 A H_1^T H_2^T = \begin{bmatrix} * & * & 0 & 0 \\ * & * & * & 0 \\ 0 & * & * & * \\ 0 & 0 & * & * \end{bmatrix} = T,$$

where H_1 and H_2 are Household reflections. The matrix Q is given by $Q = H_1 H_2$.

To find the eigenvalues and eigenvectors of T, we use the so-called **QR method**. The main step is to repeatedly

- find an approximate eigenvalue s (s is called a shift),
- find the QR factorization of $T - sI$, i.e., $T - sI = QR$,
- find the new T by $T := sI + RQ$.

The new T is not the same as the original T, but it is still symmetric tridiagonal, and it has the same eigenvalues as the original T. If the original T is

$$T = \begin{bmatrix} \alpha_1 & \beta_1 & & \\ \beta_1 & \ddots & \ddots & \\ & \ddots & \alpha_{n-1} & \beta_{n-1} \\ & & \beta_{n-1} & \alpha_n \end{bmatrix}$$

Then, we choose the shift s as an eigenvalue of $\begin{pmatrix} \alpha_{n-1} & \beta_{n-1} \\ \beta_{n-1} & \alpha_n \end{pmatrix}$. Since this matrix has two eigenvalues, we choose s to be the eigenvalue that is closer to α_n. This is called **Wilkinson's shift**. With this kind of shift (and some additional details), the matrix T will converges to a diagonal matrix of the eigenvalues.