# MA580: Numerical Analysis: I

Zhilin Li

# Chapter 1

# Introduction

- Why is this course important (motivations)? What is the role of this class in the problem solving process using mathematics and computers?
- Model problems and relations with course materials.
- Errors (definition and how to avoid them)

In Fig.1.1, we show a flow chart of a problem solving process. In this class, we will focus on numerical solutions using computers, especially the problems in linear algebra. Thus this course can also be called "Numerical Linear Algebra".

## 1.1 Model problems: Data fitting and interpolation

Give data

| | | | |
|---|---|---|---|
| $t_0$ | $t_1$ | $\cdots$ | $t_m$ |
| $y_0$ | $y_1$ | $\cdots$ | $y_m$ |

(1.1.1)

The data can be taken from a function $y(t)$, or a set of observed data. We want to find a simple function $y_a(t)$ to approximate $y(t)$ so that we can predict $y(t)$ everywhere.

**Approach I: Polynomial approximation.**

Let

$$y(t) \approx a_0 + a_1 t + a_2 t^2 + \cdots + a_n t^n. \tag{1.1.2}$$

We need to find the coefficients $a_0, a_1, a_2, \cdots, a_n$ so that we can have an analytic expression.

$n = 0$ : constant approximation
$n = 1$ : linear regression
$n = 2$ : quadratic regression
... ...

We should choose the coefficients in such a way that

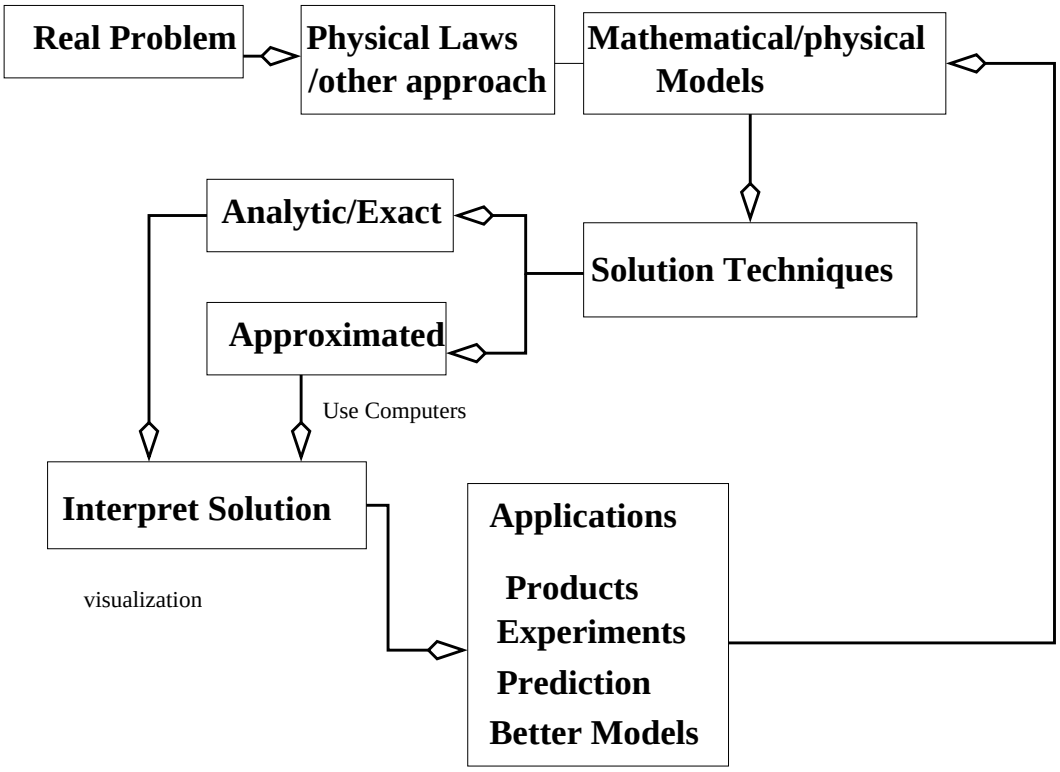


Figure 1.1: A flow chart of a problem solving process.

they can match the data at the sample points[1]. Thus we have

$$\begin{array}{cc} t=t_0: & \left\{ \begin{array}{c} a_0+a_1t_0+a_2t_0^2+\cdots+a_nt_0^n=y_0 \\ a_0+a_1t_1+a_2t_1^2+\cdots+a_nt_1^n=y_1 \\ \cdots\cdots\cdots \\ a_0+a_1t_m+a_2t_m^2+\cdots+a_nt_m^n=y_m \end{array} \right. \\ t=t_1: & \\ \cdots & \\ t=t_m: & \end{array} \tag{1.1.3}$$

This is a linear system of equations for the unknown coefficients $a_i$, $i = 0, \cdots, n$. In the matrix-vector form, it is

$$\begin{pmatrix} 1 & t_0 & t_0^2 & \cdots & t_0^n \\ 1 & t_1 & t_1^2 & \cdots & t_1^n \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & t_m & t_m^2 & \cdots & t_m^n \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_m \end{pmatrix}. \tag{1.1.4}$$

We can simply write it as $Ax = b$, where $A$ is an $(m+1) \times (n+1)$ matrix, and $x = \begin{bmatrix} a_0 & a_1 & \cdots & a_n \end{bmatrix}^T$ is an $n$ by 1 column vector, and $\begin{bmatrix} y_0 & y_1 & \cdots & y_m \end{bmatrix}^T$ is an $(m+1)\times 1$ column vector. We distinguish the following cases assuming that $t_i$ are distinct ($t_i \neq t_j$).

- $m > n$, that is, we have more equations than the unknowns. The system is *over-determined* and we can only find the *the best solution*, for example the least squares solution. Such a problem is a curve-fitting problems. When $n = 1$, it is also called linear regression.

- $m = n$, we have the same number of equations and unknowns. There is a unique solution to the linear system of equations. Such a problem is called an interpolation because $y_a(t)$ will pass all the selected data.

- $m < n$, we have fewer equations than the unknowns. The system is *under-determined* and we can find infinite number of the solutions. Often we prefer *the SVD solution* which has the least length among all the solutions.

Note that the coefficient matrix is dense (not many zero entries) in this application.

## 1.2 A non-linear model

Not all the functions are polynomials, if we want to approximate $y(t)$ by

$$y_a(t) \approx \alpha e^{\beta t} \tag{1.2.1}$$

[1]Not always possible.

for example, where $\alpha$ and $\beta$ are unknowns, we would have

$$\begin{cases} \alpha e^{\beta t_0} = y_0 \\ \alpha e^{\beta t_1} = y_1 \\ \cdots\cdots \\ \alpha e^{\beta t_m} = y_m \end{cases}$$

We get a non-linear system of equations for $\alpha$ and $\beta$.

## 1.3 Finite difference method and linear system of equations

The modern computers are designed in part to solve practical problems such as weather forecast, computing the lift and drag of airplanes, missiles, space shuttle etc. In these application, often partial differential equations, particularly the Navier-Stokes equations are used to model the physical phenomena. How to solve three dimensional Navier-Stokes equations is a still challenge today.

To solve the Navier-Stokes equations, often finite difference or finite element methods are used. In a finite difference method, the partial derivatives are replaced by finite difference, which is a combination of the function values. Such a process is called finite difference discretization. After the discretization, the system of partial differential equations because a system of algebraic system of equations either linear or non-linear. We use a one-dimensional example here to illustrate the idea. Consider the two-point boundary value problem

$$\frac{d^2u(x)}{dx^2} = f(x), \quad 0 < x < 1 \tag{1.3.1}$$

$$u(0) = u(1) = 0. \tag{1.3.2}$$

In a finite difference method, we try to an approximate solution of $u(x)$ at a finite number of points (not everywhere). The procedure is as follows:

- Generate a grid. For example, we can select $n$ equally spaced points between 0 and 1 to find the approximate solution of $u(x)$. The the spacing between two points is $h = 1/n$, and these points are $x_i = ih$, $i = 0, 1, \cdots, n$ with $x_0 = 0$ and $x_n = 1$. We look for an approximate solution of $u(x)$ at $x_1$, $x_1$, $\cdots$, $x_{n-1}$. Note that we know $u(0) = u(x_0) = 0$ and $u(1) = u(x_n) = 0$ already from the boundary condition.

- Replace the derivative by a finite difference formula. It can be proved that

$$\lim_{h\to 0} \frac{u(x-h) - 2u(x) + u(x+h)}{h^2} = \frac{d^2u(x)}{dx^2}. \tag{1.3.3}$$

  Or

$$\frac{u(x-h) - 2u(x) + u(x+h)}{h^2} = \frac{d^2u(x)}{dx^2} + \frac{h^2}{12}\frac{d^4u(x)}{dx^4} \tag{1.3.4}$$

At every grid points $x_i$, $i = 0, 1, \cdots, n$, we use the above formula ignoring the high order terms to get

$$\frac{u(x_1 - h) - 2u(x_1) + u(x_1 + h)}{h^2} \approx \frac{d^2u(x_1)}{dx_1^2} = f(x_1)$$
$$\frac{u(x_2 - h) - 2u(x_2) + u(x_2 + h)}{h^2} \approx \frac{d^2u(x_2)}{dx_2^2} = f(x_2)$$
$$\cdots\cdots$$

$$\frac{u(x_i - h) - 2u(x_i) + u(x_i + h)}{h^2} \approx \frac{d^2u(x_i)}{dx_i^2} = f(x_i)$$
$$\cdots\cdots$$
$$\frac{u(x_{n-1} - h) - 2u(x_{n-1}) + u(x_{n-1} + h)}{h^2} \approx \frac{d^2u(x_{n-1})}{dx_{n-1}^2} = f(x_{n-1})$$

If we replace the '$\approx$' with the '$=$' sign, and replace $u(x_i)$ which we do not know with $U_i$ which is the solution to the linear system of equations:

$$\frac{0 - 2U_1 + U_2}{h^2} = f(x_1)$$
$$\frac{U_1 - 2U_2 + U_3)}{h^2} = f(x_2)$$
$$\cdots\cdots$$

$$\frac{U_{i-1} - 2U_i + U_{i+1}}{h^2} = f(x_i)$$
$$\cdots\cdots$$
$$\frac{U_{n-2} - 2U_{n-1} + 0}{h^2} = f(x_{n-1})$$

This system of equations $Ax = b$ can be written as the matrix and vector form:

$$\begin{bmatrix} -\frac{2}{h^2} & \frac{1}{h^2} & & & & \\ \frac{1}{h^2} & -\frac{2}{h^2} & \frac{1}{h^2} & & & \\ & \frac{1}{h^2} & -\frac{2}{h^2} & \frac{1}{h^2} & & \\ & & \ddots & \ddots & \ddots & \\ & & & \frac{1}{h^2} & -\frac{2}{h^2} & \frac{1}{h^2} \\ & & & & \frac{1}{h^2} & -\frac{2}{h^2} \end{bmatrix} \begin{bmatrix} U_1 \\ U_2 \\ U_3 \\ \vdots \\ U_{n-2} \\ U_{n-1} \end{bmatrix} = \begin{bmatrix} f(x_1) - \frac{u_a}{h^2} \\ f(x_2) \\ f(x_3) \\ \vdots \\ f(x_{n-2}) \\ f(x_{n-1}) - \frac{u_b}{h^2} \end{bmatrix} \tag{1.3.5}$$

with $u_0 = 0$ and $u_b = 0$[2].

- Solve the system of equations to get the approximate solution at each grid point.
- Implement and debug the computer code. Run the program to get the output. Analyze the results (tables, plots etc.).
- Error analysis: method error using finite difference to approximate derivative; machine error (round-off errors).

We will discuss how to use computer to solve the system of equations. Note that the coefficient matrix has special structure: tri-diagonal and most entries are zero. In two space dimensions, a useful partial differential equation is the Poisson equation

$$u_{xx} + u_{yy} = f \tag{1.3.6}$$

For a rectangular grid and uniform mesh, the finite differential equation at a grid point $(x_i, y_j)$ is

$$\frac{U_{i-1,j} + U_{i+1,j}}{(h_x)^2} + \frac{U_{i,j-1} + U_{i,j+1}}{(h_y)^2} - \left(\frac{2}{(h_x)^2} + \frac{2}{(h_y)^2}\right) U_{ij} = f_{ij} \tag{1.3.7}$$

For a general $n$ by $n$ grid, we will have

$$A = \frac{1}{h^2}\begin{bmatrix} B & I & & \\ I & B & I & \\ & \ddots & \ddots & \ddots \\ & & I & B \end{bmatrix}, \qquad B = \begin{bmatrix} -4 & 1 & & \\ 1 & -4 & 1 & \\ & \ddots & \ddots & \ddots \\ & & 1 & -4 \end{bmatrix}.$$

Note that the size of $A$ is $(n-1)^2 \times (n-1)^2$. If we take $n = 101$, we will have one million unknowns. A very large number that most laptop today may not be able to handle if we store all the entries. For such a system of equations, an iterative method or sparse matrix techniques are preferred. We will get back to this later.

## 1.4 Continuous and discrete eigenvalue problems

Consider the differential equation

$$u'' + \lambda u = 0, \quad 0 < x < 1, \quad u(0) = 0, \quad u(1) = 0. \tag{1.4.1}$$

The solution is not unique. One particular solution is

$$u(x) = \sin(k\pi x) \tag{1.4.2}$$

[2]For non-homogeneous boundary condition, we can plug in the value $u(0)$ and $u(1)$ in the place of $u_0 = 0$ and $u_b$.

corresponding to the eigenvalue $\lambda_k = (k\pi)^2$ for $k = 1, 2, \cdots$. The significance of the eigenfunctions and eigenvalues is the basis of the Fourier series and theory. For example, for non-homogeneous differential equation,

$$u'' = f(x), \quad 0 < x < 1, \quad u(0) = 0, \quad u(1) = 0. \tag{1.4.3}$$

The solution can be expressed as

$$u(x) = \sum_{k=1}^{\infty} \alpha_k \sin(k\pi), \tag{1.4.4}$$

with

$$\alpha_k = -\frac{1}{(k\pi)^2} \frac{\int_0^1 f(x) \sin(k\pi x) dx}{\int_0^1 \sin^2(k\pi x)}. \tag{1.4.5}$$

See an ordinary differential equation text book for the details. More general eigenvalue of this type would be

$$-(pu')' + qu + \lambda u = 0, \quad p(x) \geq p_0 > 0, q(x) \geq 0 \tag{1.4.6}$$

with a more general boundary condition.

The associate numerical method is the spectral method. For example, if we apply the finite difference method as discussed earlier, we would have

$$\frac{U_{i-1} - 2U_i + U_{i+1}}{h^2} + \lambda U_i = 0, \quad , i = 1, 2, c \ldots n - 1. \tag{1.4.7}$$

In the matrix vector form, it is $Ax = (-\lambda)\, x$, which is an algebraic eigenvalue problem. The matrix $A$ is given in (3.0.1).

## 1.5 Objectives of the class

- Learn how to solve these problems (mainly linear algebra) efficiently and reliably on computers.
- Method and software selection: what to use for what problems.
- Related theory and analysis.
- Preparation for other courses such as MA780, MA584, MA587 etc.

## 1.6 A computer number system

We want to use computers to solve mathematical problems and we should know the number system in a computer.

A primitive computer system is only part of a real number system. A number in a computer system is represented by

$$\begin{array}{lcll} & & \text{fraction} & \text{exponential} & \\ x_c = & \pm. & d_1 d_2 \cdots d_n & \beta^s, & 0 \le d_i \le \beta - 1, \quad -S_{max} \le s \le S_{max} \\ & \text{sign} & \text{mantissa} & \text{base} & \end{array} \tag{1.6.1}$$

Below are some examples of such numbers

$$-0.11112^5 \quad (\text{binary}, \beta = 2), \quad -0.314210^0, \quad -0.031410^0 \tag{1.6.2}$$

The choice of base number is

| | | |
|---|---|---|
| $\beta = 2$ | binary | primitive |
| $\beta = 8$ | octal | used for transition |
| $\beta = 10$ | decimal | custom & convenient |
| $\beta = 16$ | hexadecimal | save storage |

The float number is denoted by $fl(x)$. We can see that the expression of a floating number is not unique. To get a unique expression, it is often designed that $d_1 \neq 0$ if $x_c$ is a non-zero number. Such a floating number is called a *normalized* floating number. The number zero is expressed as $0.00 \cdots 0\beta^0$. Note that one bite is used to represent the sign in the exponential.

Often there are two number systems in a programming language for a particular computer, single precision corresponding to 32 bits; and double precision for 64 bits.

In a 32 bites computer number system, we have

| sign | exponential | fraction |
|---|---|---|
| 1 | 8 | 23 |

In a 64 bites computer number system, we have

| sign | exponential | fraction |
|---|---|---|
| 1 | 11 | 52 |

### 1.6.1 Properties of a computer number system

- It is a subset of the real number system with finite number of floating numbers. For a 32-bit system, the total numbers is roughly $2\beta^n(S_{max} - S_{min} + 1) - 1$.
- Even if $x$ and $y$ are in the computer number system, their operations, for example, $fl(xy)$, can be out of the the computer number system.

- It has the maximum and minimum numbers; and maximum and non-zero minimum magnitude. For a 32-bit system, the largest and smallest numbers can be calculated from the following:

  the largest exponential: $= 2^0 + 2^1 + \cdots + 2^6 = 2^7 - 1 = 127$

  the largest fraction $= 0.1111\cdots 1 = 1 - 2^{-23}$

  the largest positive number $= 2^{127}(1 - 2^{-23}) = 1.7014116 \times 10^{38}$.

  The smallest number is then $-1.7014116 \times 10^{38}$. The smallest positive number (or smallest magnitude) is

  $$0.000\cdots 1 \times 2^{-127} = 2^{-127-23} = 7.0064923 \times 10^{-46} \tag{1.6.3}$$

  while the smallest normalized magnitude is

  $$0.100\cdots 0 \times 2^{-127} = 2^{-128} = 2.9387359 \times 10^{-39}. \tag{1.6.4}$$

  **Overflow and underflow**

  If a computer system encounter a number whose magnitude is larger than the largest floating number of the computer system, it is called *OVERFLOW*. This often happens when a number is divided by zero, for example, we want to compute $s/a$ but $a$ is undefined, or evaluate a function outside of the definition, for example, $\log(-5)$. Computers often returns symbol such as $NAN$, $inf$, or simply stops the running process. This can also happen when a number is divided by a very small number. Often an overflow indicates a bug in the coding and should be avoided.

  If a computer system encounter a number whose magnitude is smaller than the smallest positive floating number of the computer system, it is called *underflow*. Often, the computer system can set this number as zero and there is no harm to the running process.

- The numbers in a computer number system is not evenly spaces. It is more clustered around the origin and get sparser far away.

While a computer system is only a subset of the real number system, often it is good enough if we know how to use it. If a single precision system is not adequate, we can use the double precision system.

## 1.7 Round-off errors and floating point arithmetics

Since computer number system is only a subset of a real number system, errors (called round-off errors) are inevitable when we solver problems using computers. The question that we need to ask is how the errors affect the final results and how to minimize the negative impact of errors.

### Input errors

When we input a number into a computer, it is likely to have some errors. For example, the number $\pi$ can be represented exact in a computer number system. Thus a floating number of expression of $\pi$ denoted as $fl(\pi)$ is different from $\pi$. The first question is, how a computer system approximate $\pi$. The default is the round-off approach. Let us take the cecimal system as an example. Let $x$ be a real number that is in the range of the computer number system in terms of the magnitude, and we express it as a normalized floating number

$$x = 0.d_1d_2\cdots d_nd_{n+1}\cdots \times 10^b, \quad d_1 \neq 0. \tag{1.7.1}$$

The floating number if the computer system using the round-off approach is

$$fl(x) = \begin{cases} 0.d_1d_2\cdots d_n \times 10^b, & \text{if } d_{n+1} \leq 4, \\ 0.d_1d_2\cdots (d_n+1) \times 10^b, & \text{if } d_{n+1} \geq 5 \end{cases} \tag{1.7.2}$$

### 1.7.1 Definition of errors

**The absolue error** is defined as the difference between the true value and the approximated,

$$\text{absolute error} = \text{true - approximated.} \tag{1.7.3}$$

Thus the error for $fl(x)$ is

$$\text{absolution error of } fl(x) = x - fl(x). \tag{1.7.4}$$

The absolution error is often simply called the error.

Absolution error may not reflect the reality. One picks up 995 correct answers from 1000 problems certainly is better than the one that picks up 95 correct answers from 100 problems although both of the errors are 5. A more realistic error measurement is the relative error which is defined as

$$\text{relative error} = \frac{\text{absolution error}}{\text{true value}}, \tag{1.7.5}$$

for example, if $x \neq 0$, then

$$\text{relative error of } fl(x) = \frac{x - fl(x)}{x}. \tag{1.7.6}$$

Obviously, for different $x$, the error $x - fl(x)$ and the relative error are different. How do we then characterize the round-off errors? We seek the upper bounds, or the worst case, which should apply for all $x$'s.

For the round-off approach, we have

$$\begin{aligned} |x - fl(x)| &= \begin{cases} 0.00\cdots 0 d_{n+1}\cdots \times 10^b, & \text{if } d_{n+1} \le 4, \\ 0.00\cdots 0(1-d_{n+1})\cdots \times 10^b, & \text{if } d_{n+1} \ge 5 \end{cases} \\ &\le 0.00\cdots 05 \times 10^b = \frac{1}{2}10^{b-n}, \end{aligned}$$

which only depends on the magnitude of $x$. The relative error is

$$\frac{|x - fl(x)|}{|x|} \le \frac{\frac{1}{2}10^{b-n}}{|x|} \le \frac{\frac{1}{2}10^{b-n}}{0.1 \times 10^b} = \frac{1}{2}10^{-n+1} \overset{define}{==} \epsilon = \text{machine precision} \tag{1.7.7}$$

Note that the upper bound of the relative error for the round-off approach is independent of $x$, only depends on the computer number system. This upper bound is called the machine precision, or machine epsilon, which indicates the best accuracy that we can expect using the computer number system.

In general we have

$$\frac{|x - fl(x)|}{|x|} \le \frac{1}{2}\beta^{-n+1} \tag{1.7.8}$$

for any base $\beta$. For a single precision computer number system (32 bits) we have[3]

$$\epsilon = \frac{1}{2}2^{-23+1} = 2^{-23} = 1.192093 \times 10^{-7}. \tag{1.7.9}$$

For a 64-bits number system (double precision), we have

$$\epsilon = \frac{1}{2}2^{-52+1} = 2^{-52} = 2.220446 \times 10^{-16}. \tag{1.7.10}$$

Relative error is closely associate with the concept of the significant digits. In general, if a relative error is of order $10^{-5}$, for example, it is likely the result has 5 significant digits.

An approximate number can be regarded as a perturbation of the true vales according to the following theorem.

**Theorem 1.7.1** *If $x \in R$, then $fl(x) = x(1+\delta)$, $|\delta| \le \epsilon$ ia the relative error.*

There are other ways to input a number into a computer system. Two other approaches are *rounding*, and *chopping*, in which we have

$$fl(x) = 0.d_1 d_2 \cdots d_n \times 10^b \tag{1.7.11}$$

[3]Accoding to the definition, the $\epsilon \approx 1.2 \times 10^{-7}$ which is the same as Kahaner, Moler, and Nash's book, but it twice as larger as the result given in Demmel's book, which we think it is wrong.

for chopping and

$$fl(x) = 0.d_1 d_2 \cdots (d_n + 1) \times 10^b. \tag{1.7.12}$$

The errors bounds are twice as much as the round-off approach.

### 1.7.2 Error analysis of computer arithmetics

The primitive computer arithmetic only include addition, subtraction, multiplication, division, and logical operations. Logic operations do not generate errors. But the basic arithmetic operations will introduce errors. The error bounds are given in the following theorem.

**Theorem 1.7.2** *If $a$ and $b$ are two floating numbers in a computer number system, $fl(a \circ b)$ is in the range of the computer number system, then*

$$fl(a \circ b) = (a \circ b)(1 + \delta), \quad \circ : \quad + \quad - \quad \times \quad \div \tag{1.7.13}$$

*where*

$$|\delta| = |\delta(a, b)| \leq \epsilon, \tag{1.7.14}$$

*we also have*

$$fl(\sqrt{a}) = \sqrt{a}\,(1 + \delta). \tag{1.7.15}$$

Note that $\delta$ is the relative error if $(a \circ b) \neq 0$ of the operations and is bounded by the machine precision. This is because

$$fl(a \circ b) - (a \circ b) = \delta(a \circ b) \quad \text{absolution error}$$
$$\frac{fl(a \circ b) - (a \circ b)}{(a \circ b)} = \delta, \quad |\delta| \leq \epsilon.$$

We conclude that the arithmetic operations within a computer number system give the 'best' results that we can possible get. Does it mean that we do not need to worry about round-off errors at all? Of course not!

### 1.7.3 Round-off error analysis and how to avoid round-off errors

Now we assume that $x$ and $y$ are two real numbers. When we input them into a computer, we will have errors. First we consider the multiplications and divisions

$$\begin{aligned} fl(x \circ y)) &= fl\,(fl(x) \circ fl(y)) = fl\,(x(1 + \epsilon_x) \circ y(1 + \epsilon_y)), \qquad |\epsilon_x| \leq \epsilon, \quad |\epsilon_y| \leq \epsilon, \\ &= (x(1 + \epsilon_x) \circ y(1 + \epsilon_y))(1 + \epsilon_{x \circ y}), \quad |\epsilon_{x \circ y}| \leq \epsilon. \end{aligned}$$

Note that $\epsilon_x$, $\epsilon_y$, and $\epsilon_{x \circ y}$ are *different numbers* although they have the same upper bound! We distinguish several different cases

- Multiplication/division ($\circ = \times$ or $\div$), take the multiplication as an example, we have

$$\begin{aligned} fl(xy)) &= x(1+\epsilon_x)y((1+\epsilon_y)(1+\epsilon_{xy}) = xy\left(1+\epsilon_x+\epsilon_y+\epsilon_{xy}+O(\epsilon^2)\right) \\ &= xy(1+\delta), \qquad |\delta| \le 3\epsilon. \end{aligned}$$

  Often we ignore the high order term (h.o.t) since they one much smaller (for single precision, we have $10^{-7}$ versus $10^{-14}$). Thus delta is the *relative error* as we mentioned before. The error bound is understandable with the fact of 3 with two input errors and one from the multiplication. The absolute error is $-xy\delta$ which is bounded by $3|xy|\epsilon$. The same bounds hold for the division too if the divisor is not zero. Thus the errors from the multiplications/divisions are not big concerns here. But we should avoid dividing by small numbers if possible.

- Now we consideration a subtraction $\circ$ ( an addition can be treated as a subtraction since $a+b = a-(-b)$ or vise versa.). Now we have

$$\begin{aligned} fl(x-y)) &= (x(1+\epsilon_x) - y((1+\epsilon_y))\,(1+\epsilon_{xy}) \\ &= x-y+x\epsilon_x - y\epsilon_y + (x-y)\epsilon_{xy} + O(\epsilon^2). \end{aligned}$$

  The absolution error is

$$\begin{aligned} (x-y) - fl(x-y) &= -x\epsilon_x + y\epsilon_y - (x-y)\epsilon_{xy} \\ |(x-y) - fl(x-y)| &= |x|\epsilon + |y|\epsilon_y + |x-y|\epsilon \end{aligned}$$

  which does not seem to be too bad. But the relative error may be unbounded because

$$\begin{aligned} \frac{|(x-y) - fl(x-y)|}{|x-y|} &= \left|\frac{x\epsilon_x - y\epsilon_y - (x-y)\epsilon_{xy}}{x-y}\right| + O(\epsilon^2) \\ &\le \frac{|x\epsilon_x - y\epsilon_y|}{|x-y|} + \epsilon. \end{aligned}$$

  In general, $\epsilon_x \neq \epsilon_x$ even though they are very small and have the same upper bound. Thus the relative error can be arbitrarily large if $x$ and $y$ are very close! That means the addition/subtraction can lead the ***loss of accuracy***, or significant digits. It is also called ***catastrophic cancellation*** as illustrate in the following example

$$0.31343639 - 0.31343637 = 0.00000002.$$

  If the last two digits of the two numbers are wrong (likely in many circumstance), then there is no significant digit left in the result. In this example, the absolute error is till small, but the relative error is very large!

**Round-off error analysis summary**

- Use formulas $fl(x) = x(1+\epsilon_1)$, $fl(x \circ y) = (x \circ y)(1+\epsilon_2)$ etc.
- Expand and collect terms.
- Ignore high order terms.

### 1.7.4 An example of accuracy loss

Assume we want to solve a quadratic equation $ax^2 + bx + c = 0$ on a computer, how do we do it? First of all, we need to write down the algorithm mathematically before coding it. There are at least three methods.

- Algorithm 1: $x_1 = \dfrac{-b + \sqrt{b^2 - 4ac}}{2a}, \quad x_2 = \dfrac{-b - \sqrt{b^2 - 4ac}}{2a},$
- Algorithm 2: $x_1 = \dfrac{-b + \sqrt{b^2 - 4ac}}{2a}, \quad x_2 = \dfrac{c}{x_1},$
- Algorithm 3: $x_1 = \dfrac{-b + -\sqrt{b^2 - 4ac}}{2a}, \quad x_2 = \dfrac{c}{x_1}.$

Mathematically, there are all equivalent (thus they are all call consistent). But occasionally, they may give very different results especially if $c$ is very small. When we put select the algorithm to run on computer, we should choose Algorithm 2 if $b \leq 0$ and Algorithm 3 if $b \geq 0$, why? This can be done using *if* $\cdots$ *then* conditional expression using any computer language.

Lets check with the simple case $a =$ 1a $b = 2$, $x^2 + 2x + e = 0$. When $e$ is very small, we have

$$x_1 = \frac{-2 - \sqrt{4 - 4e}}{2} = -1 - \sqrt{1 - e} \approx -2$$

$$x_2 = \frac{-2 + \sqrt{4 - 4e}}{2} = -1 + \sqrt{1 - e} = \frac{e}{-1 - \sqrt{1 - e}} \approx -0.5e$$

The last equality was obtained by rationalize to the denomenator.

Below is a Matlab code to illustrate four different algorithms:

```
function [y1,y2,y3,y4,y5,y6,y7,y8] = quad_err(e)

y1 = (-2 + sqrt(4 - 4*e))/2; y2 = (-2 -sqrt(4 - 4*e))/2;
y3 = (-2 + sqrt(4 - 4*e))/2; y4 = e/y3;
y5 = (-2 -sqrt(4 - 4*e))/2; y6 = e/y5;

y7 = -4*e/(-2 -sqrt(4 - 4*e))/2; y8 = e/y7;
```

From input various $e$, we can see how the accuracy gets lost. In general, when we have $e = 210^{-k}$, we will lose about $k$ significant digits.

### 1.7.5 How to avoid loss of accuracy?

- Use different formula, for example,

$$-b + \sqrt{b^2 - 4ac} = -\frac{4ac}{b + \sqrt{b^2 - 4ac}} \quad \text{if } b > 0.$$

- Use Taylor expansion, for examples,

$$1 - \cos x = 1 - \left(1 - \frac{x^2}{2} + \frac{x^4}{4!} - \cdots\right) \approx \frac{x^2}{2}.$$

$$f(x+h) - f(x) = hf'(x) + h^2 \frac{f''(x)}{2} + h^3 \frac{f'''(x)}{3!} + \cdots$$

- Another rule of thumb for summations $fl(\sum_{i=1}^{n})x_i$. We should add those numbers with small magnitude first to avoid "large numbers eat small numbers".

## 1.8 Some basic algorithms and Matlab codes

- Sum $\sum_{i=1}^{n} x_i$:

```
s=0;      % initialize
for i=1:n
  s = s + a(i);     % A common mistake is forget the s here!
end
```

- Product $\prod_{i=1}^{n} x_i$:

```
s=1;      % initialize
for i=1:n
  s = s * a(i);     % A common mistake is forget the s here!
end
```

**Example: Matrix-vector multiplication** $y = Ax$

In Matlab, we can simply use $y = A * x$. Or we can use the component form so that we can easily convert the code to other computer languages. We can put the following into a Matlab .m file, say, test_Ax.m with the following contents:

```
n=100; A=rand(n,n); x=rand(n,1);   % Generate a set of data
for i=1:n;
  y(i) = 0;      % initialize
  for j=1:n
     y(i) = y(i) + A(i,j)*x(j);    % Use ';' to compress the outputs.
  end
end
```

We wish to develop efficient algorithms (fast, less storage, accurate, and easy to program). Note that Matlab is case sensitive and the index of arrays should be positive integers (can not be zero).

### 1.8.1 Hornet's algorithm for computing a polynomial

Consider a polynomial $p_n(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$. How do we evaluate its value at a point $x$. We can store the coefficients first in an array, say, $a(1), a(2), \cdots a(n+1)$ since we can not use $a(0)$ in Matlab, then we can evaluate $p_(x)$ using

```
p = a(1);
for i=1:n
   p = p + a(i+1)*x^(i+1);
end
```

The total number of operations are about $O(n^2/2)$ multiplications, and $n$ additions. Howere, from the following observations

$$\begin{aligned} p_3(x) &= a_3x^3 + a_2x^2 + a_1x + a_0 \\ &= x(a_3x^2 + a_2x + a_1) + a_0 \\ &= x(x(a_3x + a_2) + a_1) + a_0 \end{aligned}$$

we can form the Hornet's algorithm (pseudo-code)

```
p = a(n)
for i=n-1,-1,0
   p = x*p + a(i)
endfor
```

which only requires $n$ multiplications and additions!

# Chapter 2

# Vector and matrix norms

Vector and matrix norms are generalization of the function of absolute value for single variable. There are at least two motivations to use them.

- Error estimates and analysis. How do we measure $fl(\mathbf{x})-\mathbf{x}$ if $\mathbf{x}$ is a vector or $fl(A)-A$ when $A$ is a matrix?
- Convergence analysis for iterative methods? How do we know a vector sequence $\mathbf{x}^k$ converges or not.

### 2.0.2 Definition of a vector norm

Give a $R^n$ space

$$R^n = \{\mathbf{x},\ \mathbf{x} = [x_1, x_2, \cdots, x_n]^T\}$$

A vector norm is defined as a *multi-variable function* $f(\mathbf{x})$ of its components $x_1$, $x_2$, $\cdots$, $x_n$ satisfying

1. $f(\mathbf{x}) \geq 0$ for $\forall x \in R^n$, and $f(\mathbf{x}) = 0$ if and only ($\Leftrightarrow$) if $\mathbf{x} = \mathbf{0}$.
2. $f(\alpha\mathbf{x}) = |\alpha| f(\mathbf{x})$.
3. $f(\mathbf{x}+\mathbf{y}) \leq f(\mathbf{x}) + f(\mathbf{y})$. It is called the triangle inequality.

If a function $f(\mathbf{x})$ satisfies (1)-(3), we use a special notation $f(\mathbf{x}) = \|x\|$ and call this function a norm in $R^n$.

#### An example of a vector norm

Let

$$f(\mathbf{x}) = \max_{1\leq i\leq n} \{|x_i|\}$$

. Is $f(\mathbf{x})$ a vector norm?

• It is obvious that $f(\mathbf{x}) \geq 0$. If $\mathbf{x} = 0$, then all $x_i = 0$, so $\max\{|x_i|\} = 0$, that is $f(\mathbf{x}) = 0$. On the other hand, if $f(\mathbf{x}) = \max\{|x_i|\} = 0$. That is, the largest magnitude is zero, which means all the components have to be zero. We conclude $\mathbf{x} = 0$.

•

$$f(\alpha\mathbf{x}) = \max_{1\leq i\leq n}\{|\alpha x_i|\} = \max_{1\leq i\leq n}\{|\alpha|\,|x_i|\} = |\alpha| \max_{1\leq i\leq n}\{|x_i|\} = |\alpha| f(\mathbf{x})$$

•

$$\begin{aligned} f(\mathbf{x}+\mathbf{y}) &= \max_{1\leq i\leq n}\{|x_i + y_i|\} = \max_{1\leq i\leq n}\{|x_i| + |y_i|\} \\ &\leq \max_{1\leq i\leq n}\{|x_i|\} + \max_{1\leq i\leq n}\{|y_i|\} = f(\mathbf{x}) + f(\mathbf{y}) \end{aligned}$$

Therefore $f($**x** is a vector and it is called the infinity norm, or maximum norm. It is denoted as $f(\mathbf{x} = \|\mathbf{x}\|_\infty$.

### Are the following functions vector norms?

- $f(\mathbf{x}) = \max_{1\leq i\leq n}\{|x_i|\} + 5$. No, since $f(\mathbf{0}) \neq 0$.
- $f(\mathbf{x}) = \dfrac{\max_{1\leq i\leq n}\{|x_i|\}}{x_1^2}$.No, since $f(\alpha\mathbf{x}) \neq |\alpha| f(\mathbf{x})$, and $f(\alpha\mathbf{x})$ has no definition for those vectors whose first component is zero.
- $f(\mathbf{x}) = \left\{\sum_{i=1}^{n} x_i^2\right\}^{1/2}$. Yes, it is called 2-norm, or Euclidian norm, it is denoted as $f(\mathbf{x}) = \|\mathbf{x}\|_2$. The sketch of the proof is given below.

### Sketch of the proof for 2-norm

Proof: (1) and (2) are obvious. The triangle inequality is

$$\left\{\sum_{i=1}^{n}(x_i^2 + y_i^2)\right\}^{1/2} \leq \left\{\sum_{i=1}^{n} x_i^2\right\}^{1/2} + \left\{\sum_{i=1}^{n} y_i^2\right\}^{1/2} \quad \text{or}$$

$$\sum_{i=1}^{n}(x_i^2 + y_i^2) \leq \sum_{i=1}^{n} x_i^2 + \sum_{i=1}^{n} y_i^2 + 2\left\{\sum_{i=1}^{n} x_i^2\right\}^{1/2}\left\{\sum_{i=1}^{n} y_i^2\right\}^{1/2} \quad \text{or}$$

$$2\sum_{i=1}^{n} x_i y_i \leq \left\{\sum_{i=1}^{n} x_i^2\right\}^{1/2}\left\{\sum_{i=1}^{n} y_i^2\right\}^{1/2}.$$

The last inequality is the Cauchy-Schwarz inequality. To prove this inequality, we consider a special quadratic function

$$\begin{aligned} g(\lambda) &= \sum_{i=1}^{n}(x_i^2 - \lambda y_i^2) \geq 0 \\ &= \sum_{i=1}^{n} x_i^2 - 2\lambda \sum_{i=1}^{n} x_i\, y_i + \lambda^2 \sum_{i=1}^{n} y_i^2 \\ &= c + b\lambda + a\lambda^2, \quad a = \sum_{i=1}^{n} y_i^2, \quad b = -2\sum_{i=1}^{n} x_i\, y_i, \quad c = \sum_{i=1}^{n} x_i^2. \end{aligned}$$

The function $g(\lambda)$ is a non-negative function, and the quadratic equation $g(\lambda) = 0$ has at most one real root or no roots. Therefore the discriminant should satisfy $b^2 - 4ac \leq 0$, that is

$$4\left(\sum_{i=1}^{n} x_i\, y_i\right)^2 - 4\sum_{i=1}^{n} y_i^2 \sum_{i=1}^{n} x_i^2 \leq 0.$$

This is equivalent to

$$\left|\sum_{i=1}^{n} x_i\, y_i\right| \leq \sqrt{\sum_{i=1}^{n} y_i^2 \sum_{i=1}^{n} x_i^2}.$$

This concludes

$$\sum_{i=1}^{n} x_i\, y_i \leq \left|\sum_{i=1}^{n} x_i\, y_i\right| \leq \left\{\sum_{i=1}^{n} x_i^2\right\}^{1/2} \left\{\sum_{i=1}^{n} y_i^2\right\}^{1/2}.$$

There are different Cauchy-Schwarz inequality in different space, for example, $L^2$ space, Sobolev space, Hilbert space etc. The proof process is similar. A special case is $y_i = 1$ for which we have

$$\left|\sum_{i=1}^{n} x_i\right| \leq \left\{\sum_{i=1}^{n} x_i^2\right\}^{1/2} \left\{\sum_{i=1}^{n} 1^2\right\}^{1/2} = \sqrt{n}\sqrt{\sum_{i=1}^{n} x_i^2}.$$

### 2.0.3 1-norm and $L^p$ norms

The function $f(\mathbf{x}) = \sum_{i=1}^{n} |x_i|$ is a vector. It is called 1-norm: $\|\mathbf{x}\| = \sum_{i=1}^{n} |x_i|$.

**An example:**

Let $\mathbf{x} = \begin{pmatrix} -5 \\ 1 \end{pmatrix}$, find $\|\mathbf{x}\|_p, p = 1, 2, \infty$.

The solution is $\|\mathbf{x}\|_1 = 6$, $\|\mathbf{x}\|_\infty = 5$, $\|\mathbf{x}\|_2 = \sqrt{26}$. Note that, we have $\|\mathbf{x}\|_\infty \leq \|\mathbf{x}\|_2 \leq \|\mathbf{x}\|_1$. This is true for any $\mathbf{x}$.

In general, we can define the $p$-norm for $p \geq 1$,

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

### 2.0.4 Some properties of vector norms

- There are infinity number of vector norms (not unique).
- All vector norms are equivalent in *finite dimensional space*. That is, give any two vector norms, say $\|\mathbf{x}\|_\alpha$ and $\|\mathbf{x}\|_\beta$, there are two constant $C_{\alpha\beta}$ and $c_{\alpha\beta}$ such that

  $$c_{\alpha\beta}\|\mathbf{x}\|_\alpha \leq \|\mathbf{x}\|_\beta \leq C_{\alpha\beta}\|\mathbf{x}\|_\alpha$$

  Note that, from the above inequality, we immediately have

  $$\frac{1}{C_{\alpha\beta}}\|\mathbf{x}\|_\beta \leq \|\mathbf{x}\|_\alpha \leq \frac{1}{c_{\alpha\beta}}\|\mathbf{x}\|_\beta.$$

Note that the constants are independent of $\mathbf{x}$ but may depend on the dimension ($n$).

Thus the one, two, and infinity norm of a vector norms are all equivalent. What are the smallest ($C$) and the largest $c$ constants?

**Theorem 2.0.1**

$$\frac{||\boldsymbol{x}||_1}{n} \leq ||\boldsymbol{x}||_\infty \leq ||\boldsymbol{x}||_2 \leq \sqrt{n}\,||\boldsymbol{x}||_\infty \leq \sqrt{n}\,||\boldsymbol{x}||_2 \leq \sqrt{n}\,||\boldsymbol{x}||_1.$$

As an illustration, we prove

$$||\mathbf{x}||_2 \leq ||\mathbf{x}||_1 \leq \sqrt{n}\,||\mathbf{x}||_2$$

**Proof:**

$$\begin{aligned} ||\mathbf{x}||_1^2 &= \left( \sum_{i=1}^{n} |x_i| \right)^2 = \sum_{i=1}^{n} |x_i|^2 + 2\sum_{i<j} |x_i||x_j| \\ &= ||\mathbf{x}||_2^2 + 2\sum_{i<j} |x_i||x_j| \geq |\mathbf{x}||_2^2. \end{aligned}$$

On the other hand, from the Cauchy-Schwarz inequality, we have already known that

$$||\mathbf{x}||_1 = \sum_{i=1}^{n} |x_i| \cdot 1 \leq \sqrt{n}\sqrt{\sum_{i=1}^{n} x_i^2} = \sqrt{n}||\mathbf{x}||_2.$$

### 2.0.5 Connection between the inner product and $||\mathbf{x}||_2$

The inner product of two vectors $\mathbf{x} \in R^n$, $\mathbf{y} \in R^n$ is defined as

$$(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{n} x_i y_i.$$

Particularly, if $\mathbf{y} = \mathbf{x}$, we have

$$(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{n} x_i x_i = \sum_{i=1}^{n} x_i^2 = ||\mathbf{x}||_2.$$

From Cauchy-Schwarz inequality, we also have

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

## 2.1 Matrix norms

There are two definitions of a matrix norms. The first one is to use the same definition as a vector norm.

**Definition:** A matrix norm is a multi-variable function of its entries that satisfy the following relations:

1. $f(\mathbf{A}) \geq 0$ for $\forall A \in R^{m \times n}$, and $f(\mathbf{A}) = 0$ if and only ($\Leftrightarrow$) if $\mathbf{A} = \mathbf{0}$.
2. $f(\alpha \mathbf{A}) = |\alpha| f(\mathbf{A})$.
3. $f(\mathbf{A} + \mathbf{B}) \leq f(\mathbf{A}) + f(\mathbf{B})$. It is called the triangle inequality.

If a function $f(\mathbf{A})$ satisfies (1)-(3), we use a special notation $f(\mathbf{x}) = \|A\|$ and call this function a norm in $R^{m \times n}$.

Or alternatively, we can treat a matrix as a long vector, then use the definition of the vector norm. For an example, if $A \in R^{m \times n} = \{a_{ij}\}$, if we treat the matrix as a long vector, either row-wise or column-wise, the 2-norm, now it is called ***Frobenius norm***, of the matrix is

$$\|A\|_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij}^2}. \tag{2.1.1}$$

For an $n$ by $n$ identity matrix, we have $\|I\| = \sqrt{n}$ instead of $\|I\| = 1$ we may have expected.

Since matrices are often used along with vectors, *e.g.* $Ax = b$, $Ax = \lambda x$, it is naturally to define a matrix norm from a vector norm.

### 2.1.1 Associated (induced, subordinate) matrix norms

**Theorem 2.1.1** *Given a vector norm $\|\cdot\|$ in $R^n$ space, the function of the matrix function in $A \in R^{m\times n}$ space defined below*

$$f(A) = \sup_{\boldsymbol{x}\neq 0} \frac{\|A\boldsymbol{x}\|}{\|\boldsymbol{x}\|} \qquad = \max_{\boldsymbol{x}\neq 0} \frac{\|A\boldsymbol{x}\|}{\|\boldsymbol{x}\|} = \max_{\|\boldsymbol{x}\|=1} \|A\boldsymbol{x}\| \tag{2.1.2}$$

is a matrix norm in $R^{m\times n}$ space. It is called the associated (or induced, or subordinate) matrix norm.

**Proof:**

- It is obvious that $f(A) \geq 0$. If $A = 0$, then $A\mathbf{x} = 0$ for any $\mathbf{x}$, so $\|A\mathbf{x}\| = 0$, thus $f(A) = 0$. On the other hand, if $f(A) = 0$, then we must have $A = 0$. Otherwise, there would be one entry of $A$ is non-zero, say $a_{ij} \neq 0$. We would conclude the vector $A\mathbf{e}_j$, which the $j$-th column of $A$, is a non-zero vector since one of component is $a_{ij}$. Therefore, $\|A\mathbf{e}_j\| \neq 0$, and $f(A) \geq \|A\mathbf{e}_j\|/\|\mathbf{e}_j\| \geq 0$ which contradicts the fact that $f(A) = 0$.
- $f(\alpha A) = \max_{\mathbf{x}\neq\mathbf{0}} \dfrac{\|\alpha A\mathbf{x}\|}{\|\mathbf{x}\|} = \max_{\mathbf{x}\neq\mathbf{0}} |\alpha| \dfrac{\|A\mathbf{x}\|}{\|\mathbf{x}\|} = |\alpha| \max_{\mathbf{x}\neq\mathbf{0}} \dfrac{\|A\mathbf{x}\|}{\|\mathbf{x}\|} = |\alpha| f(A)$.
- For any $\mathbf{x}$, from the property of a vector norm, we have $\|(A+B)\mathbf{x}\| = \|A\mathbf{x} + B\mathbf{x}\| \leq \|A\mathbf{x}\| + \|B\mathbf{x}\|$. Thus we have

$$\begin{aligned} \max_{\mathbf{x}\neq\mathbf{0}} \frac{\|(A+B)\mathbf{x}\|}{\|\mathbf{x}\|} &\leq \max_{\mathbf{x}\neq\mathbf{0}} \left( \frac{\|A\mathbf{x}\|}{\|\mathbf{x}\|} + \frac{\|B\mathbf{x}\|}{\|\mathbf{x}\|} \right) \\ &\leq \max_{\mathbf{x}\neq\mathbf{0}} \frac{\|A\mathbf{x}\|}{\|\mathbf{x}\|} + \max_{\mathbf{x}\neq\mathbf{0}} \frac{\|B\mathbf{x}\|}{\|\mathbf{x}\|} \\ &\leq f(A) + f(B) \end{aligned}$$

### 2.1.2 Properties of associated matrix norms

from the definition of associated matrix norms, we can conclude the following important properties.

- $\|I\| = 1$. This is obvious since $\max_{\mathbf{x}\neq\mathbf{0}} \dfrac{\|I\mathbf{x}\|}{\|\mathbf{x}\|} = 1$.
- $\|A\mathbf{x}\| \leq \|A\| \, \|\mathbf{x}\|$ for any $\mathbf{x}$. It is obviously true if $\mathbf{x} = \mathbf{0}$.

  **Proof:** If $\mathbf{x} \neq \mathbf{0}$, then we have

$$\|A\| = \max_{\mathbf{y}\neq\mathbf{0}} \frac{\|A\mathbf{y}\|}{\|\mathbf{y}\|} \geq \frac{\|A\mathbf{x}\|}{\|\mathbf{x}\|}.$$

  Multiplying $\|\mathbf{x}\|$ to both sides, we get $\|A\| \, \|\mathbf{x}\| \geq |A\mathbf{x}\|$.

- $\|AB\| \leq \|A\| \, \|B\|$ for any $A$ and $B$ that $AB$ exists.

  **Proof:** According to the definition, we have

$$\max_{\mathbf{x}\neq\mathbf{0}} \frac{\|AB\mathbf{x}\|}{\|\mathbf{x}\|} \leq \max_{\mathbf{x}\neq\mathbf{0}} \frac{\|A\|\|B\mathbf{x}\|}{\|\mathbf{x}\|} \leq \|A\| \max_{\mathbf{x}\neq\mathbf{0}} \frac{\|B\mathbf{x}\|}{\|\mathbf{x}\|} = \|A\| \, \|B\|.$$

### 2.1.3 Some commonly used matrix norms

For any vector norm, there is an associate matrix norm. Since we know how to evaluate $\|\mathbf{x}\|_p$, $p = 1, 2, \infty$, we should know how to evaluate $\|A\|_p$ as well. It is not practical to use the definition all the time.

**Theorem 2.1.2**

$$\|A\|_\infty = \max \left\{ \sum_{j=1}^n |a_{1j}|, \sum_{j=1}^n |a_{2j}|, \cdots, \sum_{j=1}^n |a_{ij}|, \cdots, \sum_{j=1}^n |a_{mj}| \right\} = \max_i \sum_{j=1}^n |a_{ij}| \quad (2.1.3)$$

$$\|A\|_1 = \max \left\{ \sum_{i=1}^n |a_{i1}|, \sum_{i=1}^n |a_{i2}|, \cdots, \sum_{i=1}^n |a_{ij}|, \cdots, \sum_{i=1}^n |a_{in}| \right\} = \max_j \sum_{i=1}^n |a_{ij}| \quad (2.1.4)$$

In other words, we add the magnitude of element in each rows, then we selected the largest one, which is the infinity norm of the matrix $A$.

**An example:** Let

$$A = \begin{pmatrix} -5 & 0 & 7 \\ 1 & 3 & -1 \\ 0 & 0 & 1 \end{pmatrix} \begin{matrix} 12 \\ 5 \\ 1 \end{matrix}$$

$$6 \quad 3 \quad 9$$

Then $\|A\|_\infty = 12$, $\|A\|_1 = 12$.

**Proof for $\|A\|_\infty\|$ norm.** Let

$$M = \max_i \sum_{j=1}^n |a_{ij}|.$$

The proof has two parts: (a), show that $\|A\|_\infty\| \leq M$; (b), find a specific $\mathbf{x}^*$, $\|\mathbf{x}^*\|_\infty = 1$, such that $\|Ax\|_\infty\| \geq M$. For any $\mathbf{x}$, $\|\mathbf{x}\|_\infty \leq 1$, we have

$$A\mathbf{x} = \begin{pmatrix} \sum_{j=1}^n a_{1j} \\ \sum_{j=1}^n a_{2j} \\ \vdots \\ \sum_{j=1}^n a_{ij} \\ \vdots \\ \sum_{j=1}^n a_{mj} \end{pmatrix}, \qquad |x_j| \leq 1, \qquad |\sum_{j=1}^n a_{ij}x_j| \leq \sum_{j=1}^n |a_{ij}||x_j| \leq \sum_{j=1}^n |a_{ij}| \leq M.$$

Now we conduct the second step. The largest sum of the magnitude has to be one (or more) particular rows, say $i^*$-th row. We choose

$$\mathbf{x}^* = \begin{pmatrix} \text{sign}(a_{i^*,1}) \\ \text{sign}(a_{i^*,2}) \\ \vdots \\ \text{sign}(a_{i^*,n}) \end{pmatrix}, \qquad \text{sign}(x) = \begin{cases} 1 & x > 0 \\ 0 & x = 0 \\ -1 & x < 0 \end{cases}, \qquad x\,\text{sign}(x) = |x|.$$

Thus we get

$$\|A\mathbf{x}^*\|_\infty = \| \begin{pmatrix} \times \\ \vdots \\ \sum_{j=1}^{n} a_{ij}\, x_j^* \\ \vdots \\ \times \end{pmatrix} \|_\infty = \sum_{j=1}^{n} |a_{ij}| = M,$$

where $\times$ means some number. That completes the proof.

**Theorem 2.1.3**

$$\begin{aligned} \|A\|_2 &= \max\left\{\sqrt{\lambda_1(A^HA)}, \sqrt{\lambda_2(A^HA)}, \cdots, \sqrt{\lambda_i(A^HA)}, \cdots \sqrt{\lambda_n(A^HA)}\right\} \\ &= \max_i \sqrt{\lambda_i(A^HA)}, \end{aligned}$$

*where $\lambda_i(A^HA)$, $i = 1, 2, \cdots, n$, are eigenvalues of $A^HA$, $A^H$ is the conjugate transpose of $A$.*

The proof is left as an exercise. Note that even if $A$ is a complex matrix, $A^HA$ or $AA^H$ are symmetric semi-positive definite matrix. A symmetric semi-positive definite matrix $B$ ($B = B^H = \{b_{ij}\}$) satisfies the following:

- For any $x$, $\mathbf{x}^HB\mathbf{x} \geq 0$. Note that if $B$ is a real matrix, then $\mathbf{x}^HB\mathbf{x} = \sum_{ij} b_{ij}x_ix_j$.
- All eigenvalues of $B$ are non-negative.

**An example:** Consider the following matrix:

$$A = \begin{bmatrix} 1 & -1 \\ 0 & 4 \end{bmatrix}.$$

We have $\|A\|_\infty = 5$, $\|A\|_1 = 4$. To get $\|A\|_2$. We have

$$A^TA = \begin{bmatrix} 1 & 0 \\ -1 & 4 \end{bmatrix} \begin{bmatrix} 1 & -1 \\ 0 & 4 \end{bmatrix} = \begin{bmatrix} 1 & -1 \\ -1 & 17 \end{bmatrix}.$$

Thus $det(\lambda I - {}^T A) = (\lambda - 1)(\lambda - 17) - 1 = 0$, or $\lambda^2 - 18\lambda + 16 = 0$ whose roots are $\lambda = (18 \pm \sqrt{18^2 - 64})/2$. We conclude that $\|A\|_2 = \sqrt{9 + \sqrt{65}} \approx 4.1306$. We can see that $\|A\|_2$ is more difficult to get. However, if $A$ is symmetric, that is $A = A^H$, then we can show that

$$\|A\|_2 = \max_i\{|\lambda_i(A)|\}, \qquad \text{if } A \text{ is symmetric.}$$

Furthermore, if $det(A) \neq 0$, that is $A$ is invertible, then

$$\|A^{-1}\|_2 = \frac{1}{\min\{|\lambda_i(A)|\}}, \qquad \text{if } A \text{ is symmetric and } det(A) \neq 0.$$

# Chapter 3

# Solving a linear system of equations–Direct method

In this chapter, we discuss some most used direct methods for solving a linear system of equations of the following form

$$A\mathbf{x} = \mathbf{b}, \qquad A \in R^{n\times n}, \qquad \mathbf{b} \in R^n, \qquad det(A) \neq 0. \tag{3.0.1}$$

The condition of $det(A) \neq 0$ has the following equivalent statements

- The linear system of equations $A\mathbf{x} = b$ has a unique solution.
- $A$ is invertible, that is, $A^{-1}$ exists.

Almost all the direct methods are based on Gaussian elimination. ***A direct method*** is a method that returns the exact solution in *finite number of operations* with exact computation (no round-off errors present). Such a method is often suitable for small to modest, dense matrices.

The main idea of *Gaussian elimination algorithm* is based on the following observation. Consider the following (upper) triangular system of equations

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & \cdots & a_{1n} \\ & a_{22} & \cdots & \cdots & a_{2n} \\ & & \ddots & \ddots & \vdots \\ & & & \ddots & \vdots \\ & & & & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ \vdots \\ b_n \end{pmatrix}. \tag{3.0.2}$$

From the structure of the system of equations, we can

- From the last equation: $a_{nn}x_n = b_n$, we get $x_n = b_n/a_{nn}$.

- From the last but second equation: $a_{n-1,n-1}x_{n-1} + a_{n-1,n}x_n = b_{n-1}$, we get $x_{n-1} = (b_{n-1} - a_{n-1,n}x_n)/a_{n-1,n-1}$. Note that, we need the result $x_n$ from previous step.

- In general (the idea of induction), assume we have computed $x_n$, $x_{n-1}$, $\cdots$, $x_{i+1}$, from the $i$-th equation, $a_{ii}x_i + a_{i,i+1}x_i + \cdots + a_{in}x_n = b_i$, we get

$$x_i = \left( b_i - \sum_{j=i+1}^{n} a_{ij}x_j \right) /a_{ii}, \quad i = n, n-1, \cdots, 1. \tag{3.0.3}$$

Equation is called the *backward substitution*. A psuedo-code is given below:

for $i = n, -1, 1$

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

endfor

Below is a matlab function of the backward substitution.

```
function [x] = backward(n,A,b)
for i=n:-1:1
  x(i) = b(i);
  for j=i+1:n
     x(i) = x(i) - a(i,j)*x(j);
  end
  x(i) = x(i)/a(i,j);
end
```

In one step, we can count the number of operations. In i-th step, there are $(n - i)$ multiplications and one division; and $(n - i)$ subtractions. Thus the total number of multiplications and divisions is

$$1 + 2 + \cdots + n = \frac{n(n+1)}{2} = \frac{n^2}{2} + O(n).$$

The total number of addtions/subtractions is

$$1 + 2 + \cdots + n - 1 = \frac{n(n-1)}{2} = \frac{n^2}{2} + O(n).$$

The total cost is only about one matrix-vector multiplications which is considered to be very fast.

### 3.0.4 Derivation of the Gaussian elimination algorithm

The main idea of Gaussian elimination (GE) is to use a row transforms to reduce they system to an upper triangular one while keep the solution unchanged. For this purpose, we can apply the Gaussian elimination to the coefficient matrix or to the augmented matrix which is defined as $\left[ A \vdots \mathbf{b} \right]$, that is, the matrix is enlarged by a column.

First, we use a 4 by 4 matrix to illustrate the idea. We use the number to indicate the number of the times that the entries have been changed, and the sequence of changes.

$$\left[\begin{array}{cccc:c} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{array}\right] \Longrightarrow \left[\begin{array}{cccc:c} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \end{array}\right] \Longrightarrow$$

$$\Longrightarrow \left[\begin{array}{cccc:c} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 2 & 2 & 2 \\ 0 & 0 & 2 & 2 & 2 \end{array}\right] \Longrightarrow \left[\begin{array}{cccc:c} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 2 & 2 & 2 \\ 0 & 0 & 0 & 3 & 3 \end{array}\right]$$

We can see that we need $(n-1)$ step to reach this goal.

In order to do these steps, we multiply a sequence of simple matricides to the augmented matrix (or original one):

$$L_{n-1} L_{n-2} \cdots L_2 L_1 \left[ A \vdots \mathbf{b} \right]. \tag{3.0.4}$$

We need to derive the recursive relations so that we can implement the algorithm. The general procedure is to derive the first step, maybe second step if necessary..., the general step to see if we have a complete algorithm. For convenience, we denote the right hand side

as $\mathbf{b}^T = [a_{1,n+1}, a_{2,n+1}, \cdots, a_{n,n+1}]$, we set $L_1 \left[ A \vdots \mathbf{b} \right]$

$$\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ -l_{21} & 1 & \ddots & \ddots & 0 \\ -l_{31} & 0 & 1 & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & 0 \\ -l_{n1} & 0 & \cdots & 0 & 1 \end{bmatrix} \left[ \begin{array}{ccccc:c} a_{11} & a_{12} & \cdots & \cdots & a_{1n} & a_{1,n+1} \\ a_{21} & a_{22} & \cdots & \cdots & a_{2n} & a_{2,n+1} \\ \vdots & \vdots & \cdots & \cdots & \cdots & \vdots \\ \vdots & \vdots & \cdots & \cdots & \cdots & \vdots \\ a_{n1} & a_{n2} & \cdots & \cdots & a_{nn} & a_{n,n+1} \end{array} \right]$$

$$= \left[ \begin{array}{ccccc:c} a_{11} & a_{12} & \cdots & \cdots & a_{1n} & a_{1,n+1} \\ 0 & a_{22}^{(2)} & \cdots & \cdots & a_{2n}^{(2)} & a_{2,n+1}^{(2)} \\ \vdots & & & & & \\ \vdots & & & a_{ij}^{(2)} & & \\ 0 & & & & & \end{array} \right]$$

We need to derive the formulas for $l_{i1}$ and $a_{ij}^{(2)}$. We multiply the 2-nd row of $L_1$ to the 1-st column of $[A|\mathbf{b}] = A^{(1)}$ to get

$$-l_{21}a_{11}^{(1)} + a_{21}^{(1)} = 0, \quad \Longrightarrow l_{21} = \frac{a_{21}^{(1)}}{a_{11}^{(1)}}.$$

We multiply the 3-rd row of $L_1$ to the 1-st column of $[A|\mathbf{b}] = A^{(1)}$ to get

$$-l_{31}a_{11}^{(1)} + a_{31}^{(1)} = 0, \quad \Longrightarrow l_{31} = \frac{a_{31}^{(1)}}{a_{11}^{(1)}}.$$

In general, we multiply the $i$-th row of $L_1$ to the 1-st column of $[A|\mathbf{b}] = A^{(1)}$ to get

$$-l_{i1}a_{11}^{(1)} + a_{i1}^{(1)} = 0, \quad \Longrightarrow l_{i1} = \frac{a_{i1}^{(1)}}{a_{11}^{(1)}}, \qquad i = 2, 3, \cdots, n.$$

So we have general formula for $L_1$.

Now we consider the general formulae for $a_{ij}^{(2)}$, $i = 2, 3, \cdots, n$, $j = 2, 3, \cdots, n, n+1$. We multiply the i-th row of $L_1$ to the j-st column of $[A|\mathbf{b}] = A^{(1)}$ to get

$$-l_{i1}a_{1j}^{(1)} + a_{ij}^{(1)} = a_{ij}^{(2)}, \qquad a_{ij}^{(2)} = a_{ij}^{(1)} - \frac{a_{i1}^{(1)}}{a_{11}^{(1)}} a_{1j}^{(1)}$$

$$i = 2, 3, \cdots, n, \qquad j = 2, 3, \cdots, n, n+1.$$

Let us do one more step: $L_2L_1A^{(1)} = L_2A^{(2)} = A^{(3)}$,

$$\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & \ddots & \ddots & 0 \\ 0 & -l_{32} & 1 & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & -l_{n2} & \cdots & 0 & 1 \end{bmatrix} \begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \cdots & \cdots & a_{1n} & \vdots & a_{1,n+1}^{(1)} \\ 0 & a_{22}^{(2)} & \cdots & \cdots & a_{2n}^{(2)} & \vdots & a_{2,n+1}^{(2)} \\ \vdots & \vdots & \cdots & \cdots & \cdots & \vdots & \vdots \\ \vdots & \vdots & \cdots & \cdots & \cdots & \vdots & \vdots \\ 0 & a_{n2}^{(2)} & \cdots & \cdots & a_{nn}^{(2)} & \vdots & a_{n,n+1}^{(2)} \end{bmatrix}$$

$$= \begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \cdots & \cdots & a_{1n}^{(1)} & \vdots & a_{1,n+1}^{(1)} \\ 0 & a_{22}^{(2)} & \cdots & \cdots & a_{2n}^{(2)} & \vdots & a_{2,n+1}^{(2)} \\ \vdots & & & & & \vdots & \\ \vdots & & & a_{ij}^{(3)} & & \vdots & \\ 0 & & & & & \vdots & \end{bmatrix}$$

Since the formulas only depend on the indexes, we just need to replace the formulas using the substitutions: $1 \Longrightarrow 2$ and $2 \Longrightarrow 3$ to get the following formulas:

$$l_{i2} = \frac{a_{i2}^{(2)}}{a_{22}^{(2)}}, \qquad i = 3, \cdots, n.$$

$$a_{ij}^{(3)} = a_{ij}^{(2)} - \frac{a_{i2}^{(2)}}{a_{22}^{(2)}} a_{2j}^{(2)}$$

$$i = 3, 4, \cdots, n, \qquad j = 3, 4, \cdots, n, n+1.$$

Since there are $(n-1)$ steps, in $k$-step, we should have

$$l_{i,k+1} = \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}}, \qquad i = k+1, \cdots, n.$$

$$a_{ij}^{(k+1)} = a_{ij}^{(k)} - \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}} a_{kj}^{(k)}$$

$$i = k+1, \cdots, n, \qquad j = k+1, \cdots, n, n+1.$$

Important observations of the Gaussian elimination process:

- After we get $a_{ij}^{(2)}$, we do not need $a_{ij}^{(1)}$ anymore. We can overwrite to save the storage.
- We do not need to store zeros below the diagonals. Often we store $l_{ij}$.

Below is the pseuo-code using the overwrite

$$
\begin{aligned}
&\text{for } k = 1, n-1 \\
&\quad \text{for } i = k+1, n \\
&\quad\quad a_{ik} := \frac{a_{ik}}{a_{kk}} \\
&\quad\quad \text{for } j = k+1, n \\
&\quad\quad\quad a_{ij} := a_{ij} - a_{ik}a_{kj} \\
&\quad\quad \text{end} \\
&\quad \text{end} \\
&\text{end}
\end{aligned}
$$

### 3.0.5 What can we get after the Gaussian elimination?

- An upper triangular system

$$
\begin{pmatrix}
a_{11} & a_{12} & \cdots & \cdots & a_{1n} & \vdots & a_{1,n+1} \\
 & a_{22} & \cdots & \cdots & a_{2n} & \vdots & a_{2,n+1} \\
 & & \ddots & \ddots & \vdots & \vdots & \vdots \\
 & & & \ddots & \vdots & \vdots & \vdots \\
 & & & & a_{nn} & \vdots & a_{n,n+1}
\end{pmatrix} \tag{3.0.5}
$$

  which can be solved using the backward substitution.

- A matrix factorization of $A = LU$, where $L$ is a unit lower triangular matrix and $U$ is an upper triangular matrix.

$$
A = \begin{bmatrix}
1 & 0 & 0 & \cdots & 0 \\
a_{21} & 1 & \ddots & \ddots & 0 \\
a_{31} & a_{32} & 1 & \ddots & 0 \\
\vdots & \ddots & \ddots & \ddots & 0 \\
a_{n1} & a_{n2} & \cdots & a_{n,n-1} & 1
\end{bmatrix}
\begin{pmatrix}
a_{11} & a_{12} & \cdots & \cdots & a_{1n} \\
 & a_{22} & \cdots & \cdots & a_{2n} \\
 & & \ddots & \ddots & \vdots \\
 & & & \ddots & \vdots \\
 & & & & a_{nn}
\end{pmatrix} = LU \tag{3.0.6}
$$

- The determinant of $A$ is

$$
det(A) = a_{11}a_{22}\cdots a_{nn} = det(U). \tag{3.0.7}
$$

  This is obvious since $det(L_i) = 1$ and the property of $det(AB) = det(A)det(B)$.

- The GE method breaks down if one of the diagonals is zero!

**Sketch of the proof of $LU$ decomposition from GE process.**

From

$$L_{n-1}L_{n-2}\cdots L_2L_1A = U,$$

we have

$$A = (L_{n-1}L_{n-2}\cdots L_2L_1)^{-1}\,U = L_1^{-1}L_2^{-1}\cdots L_{n-2}^{-1}L_{n-1}^{-1}U.$$

It is easy to check that

$$L_1^{-1} = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ l_{21} & 1 & \ddots & \ddots & 0 \\ l_{31} & 0 & 1 & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & 0 \\ l_{n1} & 0 & \cdots & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ \frac{a_{21}}{a_{11}} & 1 & \ddots & \ddots & 0 \\ \frac{a_{31}}{a_{11}} & 0 & 1 & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & 0 \\ \frac{a_{n1}}{a_{11}} & 0 & \cdots & 0 & 1 \end{bmatrix} \tag{3.0.8}$$

In other words, we just need to *change the sign* of the non-zero column below the diagonal. It is also easy to show that

$$L_1^{-1}L_2^{-1} = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ l_{21} & 1 & \ddots & \ddots & 0 \\ l_{31} & l_{32} & 1 & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & 0 \\ l_{n1} & l_{n2} & \cdots & 0 & 1 \end{bmatrix} \tag{3.0.9}$$

In other words, we can simply *add the non-zero columns together*. Note that this is not true for $L_2^{-1}L_2^{-1}$. Repeat the process leads to the desired $LU$ decomposition.

### 3.0.6 Operation counts for the Gaussian elimination algorithm

As part of analysis, we need to know the *cost* of an algorithm, or the total number of operations needed to complete the algorithm.

For the GE algorithm, we count the number of operations in the $k$-th step, then add them together. In $k$-th step of GE algorithm, we need to compute

$$\begin{aligned} &a_{i,k+1} := \frac{a_{ik}}{a_{kk}}, \qquad i = k+1, \cdots, n. \\ &a_{ij} =: a_{ij} - a_{ik}a_{kj} \\ &i = k+1, \cdots, n, \qquad j = k+1, \cdots, n, n+1. \end{aligned}$$

We need $(n-k)$ divisions for computing $a_{ik}/a_{kk}$ and $(n-k)(n-k+1)$ multiplications for updating the modified elements. Thus the total number of multiplications/divisions[1] is $(n-k)(n-k+2)$ for $k = 1, 2, \cdots n-1$. Thus we have the table for the number of multiplications/divisions

| | |
|---|---|
| $k=1$ | $(n-1)(n+1)$ |
| $k=2$ | $(n-2)n$ |
| $\vdots$ | $\vdots$ |
| $n-1$ | $1 \cdot 3$ |

The total is

$$\sum_{k=1}^{n-1} k(k+2) = \sum_{k=1}^{n-1} k^2 + \sum_{k=1}^{n-1} 2k = \frac{(n-1)n(2n-1)}{6} + n(n-1) = O(\frac{n^3}{3})$$

Note that the first term is actually the cost if we apply the GE process to the matrix $A$, the second part if the same transform applied to the right hand side which is equivalent to the *forward substitution*. We often emphasize the order of operations which is $O(n^3/3)$. The constant coefficient $(1/3)$ is also important here.

The number of additions/substractions is almost the same. When $n = 1000$, the total number of operations is roughly $2 \cdot 10^9/3$ (about a billion), which is a quite large number. It is true that for large and dense matrix, the GE algorithm is *not* very fast!

### 3.0.7 Partial pivoting strategy to avoid break-down and large errors

When $a_{11} = 0$, the GE algorithm breaks down even if $A$ is non-singular. For example, we consider $A\mathbf{x} = \mathbf{b}$ where

$$A = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \qquad det(A) = -1 \neq 0, \qquad A_1 = \begin{pmatrix} \epsilon & 1 \\ 1 & 0 \end{pmatrix}$$

The Gaussian elimination fails for the first $A$, for the second one, in general we will have

$$\begin{aligned} & fl\left(a_{ij} - \frac{a_{i1}}{a_{11}} a_{1j}\right) \\ & \quad = \left(a_{ij} - \frac{a_{i1}}{a_{11}} a_{1j}(1+\delta_1)(1+\delta_2)\right)(1+\delta_3) \\ & \quad = a_{ij} - \frac{a_{i1}}{a_{11}} a_{1j} - \frac{a_{i1}}{a_{11}} a_{1j}\delta_3 + \cdots \end{aligned}$$

We can see that if $|a_{11}|$ is very small, the round-off error will be amplified. The element $a_{11}$ is called the *pivot element.*

[1] Usually we put the multiplication and divisions into one category; and additions/subtractions into another category. An operation in the first category often take slightly longer time than that in the second category

**Partial column pivoting strategy:** Before the Gaussian elimination, we exchanges the row of the augmented matrix (or the original matrix) such that the switched *pivot element* has the largest magnitude among all of the elements in the column. Below are the steps of the 1-st Gaussian elimination with column partial pivoting algorithm,

- Choose $a_{l1}$ such that

$$|a_{l1}| \geq |a_{i1}|, \qquad i = 1, 2, \cdots, n.$$

  This can be done using

```
l = 1; pivot= abs(a(1,1));
for i=2:n
   if( abs(a(i,1)) ) > pivot
      pivot = abs(a(i,1));
      l = i;
   end
end
```

- Exchanges the $l$-th row with the 1-st row, $a_{lj} \longleftrightarrow a_{1j}$, $j = 1, 2, \cdots, n, n+1$. This can be done using

```
for j=1:n+1
    tmp = a(1,j);
    a(1,j) = a(l,j);
    a(1,j) = tmp
end
```

  You should be able to see how to change the variables.

- Carry out the Gaussian elimination as usual.

Below are some example:

$$\begin{bmatrix} 1 & 2 & 3 \\ -1 & 4 & 5 \\ 1 & -1 & 0 \end{bmatrix} \qquad \text{No pivoting is necessary}$$

$$\begin{bmatrix} 1 & 2 & 3 \\ -2 & 4 & 5 \\ 3 & -1 & 0 \end{bmatrix} \longmapsto \begin{bmatrix} 3 & -1 & 0 \\ -2 & 4 & 5 \\ 1 & 2 & 3 \end{bmatrix}$$

### 3.0.8 The matrix expression of partial column pivoting and elementary permutation matrix

The partial column pivoting algorithm can be expressed as row transform as well.

$$I = \begin{pmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & 1 & & & & \\ & & & \ddots & & & \\ & & & & 1 & & \\ & & & & & \ddots & \\ & & & & & & 1 \end{pmatrix} \longrightarrow \begin{pmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & 0 & & 1 & & \\ & & & \ddots & & & \\ & & 1 & & 0 & & \\ & & & & & \ddots & \\ & & & & & & 1 \end{pmatrix} = P_{ij}$$

An elementary permutation matrix $P_{ij}$ is the matrix obtained by exchanging the $i$-th and $j$-th rows of the identity matrix, for example

$$P_{13} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}.$$

If we multiply $P_{ij}$ to a Matrix $A$ from the left, the new matrix is the one obtained by exchanging the $i$-th and $j$-th rows of the matrix $A$, for example

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

Similarly, if we multiply $P_{ij}$ to a Matrix $A$ from the left, the new matrix is the one obtained by exchanging the $i$-th and $j$-th columns of the matrix $A$.

### 3.0.9 Properties of an elementary permutation matrix $P_{ij}$

- $P_{ij}P_{ij} = I$, and $P_{ij}^{-1} = P_{ij}$.
- $P_{ij}^T = P_{ij} = P_{ij}^{-1}$, that is, $P_{ij}$ is an orthogonal matrix, $P_{ij}^T P_{ij} = P_{ij}P_{ij}^T = I$.
- $det(P_{ij}) = 1$.

### 3.0.10 $PA = LU$ decomposition

The Gaussian elimination with partial column pivoting algorithm can be written as the following matrix form

$$L_{n-1}P_{n-2}L_{n-2}\cdots L_2P_2L_1P_1A = U, \tag{3.0.10}$$

Can we put all $L_i$'s and $P_i$'s together to get a $PA = LU$ decomposition by moving $P_i$'s to the right and $L_i$'s to the left? Yes, we can with some modification. We need to exchanges rows of $P_i$'s. We can write

$$\begin{aligned} &L_{n-1}P_{n-2}L_{n-2}\cdots L_2\widetilde{L}_1P_2P_1A = U,\\ &L_{n-1}P_{n-2}L_{n-2}\cdots \widetilde{L}_2\widetilde{\widetilde{L}}_1P_3P_2P_1A = U,\\ &\cdots \qquad \cdots \qquad \cdots \end{aligned}$$

Below is a demonstration how this can be done for $P_{24}L_1 = \widetilde{L}_1P_{24}$.

$$P_{24}L_1 = \begin{bmatrix} 1 & 0 & 0 & 0\\ 0 & 0 & 0 & 1\\ 0 & 0 & 1 & 0\\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0\\ -\frac{1}{3} & 1 & 0 & 0\\ \frac{2}{3} & 0 & 1 & 0\\ 1 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0\\ 1 & 0 & 0 & 1\\ \frac{2}{3} & 0 & 1 & 0\\ -\frac{1}{3} & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 & 0\\ 1 & 1 & 0 & 0\\ \frac{2}{3} & 0 & 1 & 0\\ -\frac{1}{3} & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0\\ 0 & 0 & 0 & 1\\ 0 & 0 & 1 & 0\\ 0 & 1 & 0 & 0 \end{bmatrix} = \widetilde{L}_1P_{24}.$$

We can see that when we move $P_{ij}$ to the right passing through $L_k$, we just need to exchanges the two rows in the non-zero column below the diagonal. Finally we will have $PA = LU$.

The determinant of $A$ can be computed using the following

$$det(PA) = det(L)det(U) = u_{11}u_{22}\cdots u_{nn}, \quad \text{or} \qquad det(A) = (-1)^m u_{11}u_{22}\cdots u_{nn},$$

where $m$ is the total number of row exchanges. Below is a pseudo-code of the process

for $k = 1, n-1,$ $\qquad$ $(n-1)$-th elimination
  $a_p = |a_{kk}|$
  $i_p = k$
  for $i = k+1, n$
    if $|a_{ik}| > |a_p|$ then
      $i_p = i$
      $a_p = |a_{ik}|$
    end
  end

  for $j = k, n+1$ $\qquad$ (or $n$)
    $a_t = a_{k,j}$

$a_{kj} = a_{ip,j}$

$a_{ip,j} = a_t$

end

begin Gaussian elimination process

.....

Finally, we have the $PA = LU$ decomposition. Below is an example how this process can be used. This example can be used in the debugging process.

### 3.0.11 Solving $A\mathbf{x} = \mathbf{b}$ using the $PA = LU$ decomposition

Once we have the $PA = LU$ decomposition, we can use the decomposition to solve the linear system of equation as follows:

1. Step 1: Form $\widetilde{\mathbf{b}} = P\mathbf{b}$, that is, exchange rows of $\mathbf{b}$. This is because from $A\mathbf{x} = \mathbf{b}$, we get $PA\mathbf{x} = P\mathbf{b}$.
2. Use forward substitution to solve $L\mathbf{y} = P\widetilde{\mathbf{b}}$. This is because we have $PA\mathbf{x} = LU\mathbf{x} = P\mathbf{b}$.
3. Use the backward substitution to solve $U\mathbf{x} = \mathbf{y}$ to get the solution.

### 3.0.12 An example of Gaussian elimination

See the scanned pdf file.

## 3.1 Other pivoting strategies

- *Complete pivoting.* At the first step, we choose $|a_{11}| := \max_{i,j} |a_{ij}|$. In this approach, we have to exchange both rows and columns which makes the programming more difficult. It may also destroy some matrix structures for certain matrices. The improvement in the accuracy is marginal compared with partial column pivoting.

- *Scaled column pivoting.* If the matrix $A$ has very different magnitude in rows, this approach is strongly recommended. At first step, we choose

$$\frac{|a_{k1}|}{s_k} = \max_{1\le i\le n} \frac{|a_{i1}|}{s_i} \tag{3.1.1}$$

where $s_i = \sum_{j=1}^{n} |a_{ij}|$.

## 3.2 Error Analysis

When we input vectors or matrices into a computer number system, we are going to have round-off errors, the error satisfy the following relations

$$fl(b_i) = b_i(1+\delta_i), \quad |\delta_i| \le \epsilon, \quad fl(\mathbf{b}) = \mathbf{b} + \mathbf{E}_b, \quad \|\mathbf{E}_b\|_p \le \|\mathbf{b}\|\,\epsilon,\ p = 1, 2, \infty,$$

$$fl(a_{ij}) = a_{ij}(1+e_{ij}), \quad |e_{ij}| \le \epsilon, \quad fl(A) = A + E_A, \quad \|E_A\|_p \le \|A\|\epsilon, p = 1, \infty.$$

In general, from the equivalence, we have

$$\|\mathbf{E}_b\| \le C_1\|\mathbf{b}\|\,\epsilon, \qquad \|E_A\|_p \le C_2\|A\|\epsilon, \tag{3.2.1}$$

for any vector and matrix norms, where $C_1$, $C_2$ are two constants depend on $n$.

Even before we solve the linear system of equations, we are solving a different problems $(A + E_A)x = \mathbf{b} + \mathbf{E}_b$ due to the input errors. The question is how the errors affect the results. This is summarized in the following theorem:

**Theorem 3.2.1** *If $\|A^{-1}E_a\| < 1$ (or $\|A^{-1}\|\|E_a\| < 1$, a stronger condition), define $\boldsymbol{x}_e = A^{-1}\boldsymbol{b}$, $\delta\boldsymbol{x} = (A + E_A)^{-1}(\boldsymbol{b} + \boldsymbol{E}_b) - A^{-1}\boldsymbol{b}$, then*

$$\frac{\|\delta\boldsymbol{x}\|}{\|\boldsymbol{x}_e\|} \le \frac{\|A\|\|A^{-1}\|}{1 - \|A\|\|A^{-1}\|\frac{\|E_A\|}{\|A\|}} \left(\frac{\|E_A\|}{\|A\|} + \frac{\|\boldsymbol{E}_b\|}{\|\boldsymbol{b}\|}\right), \tag{3.2.2}$$

*or the following if we ignore the high order terms:*

$$\frac{\|\delta\boldsymbol{x}\|}{\|\boldsymbol{x}_e\|} \le \|A\|\|A^{-1}\| \left(\frac{\|E_A\|}{\|A\|} + \frac{\|\boldsymbol{E}_b\|}{\|\boldsymbol{b}\|}\right) \tag{3.2.3}$$

We can see that $\|A\|\|A^{-1}\|$ is an important ***amplifying factor*** in the error estimate. it is called the ***condition number*** of the matrix $A$,

$$cond(A) = \|A\|\|A^{-1}\|. \tag{3.2.4}$$

For 2-norm, it is also denoted as $\kappa(A) = cond_2(A) = \|A\|_2\|A^{-1}\|_2$.

To prove the main theorem, we need the Banach's lemma.

**Lemma 3.2.1** *If $\|E\| < 1$, then $I + E$ is invertible, and*

$$\|(I + E)^{-1}\| \leq \frac{1}{1 - \|E\|}. \tag{3.2.5}$$

**Proof:** Consider the following matrix series:

$$B = I - E + E^2 - E^3 + \cdots + (-1)^k E^k + = \sum_{k=0}^{\infty} (-1)^k E^k$$

It partial sum

$$B_n = I - E + E^2 - E^3 + \cdots + (-1)^n E^n = \sum_{k=0}^{n} (-1)^k E^k$$

satisfies

$$\begin{aligned} \|B_n\| &\leq \|I\| + \| - E\| + \|E^2\| + \cdots + \|(-1)^n E^n\| \\ &\leq \|I\| + \|E\| + \|E\|^2 + \cdots + \|E\|^n = \frac{1 - \|E\|^n}{1 - \|E\|} \Longrightarrow \frac{1}{1 - \|E\|} \end{aligned}$$

Let $B = \lim_{n\to\infty} B_n$, then $B$ is the inverse of $I + E$ from the following

$$(I + E)B_n = (I + E)\left(I - E + E^2 - E^3 + \cdots (-1)^n E^n\right) = I - E^{n+1} \Longrightarrow I.$$

Furthermore,

$$\|(I + E)^{-1}\| \leq \|I\| + \|E\| + \|E\|^2 + \cdots + \|E\|^n + \cdots = \frac{1}{1 - \|E\|}.$$

Now we prove the main error theorem.

$$\begin{aligned} \frac{\|\delta \mathbf{x}\|}{\|\mathbf{x}_e\|} &= \|(A + E_A)^{-1}(\mathbf{b} + \mathbf{E}_b) - A^{-1}\mathbf{b}\| \\ &\leq \|(A + E_A)^{-1}(\mathbf{b} + \mathbf{E}_b - (A + E_A)A^{-1}\mathbf{b}\| \\ &\leq [A(I + A^{-1}E_A]^{-1}(\mathbf{b} + \mathbf{E}_b - \mathbf{b} - E_A\mathbf{x}_e\| \\ &= \|A^{-1}(I + A^{-1}E_A)^{-1}(\mathbf{E}_b - E_A\mathbf{x}_e)\| \\ &\leq \|A^{-1}\|\,\|(I + A^{-1}E_A]^{-1}\|(\|\mathbf{E}_b\| + \|E_A\|\,\|\mathbf{x}_e\|) \\ &\leq \frac{\|A^{-1}\|}{1 - \|A^{-1}E_A\|}\left(|\mathbf{E}_b\| + \|E_A\|\,\|\mathbf{x}_e\|\right). \end{aligned}$$

Notice that

$$\|A^{-1}E_A\| \leq \|A^{-1}\|\,\|E_A\| \leq \|A\|\,\|A^{-1}\|\,\frac{\|E_A\|}{\|A\|}.$$

Thus we continue to get

$$\frac{\|\delta\mathbf{x}\|}{\|\mathbf{x}_e\|} \leq \frac{\|A\|\,\|A^{-1}\|}{1-\|A\|\,\|A^{-1}\|\,\frac{\|E_A\|}{\|A\|}}\left(\frac{\|\mathbf{E}_b\|}{\|A\|\|\mathbf{x}_e\|}+\frac{\|E_A\|}{\|A\|}\right).$$

Since $\mathbf{b} = A\mathbf{x}_e$, we have $\|A\|\|\mathbf{x}_e\|$, we arrive at

$$\frac{\|\delta\mathbf{x}\|}{\|\mathbf{x}_e\|} \leq \frac{cond(A)}{1-cond(A)\,\frac{\|E_A\|}{\|A\|}}\left(\frac{\|\mathbf{E}_b\|}{\|\mathbf{b}\|}+\frac{\|E_A\|}{\|A\|}\right).$$

If we ignore high order terms in the above inequality, we can go one step further

$$\begin{aligned}\frac{\|\delta\mathbf{x}\|}{\|\mathbf{x}_e\|} &\leq cond(A)\left(\frac{\|\mathbf{E}_b\|}{\|\mathbf{b}\|}+\frac{\|E_A\|}{\|A\|}\right)\left(1+cond(A)\frac{\|E_A\|}{\|A\|}+O\left(cond(A)\frac{\|E_A\|}{\|A\|}\right)^2+\cdots\right)\\ &\approx cond(A)\left(\frac{\|\mathbf{E}_b\|}{\|\mathbf{b}\|}+\frac{\|E_A\|}{\|A\|}\right).\end{aligned}$$

This completes the proof.

**Remark 3.2.1**

- *The relative errors in the data (either (both) A or (and) b) are amplified by the factor of the condition number cond(A).*
- *Usually the upper bound is* over-estimated*.*
- *However, the upper bound is attainable.*
- *The condition number cond(A) has nothing to do with any algorithm. It only depends on the matrix itself. However, if cond(A) is very large in reference to the machine precision, then no matter what algorithm we use, in general, we can not expect to good result. Such a problem is called* **ill-conditioned matrix***, or simply* **ill-conditioned***. For an ill-conditioned system of linear equations, a small perturbation in the data (A and* ***b****) will cause large change in the solution.*
- *If cond(A) is small or modest in reference to the machine precision, the problem is then called well-conditioned.*

## 3.3 Wilkinson's Backward round-off error analysis

There are various errors during a problem solving process, for example, modelling errors, input error ($fl(A)$), algorithm error (truncation errors), and round-off errors. We have assumed to start with a mathematical problem and wish to use computer to solve it, therefore we will not discuss the modelling errors here.

Using Gaussian elimination method with partial pivoting, there is no formula error with partial pivoting (that is why it is called a direct method). So we only need to consider round-off errors. Round-off error analysis is often complicated. J. H. Wilkinson made it a little simple. His technique is simple, the round-off errors can be regarded as a perturbation to the original data, for example, $fl(x+y) = (x(1+\delta_x)+y(1+\delta_y))(1+\delta_3) = x(1+\delta_4)+y(1+\delta_5) = \widetilde{x}+\widetilde{y}$. In other words, the computed result is the *exact sum* of two perturbed numbers of the original data.

For Gaussian elimination method for solving $Ax = b$, we have the following theorem due to J. H. Wilkinson.

**Theorem 3.3.1** *The computed solution of the Gaussian elimination algorithm on a computer for solving $Ax = b$ is the exact solution of the following system of linear equations*

$$(A + E)\boldsymbol{x} = \boldsymbol{b}, \tag{3.3.1}$$

*where $\|E\| \leq Cg(n)\|A\|\epsilon$. The function $g(n)$ is called the* **growth factor** *that is defined below:*

$$g(n) = \frac{\max\limits_{k}\left\{\max\limits_{1\leq i,j\leq n}|a_{ij}^{(k)}|\right\}}{\max\limits_{1\leq i,j\leq n}\left\{|a_{ij}^{(1)}|\right\}} \tag{3.3.2}$$

The growth factor $g(n) \geq 1$. We also can prove the following

- For Gaussian elimination algorithm without pivoting, $g(n) = \infty$ indicating the method may break down.
- For Gaussian elimination algorithm with partial column pivoting, $g(n) = 2^{n-1}$. This bound is reachable.

**Proof:** For Gaussian elimination algorithm with partial column pivoting, we have

$$a_{ij}^{k+1} = a_{ij}^{k} - \frac{a_{ik}^{k}}{a_{kk}^{k}}a_{jk}^{k}.$$

Since $|\frac{a_{ik}^{k}}{a_{kk}^{k}}| \leq 1$, we conclude that

$$|a_{ij}^{k+1}| \leq \max_{ij}|a_{ij}^{k}| + \max_{ij}|a_{ij}^{k}| \leq 2\max_{ij}|a_{ij}^{k}| \leq 2\times 2\max_{ij}|a_{ij}^{k-1}|\cdots \leq 2^{k}\max_{ij}|a_{ij}^{1}|.$$

This concludes that $g(n) \leq 2^{n-1}$. The following example shows that such a bound is attainable,

$$\begin{bmatrix} 1 & & & & 1 \\ -1 & 1 & & & 1 \\ -1 & -1 & 1 & & 1 \\ \vdots & \ddots & \ddots & & 1 \\ -1 & -1 & \cdots & -1 & 1 \end{bmatrix} \dashrightarrow\dashrightarrow\dashrightarrow \begin{bmatrix} 1 & & & & 1 \\ 0 & 1 & & & 2 \\ 0 & 0 & 1 & & 2^2 \\ \vdots & \ddots & & & \vdots \\ 0 & 0 & \cdots & & 2^{n-1} \end{bmatrix}$$

However, the matrix above is a specific one, the general conjecture is that for most reasonable matrices, $g(n) \sim n$.

### 3.3.1 Factors that affected the accuracy of computed solutions

For most of computational problems, the relative error of the computer solution $\mathbf{x}_c$ approximating the true solution $\mathbf{x}_e$ satisfies the following relation

$$\frac{\|\mathbf{x}_c - \mathbf{x}_e\|}{\|\mathbf{x}_e\|} \leq cond(problem)\, g(algorithm)\, \epsilon. \tag{3.3.3}$$

That is, three factors affect the accuracy

- The computer used for solving the problem characterized by the machine precision.
- The condition number of the problem. For a linear system of equations $A\mathbf{x} = \mathbf{b}$, it is $cond(A)$.
- The algorithm used to solve the problem characterized by the growth factor $g$.

## 3.4 Residual vector and error estimates

In the error estimate, $\|A^{-1}\|$ is involved. But we know it is difficult and expensive to get $A^{-1}$. Do we have a better way to estimate how accurate an approximation is? The answer is the resisual vector.

**Definition 3.4.1** *Given a system of linear equations $A\boldsymbol{x} = \boldsymbol{b}$ and an approximation $\boldsymbol{x}_a$, the residual of $\boldsymbol{x}_a$ is defined as*

$$r(\boldsymbol{x}_a) = \boldsymbol{b} - A\boldsymbol{x}_a. \tag{3.4.1}$$

If $det(A) \neq 0$ and $r(\mathbf{x}_a) = \mathbf{0}$, the $\mathbf{x}_a$ is the true solution. We can use $\|r(\mathbf{x}_a)\|$ to measure how $\mathbf{x}_a$ is close to the true solution $\mathbf{x}_e = A^{-1}\mathbf{b}$. Note that $r(\mathbf{x}_a)$ is called computable since it just need matrix-vector multiplication and does need $A^{-1}$.

**Example:** Let

$$A = \begin{bmatrix} 1 & -1 & 0 \\ 2 & 0 & 1 \\ 3 & 0 & 2 \end{bmatrix}, \qquad \mathbf{b} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \qquad \mathbf{x}_a = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}.$$

Then the residual vector of $\mathbf{x}_a$ is

$$\mathbf{r}(\mathbf{x}_a) = \mathbf{b} - A\mathbf{x}_a = \begin{bmatrix} 0 \\ -2 \\ -5 \end{bmatrix}.$$

How far is the residual from the relative error? The answer is given in the following theorem:

**Theorem 3.4.1**

$$\frac{\|r(\boldsymbol{x}_a)\|}{\|A\|\|\boldsymbol{x}_a\|} \leq \frac{\|\boldsymbol{x}_e - \boldsymbol{x}_a\|}{\|\boldsymbol{x}_a\|} \leq \|A^{-1}\| \frac{\|r(\boldsymbol{x}_a)\|}{\|\boldsymbol{x}_a\|} \tag{3.4.2}$$

In other words, if we normalize the matrix $A$ such that $\|A\| = 1$, then the difference is about the condition number of $A$.

Note that the residual vector is the gradient vector of the function $f(\mathbf{x}) = \mathbf{x}^T A\mathbf{x}$ when $A$ is a symmetric positive definite matrix. It is the search direction of the steepest descent method in optimization and important basic concept in popular conjugate gradient (CG) method.

## 3.5 The direct $LU$ decomposition and Gaussian elimination algorithm for special matrices

For some situations and various considerations, we may not need to have pivoting process in the Gaussian elimination algorithm. This can be done using the direct $LU$ decomposition.

### 3.5.1 The direct $LU$ decomposition

Assuming that we do not do the pivoting, then we can have the direct $A = LU$ decomposition, that is we can have closed formulas for the entries of $L$ and $U$, where $L$ is a unit lower triangular matrix and $U$ is an upper triangular matrix[2].

- Tridiagonal or banded matrices

[2]Or we can have $U$ to be a unit upper triangular matrix and $L$ is a lower triangular one. The formulas and algorithm are slightly different.

- Column diagonally dominant matrices
- Symmetric positive definite matrices

### 3.5.2 Direct $LU$ decomposition

The direct $LU$ decomposition of a matrix $A$ without pivoting is equivalent to the Gaussian elimination process. It has more value in theoretical purpose and in deriving other algorithms, for example, the incomplete $LU$ decomposition in optimization. In the direct $LU$ decomposition, we get the entries of $L$ and $U$ directly according to certain order. We can write

$$A = LU = \begin{bmatrix} 1 & & & & \\ l_{21} & 1 & & & \\ l_{31} & l_{32} & 1 & & \\ \vdots & \ddots & \ddots & \ddots & \\ l_{n1} & l_{n2} & \cdots & l_{n,n-1} & 1 \end{bmatrix} \begin{pmatrix} u_{11} & u_{12} & \cdots & \cdots & u_{1n} \\ & u_{22} & \cdots & \cdots & u_{2n} \\ & & \ddots & \ddots & \vdots \\ & & & \ddots & \vdots \\ & & & & u_{nn} \end{pmatrix}.$$

To derive the formulae for the entries of $L$ and $U$, the order is very important: The order is as follows.

- We can get the first row of $U$ using the matrix multiplications (first of row of $L$ and any column of $U$)

  $$a_{1j} = 1u_{1j}, \qquad \Longrightarrow \qquad u_{1j} = a_{1j}, \qquad j = 1, 2, \cdots n.$$

  We have the first row of $U$.

  We multiply the $i$-th row of $L$ to the first column of $U$

  $$a_{i1} = l_{i1}u_{11}, \qquad \Longrightarrow \qquad l_{i1} = \frac{u_{i1}}{u_{11}}, \qquad i = 2, \cdots, n.$$

- Assume we have obtained the first $(k-1)$-th rows of $U$ and first $(k-1)$-th columns of $L$, we derive the formula for $k$-th row of $U$ and $k$-th column of $L$. If we multiply the $k$-th row of $L$ to $j$-th ($j \geq k$) column of $U$, we get

  $$a_{jk} = \sum_{i=1}^{k-1} l_{ik}u_{ik} + u_{jk}, \qquad \Longrightarrow \qquad u_{jk} = a_{jk} - \sum_{i=1}^{k-1} l_{ik}u_{ik}, \qquad j = k, \cdots, n.$$

  After we get the $k$-th row of $U$, we can get the $k$-th column of $L$ by multiplying the $i$-th row of $L$ to $k$-th column of $U$

  $$a_{ik} = \sum_{j=1}^{k-1} l_{ij}u_{jk} + l_{ik}u_{kk}, \qquad \Longrightarrow \qquad l_{ik} = \frac{a_{ik} - \sum_{j=1}^{k-1} l_{ij}u_{jk}}{u_{kk}}, \qquad i = k+1, \cdots, n.$$

## 3.6 Tridiagonal system of equations

The coefficient matrix $A$ of a tridiagonal system of has the following form

$$\begin{bmatrix} d_1 & \beta_1 & & & \\ \alpha_2 & d_2 & \beta_2 & & \\ & \alpha_3 & d_3 & \beta_3 & \\ & & \ddots & \ddots & \ddots \\ & & & \alpha_n & d_n \end{bmatrix}.$$

One particular application is the system of linear equations derived from the finite difference method. Another application is the alternating directional implicit (ADI) for solving two or three dimensional partial differential equations (PDE) using dimension by dimension approach.

If we use the direct $LU$ decomposition of the tridiagonal matrix, we see we get a very simple decomposition

$$\begin{bmatrix} d_1 & \beta_1 & & & \\ \alpha_2 & d_2 & \beta_2 & & \\ & \alpha_3 & d_3 & \beta_3 & \\ & & \ddots & \ddots & \ddots \\ & & & \alpha_n & d_n \end{bmatrix} = \begin{bmatrix} 1 & & & & \\ \alpha'_2 & 1 & & & \\ & \alpha'_3 & 1 & & \\ & & \ddots & \ddots & \\ & & & \alpha'_n & 1 \end{bmatrix} \begin{bmatrix} d'_1 & \beta_1 & & & \\ & d'_2 & \beta_2 & & \\ & & d'_3 & \beta_3 & \\ & & & \ddots & \ddots \\ & & & & d'_n \end{bmatrix}$$

In other words, only two diagonals need to be changed.

From the matrix-matric multiplication and the order of the direct $LU$ decomposition, it is easy to derive

$$\begin{aligned} & d'_1 = d_1, \qquad \alpha'_1 = \frac{\alpha_1}{d'_1} \\ & d'_i = d_i - \alpha'_i \beta_{i-1} \quad \text{from} \quad \alpha'_i \beta_{i-1} + d'_i = d_i \\ & \alpha'_{i+1} = \frac{\alpha_{i+1}}{d'_i} \quad \text{from} \quad \alpha'_{i+1} d'_i = \alpha_{i+1}. \end{aligned}$$

This decomposition is called the Crout factorization in some references.

Using overwriting, we can have the following pseudo-code:

for $i = 2, n$

$\alpha_i := \alpha_i / d_{i-1}$

$d_i := d_i - \alpha_i \beta_{i-1}$

end

Once we have the $LU$ factorization, we can easily derive the formulas for forward and backward substitutions for solving $A\mathbf{x} = \mathbf{b}$.

The forward substitution is

$$y_1 = b_1,$$

$$\text{for } i = 2, n,$$

$$y_i = b_i - \alpha_i \, y_{i-1},$$

$$\text{end}$$

The backward substitution is

$$x_n = \frac{y_n}{d_n},$$

$$\text{for } i = n-1, \ -1, \ 1,$$

$$x_i = \frac{y_i - \beta_i y_{i+1}}{d_i},$$

$$\text{end}$$

The entire process (Crout decomposition, forward and backward substitutions) quires $O(5n)$ multiplications/divisions. The solution process sometimes is called *chasing method* for solving the tridiagonal system of equations.

### 3.6.1 Strictly column diagonally dominant matrices

If a matrix $A = \{a_{ij}\}$ satisfies the following conditions

$$\sum_{i=1, i\neq j}^{n} |a_{ij}| < |a_{jj}|, \quad j = 1, 2, \cdots, n \tag{3.6.1}$$

For example, the matrix is the left below is strictly column diagonally dominant while the second is not.

$$\begin{bmatrix} 5 & 0 & 1 \\ -1 & 2 & -3 \\ 3 & -1 & 7 \end{bmatrix} \qquad \begin{bmatrix} 5 & 0 & 1 \\ -2 & 4 & 1 \\ 4 & 0 & 5 \end{bmatrix}$$

If a matrix $A$ is a strictly column diagonally dominant matrix, then it is obvious that no pivoting is necessary in the first step of Gaussian elimination process because

$$\left| \frac{a_{i1}}{a_{11}} \right| \leq \frac{\sum_{i=2}^{n} |a_{ij}|}{|a_{11}|} < 1.$$

The question is: how the next step, or thereafter. We have the following theorem answers this question:

**Theorem 3.6.1** *Let $A$ be a strictly column diagonally dominant matrix. After one step Gaussian elimination*

$$L_1 A = \begin{bmatrix} a_{11} & * \\ \boldsymbol{0} & A_1 \end{bmatrix}$$

$A_1$ is still a strictly column diagonally dominant matrix.

**Proof:**

$$\begin{aligned}
\sum_{i=2,i\neq j}^{n} |a_{ij}^{(2)}| &= \sum_{i=2,i\neq j}^{n} \left| a_{ij}^{(1)} - \frac{a_{i1}^{(1)}}{a_{11}^{(1)}} a_{1j}^{(1)} \right| \qquad \left( < |a_{jj}^{(2)}| ? \right) \\
&\leq \sum_{i=2,i\neq j}^{n} |a_{ij}^{(1)}| + \sum_{i=2,i\neq j}^{n} \left| \frac{a_{i1}^{(1)}}{a_{11}^{(1)}} \right| |a_{1j}^{(1)}| + \frac{|a_{j1}^{(1)}| - |a_{j1}^{(1)}|}{|a_{11}^{(1)}|} |a_{1j}^{(1)}| \\
&= |a_{jj}^{(1)}| - |a_{1j}^{(1)}| + \left| \frac{a_{1j}^{(1)}}{a_{11}^{(1)}} \right| \left( |a_{11}^{(1)}| - |a_{j1}^{(1)}| \right) \\
&< |a_{jj}^{(1)}| - \left| \frac{a_{1j}^{(1)}}{a_{11}^{(1)}} \right| |a_{j1}^{(1)}| \\
&\leq \left| a_{jj}^{(1)} - \frac{a_{1j}^{(1)}}{a_{11}^{(1)}} a_{j1}^{(1)} \right| = |a_{jj}^{(2)}|.
\end{aligned}$$

This completes the proof.

### 3.6.2 Symmetric positive definite matrices and Cholesky decomposition

A matrix is called a symmetric positive definite (SPD) if the following conditions are met

1. $A = A^H$, or $a_{ij} = \bar{a_{ji}}$.
2. For any $\mathbf{x} \neq 0$, $\mathbf{x}^H A \mathbf{x} > 0$.

Note that the second condition has the following equivalent statements, which also give some way to judge whether a matrix is an SPD or not.

- $\lambda_i(A) > 0$, $i = 1, 2, \cdots, n$, that is, all the eigenvalues of $A \in R^{n,n}$ are real and positive.
- All the determinants of the principal sub-matrices are positive.

A principal sub-matrix $A_k$ is the matrix composed from the intersections of the first $k$ the rows and columns of the original matrix $A$, for example

$$A_1 = \{a_{11}\}, \qquad A_2 \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}, \qquad A_3 = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}, \cdots, \qquad A_n = A.$$

If $A$ is an SPD. then so are $A_i$s.

Note also that if $A$ is an SPD, then $a_{ii} > 0$, $i = 1, 2, \cdots, n$, that is, all the diagonals are positive. This is because if we take $\mathbf{x} = \mathbf{e}_i$, then $\mathbf{x}^H A\mathbf{x} = a_{ii} > 0$.

Examples: Are the following matrixes ar symmetric positive definite matrices?

$$\begin{bmatrix} -4 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 8 \end{bmatrix}; \qquad \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -2 \\ 0 & -1 & 2 \end{bmatrix}$$

The first one is not since $a_{11} < 0$, or $a_{22} = 0$. For the second one, we have $A = A^T$ and $det(A_1) = 2 > 0$, $det(A_2) = 4 - 1 = 3 > 0$, and $det(A_3) = det(A) = 8 - 2 - 2 = 4 > 0$, so $A$ is an SPD.

### 3.6.3 Cholesky Decomposition $A = LL^T$

We prefer to use the $A = LL^T$ instead of $LU$ decomposition to take advantage of the symmetry and positiveness. The decomposition is an analogue to the square root of a positive number. We can save half of the cost and the storage compared to that the standard Gaussian elimination process applied to a general matrix. Note that for SPD matrices, no pivoting is necessary. If $\|A\| \sim 1$, then the growth factor $g(n) \leq \sqrt{n}$ and is well under control. Therefore no pivoting is necessary.

Now we derive the formula for $A = LL^T$ decomposition, the process is similar to the direct $LU$ decomposition. We can write

$$A = LL^T = \begin{bmatrix} l_{11} & & & & \\ l_{21} & l_{22} & & & \\ l_{31} & l_{32} & l_{33} & & \\ \vdots & \ddots & \ddots & \ddots & \\ l_{n1} & l_{n2} & \cdots & l_{n,n-1} & l_{nn} \end{bmatrix} \begin{pmatrix} l_{11} & l_{21} & \cdots & \cdots & l_{n1} \\ & l_{22} & \cdots & \cdots & l_{n2} \\ & & \ddots & \ddots & \vdots \\ & & & \ddots & \vdots \\ & & & & l_{nn} \end{pmatrix}.$$

To derive the formulae for the entries of $L$, the order is very important: The order is as follows.

- We can get the first column of $L$ using the matrix multiplications.

$$a_{11} = l_{11} l_{11}, \qquad \Longrightarrow \qquad l_{11} = \sqrt{a_{11}}.$$

- We can get the rest of the first column of $L$ using the matrix multiplications (first of row of $L$ and any column of $L^T$)

$$a_{1j} = l_{11} l_{j1}, \qquad \Longrightarrow \qquad l_{j1} = \frac{a_{1j}}{l_{11}}, \qquad j = 2, \cdots n.$$

  We have the first column of $L$.

- Assume we have obtained the first $(k-1)$-th columns of $L$, we derive the formula for $k$-th column of $L$. If we multiply the $k$-th row of $L$ to $k$-th column of $L^T$, we get

$$a_{kk} = \sum_{j=1}^{k-1} l_{kj} l_{kj} + l_{kk}^2, \qquad \Longrightarrow \qquad l_{kk} = \sqrt{a_{kk} - \sum_{j=1}^{k-1} l_{kj}^2},$$

  After we get the first element in the $k$-th column of $L$, we can get the the rest of $k$-th column of $L$ by multiplying the $i$-th row of $L$ to $j$-th column of $L^T$,

$$a_{ik} = \sum_{j=1}^{k-1} l_{ij} l_{kj} + l_{ik} l_{kk}, \qquad \Longrightarrow \qquad l_{ik} = \frac{a_{ik} - \sum_{j=1}^{k-1} l_{ij} l_{kj}}{l_{kk}}, \qquad j = k+1, \cdots, n.$$

**Pseudo-code of $A = LL^T$:**

for $k = 1, n$

$$l_{kk} = \sqrt{a_{kk} - \sum_{j=1}^{k-1} l_{kj}^2}$$

for $i = k+1, n$

$$l_{ik} = \left( a_{ik} - \sum_{j=1}^{k-1} l_{ij} l_{kj} \right) / l_{kk}$$

end

end

**Use the Cholesky decomposition ($A = LL^T$) to solve $A\mathbf{x} = \mathbf{b}$.**

From $A\mathbf{x} = \mathbf{b}$ we get $LL^T\mathbf{x} = \mathbf{b}$. The forward and backward substitutions are the following:

1. Solve $L\mathbf{y} = \mathbf{b}$.
2. Solve $L^T\mathbf{x} = \mathbf{y}$.

The number of multiplications/divisions needed for the Cholesky decomposition is $O(n^3/6)$. The storage needed is $O(n^2/2)$. So we just need half the storage and half the computations. The only disadvantage is that we need to evaluate square roots. A slightly different version is to get the $A = LDL^T$ decomposition which may work for any symmetric matrix assuming there is no break down (not guaranteed if $A$ is not an SPD).

### 3.6.4 Software issues

- In Matlab, we can use $x = A\backslash b$ for solving $A\mathbf{x} = \mathbf{b}$; $[l, u] = lu(A)$ for $LU$ decomposition; $chol(A)$ for Cholesky decomposition; $det(A)$ for determinant of $A$; $cond(A, 2)$, for example, to find the condition number of $A$ in 2-norm.

- Free subroutines using Fortran, C, and C++ on netlib (www.netlib.org) including Linpack, blas, (collection of linear algebra), blas (basic linear algebra subroutines), slatec, sparse, and many others in both single and double precision.

- There are many books on the programming of different methods. A popular one is *Numerical Recipes* in (Fortran, C, Pascal, ...).

# Chapter 4

# Iterative methods for solving linear system of equations

The Gaussian elimination method for solving $Ax = b$ is quite efficient if the size of $A$ is small to medium (in reference the available computers) and dense matrices (most of entries of the matrix are non-zero numbers). But for several reasons, sometimes an iterative method may be more efficient

- For sparse matrices, the Gaussian elimination method may destroy the structure of the matrix and cause '*fill-in*'s, see for example,

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 2 & 1 & 0 & 0 & 0 & 0 \\ 3 & 0 & 1 & 0 & 0 & 0 \\ 4 & 0 & 0 & 1 & 0 & 0 \\ 5 & 0 & 0 & 0 & 1 & 0 \\ 6 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \Longrightarrow \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & -1 & -2 & -2 & -2 & -2 \\ 0 & -3 & -2 & -3 & -3 & -3 \\ 0 & -4 & -4 & -3 & -4 & -4 \\ 0 & -5 & -5 & -5 & -4 & -5 \\ 0 & -6 & -6 & -6 & -6 & -5 \end{bmatrix}$$

  Obviously, the case discussed above can be generalized to a general $n$ by $n$ matrix with the same structure.

- Large sparse matrices for which we may not be able to store all the entries of the matrix. Below we show an example in two dimensions.

### 4.0.5 The central finite difference method with five point stencil for Poisson equation.

Consider the Poisson equation

$$u_{xx} + u_{yy} = f(x,y), \quad (x,y) \in \Omega = (a,b) \times (c,d), \tag{4.0.1}$$

$$u(x,y)|_{\partial\Omega} = u_0(x,y), \quad \text{Dirichlet BC.} \tag{4.0.2}$$

If $f \in L^2(\Omega)$, then the solution exists and it is unique. Analytic solution is rarely available. Now we discuss how to use the finite difference equation to solve the Poisson equation.

- Step 1: Generate a grid. A uniform Cartesian grid can be used:

$$x_i = a + ih_x, \quad i = 0,1,2,\cdots m, \quad h_x = \frac{b-a}{m}, \tag{4.0.3}$$

$$y_j = c + jh_y, \quad j = 0,1,2,\cdots n, \quad h_y = \frac{d-c}{n}. \tag{4.0.4}$$

  We want to find an approximate solution $U_{ij}$ to the exact solution at all the grid points $(x_i, y_j)$ where $u(x_i, y_j)$ is unknown. So there are $(m-1)(n-1)$ unknown for Dirichlet boundary condition.

- Step 2: Substitute the partial derivatives with a finite difference formula in terms of the function values at grid points to get.

$$\frac{u(x_{i-1},y_j) - 2u(x_i,y_j) + u(x_{i+1},y_j)}{(h_x)^2} + \frac{u(x_i,y_{j-1}) - 2u(x_i,y_j) + u(x_i,y_{j+1})}{(h_y)^2}$$
$$= f_{ij} + T_{ij}, \quad i = 1,\cdots m-1, \quad j = 1,\cdots n-1,$$

  where $f_{ij} = f(x_i, y_j)$. The local truncation error satisfies

$$T_{ij} \sim \frac{(h_x)^2}{12}\frac{\partial^4 u}{\partial x^4} + \frac{(h_y)^2}{12}\frac{\partial^4 u}{\partial y^4}. \tag{4.0.5}$$

  Define

$$h = \max\{\, h_x,\ h_y \,\} \tag{4.0.6}$$

  The finite difference discretization is consistent if

$$\lim_{h\to 0} \|\mathbf{T}\| = 0. \tag{4.0.7}$$

  Therefore the discretization is consistent and second order accurate.

  If we remove the error term in the equation above, and replace the exact solution $u(x_i, y_j)$ with the approximate solution $U_{ij}$ which is the solution of the linear system of equations

$$\frac{U_{i-1,j} + U_{i+1,j}}{(h_x)^2} + \frac{U_{i,j-1} + U_{i,j+1}}{(h_y)^2} - \left(\frac{2}{(h_x)^2} + \frac{2}{(h_y)^2}\right) U_{ij} = f_{ij} \tag{4.0.8}$$

The finite difference scheme at a grid point $(x_i, y_j)$ involves five grid points, east, north, west, south, and the center. The center is called the master grid point.

- Solve the linear system of equations to get an approximate solution at grid points (how?).

- Error analysis, implementation, visualization etc.

### 4.0.6 Matrix-vector form of the finite difference equations.

Generally, if one wants to use a direct method such as Gaussian elimination method or sparse matrix techniques, then one needs to find out the matrix structure. If one use an iterative method, such as Jacobi, Gauss Seidel, SOR($\omega$) methods, then it may be not necessarily to have the matrix and vector form.

In the matrix vector form $A\mathbf{U} = \mathbf{F}$, the unknown is a one dimensional array. For the two dimensional Poisson equations, the unknowns $U_{ij}$ are a two dimensional array. Therefore we need to order it to get a one dimensional array. We also need to order the finite difference equations. It is common practice that we use the same ordering for the equations and for the unknowns.

There are two commonly used ordering. One is called the *natural ordering* that fits sequential computers. The other one is called the *red and black ordering* that fits parallel computers.

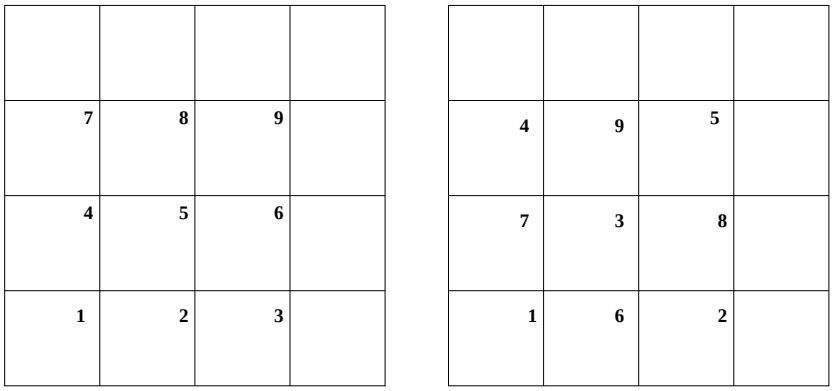


Figure 4.1: The natural ordering (left) and the red-black ordering (right).

**The natural row ordering.**

In the natural row ordering, we order the unknowns/equations row-wise, therefore the $k$-th equation corresponding to $(i, j)$ with the following relation

$$k = i + (m-1)(j-1), \quad i = 1, 2, \cdots, m-1, \quad j = 1, 2, \cdots, n-1. \tag{4.0.9}$$

We use the following example to verify the matrix-vector form of the finite difference equations.

Assume that $h_x = h_y = h$, $m = n = 4$, so we will have nine equations and nine unknowns. The coefficient matrix is 9 by 9! To write down the matrix-vector form, we use a one-dimensional array $\mathbf{x}$ to express the unknown $U_{ij}$.

$$\begin{aligned} x_1 = U_{11}, \quad x_2 = U_{21}, \quad x_3 = U_{31}, \quad x_4 = U_{12}, \quad x_5 = U_{22}, \\ x_6 = U_{32}, \quad x_7 = U_{13}, \quad x_8 = U_{23}, \quad x_9 = U_{33}. \end{aligned} \tag{4.0.10}$$

If we order the equations the same way as we order the unknowns, then the nine equations from the standard central finite difference scheme using the five point stencil are

$$\begin{aligned} \frac{1}{h^2}(-4x_1 + x_2 + x_4) &= f_{11} - \frac{u_{01} + u_{10}}{h^2}, \\ \frac{1}{h^2}(x_1 - 4x_2 + x_3 + x_5) &= f_{21} - \frac{u_{20}}{h^2} \\ \frac{1}{h^2}(x_2 - 4x_3 + x_6) &= f_{31} - \frac{u_{30} + u_{41}}{h^2} \\ \frac{1}{h^2}(x_1 - 4x_4 + x_5 + x_7) &= f_{12} - \frac{u_{02}}{h^2} \\ \frac{1}{h^2}(x_2 + x_4 - 4x_5 + x_6 + x_8) &= f_{22} \\ \frac{1}{h^2}(x_3 + x_5 - 4x_6 + x_9) &= f_{32} - \frac{u_{42}}{h^2} \\ \frac{1}{h^2}(x_4 - 4x_7 + x_8) &= f_{13} - \frac{u_{03} + u_{14}}{h^2} \\ \frac{1}{h^2}(x_5 + x_7 - 4x_8 + x_9) &= f_{23} - \frac{u_{24}}{h^2} \\ \frac{1}{h^2}(x_6 + x_8 - 4x_9) &= f_{33} - \frac{u_{34} + u_{43}}{h^2}. \end{aligned}$$

Now we can write down the coefficient matrix easily. It is *block tridiagonal* and has the following form:

$$A = \frac{1}{h^2} \begin{bmatrix} B & I & 0 \\ I & B & I \\ 0 & I & B \end{bmatrix} \tag{4.0.11}$$

where $I$ is a $3 \times 3$ identity matrix:

$$B = \begin{bmatrix} -4 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & -4 \end{bmatrix}$$

For a general $n$ by $n$ grid, we will have

$$A = \frac{1}{h^2} \begin{bmatrix} B & I & & \\ I & B & I & \\ & \ddots & \ddots & \ddots \\ & & I & B \end{bmatrix}, \qquad B = \begin{bmatrix} -4 & 1 & & \\ 1 & -4 & 1 & \\ & \ddots & \ddots & \ddots \\ & & 1 & -4 \end{bmatrix}.$$

Note that $-A$ is a symmetric positive definite matrix and it is weakly diagonally dominant. Therefore $A$ is non-singular and there is a unique solution.

The matrix-vector form is useful to understand the structure of the linear system of equations, and it may be necessary if a direct method (such as Gaussian elimination) or sparse matrix techniques are used for solving the system. However, it is more convenient sometimes to use the two parameters system $(i, j)$, especially if an iterative method is used to solve the system. It is more intuitive and useful to visualize the data using two index system.

The eigenvalues and eigenvectors of $A$ can be indexed by two parameters $p$ and $k$ corresponding to wave numbers in the $x$ and $y$ directions. The $(p, k)$-th eigenvector $u^{p,k}$ has $n^2$ elements for a $n$ by $n$ matrix of the form above:

$$u_{ij}^{p,k} = \sin(p\pi ih)\, \sin(k\pi jh), \quad i, j = 1, 2, \cdots n \tag{4.0.12}$$

for $p, k = 1, 2, \cdots n$. The corresponding eigenvalues are

$$\lambda^{p,k} = \frac{2}{h^2} \Big( \cos(p\pi h) - 1) + \cos(k\pi h) - 1) \Big). \tag{4.0.13}$$

The least dominant eigenvalue ( the smallest in the magnitude) is

$$\lambda^{1,1} = -2\pi + O(h^2). \tag{4.0.14}$$

The dominant eigenvalue (the largest in the magnitude) is

$$\lambda^{n/2,n/2} \sim -\frac{4}{h^2}. \tag{4.0.15}$$

Therefore we have the following estimates:

$$\begin{aligned} &\|A\|_2 \sim \max |\lambda^{p,k}| = \frac{4}{h^2}, \qquad \|A^{-1}\|_2 = \frac{1}{\min |\lambda^{p,k}|} \sim \frac{1}{2\pi}, \\ &cond_2(A) = \|A\|_2 \|A^{-1}\|_2 \sim \frac{2}{\pi h^2} = O(n^2). \end{aligned} \tag{4.0.16}$$

Since the condition number is considered to be large, we should use double precision to reduce the effect of round off errors.

## 4.1 Basic iterative methods for solving linear system of equations

The idea of iterative methods is to start with an initial guess, then improve the solution iteratively. The first step is to re-write the original equation $f(\mathbf{x}) = 0$ to an equivalent form $\mathbf{x} = g(\mathbf{x})$ and then we can form an iteration: $\mathbf{x}^{(k+1)} = g(\mathbf{x}^{(k)})$. For example, to find $\sqrt[3]{5}$ is equivalent to solving the equation $x^3 - 5 = 0$. This equation can be written as $x = 5/x^2$ or $x = x - \frac{x^3-5}{3x^2}$. For the second one, the iteration

$$x^{(k+1)} = x^{(k)} - \frac{(x^{(3)})^3}{3(x^{(2)})^2}, \qquad k = 0, 1, \cdots$$

is called the Newton's iterative method. The mathematical theory behind this is the *fixed point theory.*

For a linear system of equations $A\mathbf{x} = \mathbf{b}$, we hope to re-write it as an equivalent form $\mathbf{x} = R\mathbf{x} + \mathbf{c}$ so that we can form an iteration $\mathbf{x}^{(k+1)} = R\mathbf{x}^{(k)} + \mathbf{c}$ given an initial guess $\mathbf{x}^0$. We want to choose such a $R$ and $c$ that $\lim_{k\to\infty} \mathbf{x}^{(k)} = \mathbf{x}_e = A^{-1}\mathbf{b}$. A common method is called the splitting approach in which we re-write the matrix $A$ as

$$A = M - K, \qquad det(M) \neq 0. \tag{4.1.1}$$

Then $A\mathbf{x} = \mathbf{b}$ can be written as $(M-K)\mathbf{x} = \mathbf{b}$, or $M\mathbf{x} = K\mathbf{x}+\mathbf{b}$, or $\mathbf{x} = M^{-1}K\mathbf{x}+M^{-1}\mathbf{b}$, or $\mathbf{x} = R\mathbf{x}+\mathbf{c}$, where $R = M^{-1}K$ is called the iteration matrix and $c = M^{-1}\mathbf{b}$ is a constant vector. The iterative process is then given an initial guess $\mathbf{x}^{(0)}$, we can get a sequence of $\{\mathbf{x}^{(k)}\}$ according to

$$\mathbf{x}^{(k+1)} = R\mathbf{x}^{(k)} + \mathbf{c} \tag{4.1.2}$$

We first discuss three basic iterative methods for solving $A\mathbf{x} = \mathbf{b}$. To derive the three methods, we re-write the matrix $A$ as

$$A = D - L - U = \begin{bmatrix} a_{11} & & & & \\ & a_{22} & & & \\ & & a_{33} & & \\ & & & \ddots & \\ & & & & a_{nn} \end{bmatrix} - \begin{bmatrix} 0 & & & & \\ -a_{21} & 0 & & & \\ -a_{31} & -a_{32} & 0 & & \\ \vdots & \ddots & \ddots & \ddots & \\ -a_{n1} & -a_{n2} & \cdots & -a_{n,n-1} & 0 \end{bmatrix}$$

$$- \begin{pmatrix} 0 & -a_{12} & \cdots & \cdots & -a_{1n} \\ & 0 & \cdots & \cdots & -a_{2n} \\ & & \ddots & \ddots & \vdots \\ & & & \ddots & \vdots \\ & & & & 0 \end{pmatrix}$$

## 4.2 The Jacobi iterative method: Solve for the diagonals

The matrix vector form of the The Jacobi iterative method can de derived as follows:

$$(D - L - U)\mathbf{x} = \mathbf{b}$$

$$D\mathbf{x} = (L + U)\mathbf{x} + \mathbf{b}$$

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

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

The component form can be written as

$$x_i^{(k+1)} = \left( b_i - \sum_{j=1, j\neq i}^{n} a_{ij} x_j^{(k)} \right) / a_{ii}, \qquad i = 1, 2, \cdots, n. \tag{4.2.1}$$

The component form is useful for implementation while the matrix-vector form is good for convergence analysis.

## 4.3 The Gauss-Seidel iterative method: Use the most updated

In the Jacobi iterative method, when we compute $x_2^{k+1}$, we have already computed $x_1^{k+1}$. Assume that $x_1^{k+1}$ is a better approximation than $x_1^k$, why can not we use $x_1^{k+1}$ when we update $x_2^{k+1}$ instead of $x_1^k$? With this idea, we get a new iterative method which is the Gauss-Seidel iterative method for solving $A\mathbf{x} = \mathbf{b}$. The component form is

$$x_i^{(k+1)} = \left( b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} \right) / a_{ii}, \qquad i = 1, 2, \cdots, n. \tag{4.3.1}$$

To derive the matrix-vector form of the Gauss-Seidel iterative method, we write the component form above to a form $(\quad)^{(k+1)} = (\quad)^{(k)} + (\quad)$. The component form above is equivalent to

$$\sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} + a_{ii} x_i^{(k+1)} = b_i - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)}, \qquad i = 1, 2, \cdots, n, \tag{4.3.2}$$

which is the component form of the following system of equations

$$\mathbf{x}^{(k+1)} = U\mathbf{x} + \mathbf{b}, \quad \text{or} \quad \mathbf{x}^{(k+1)} = (D - L)^{-1}U\mathbf{x} + (D - L)^{-1}\mathbf{b}.$$

Thus the iteration matrix of the Gauss-Seidel iterative method is $(D - L)^{-1}U$, and the constant vector is $\mathbf{c} = (D - L)^{-1}\mathbf{b}$.

### 4.3.1 Implementation details

The iterative methods are an infinity process. However, when we implement on a computer, we have to stop it in finite time. One of several of the following stopping criteria are used

- $\|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\| \leq tol.$
- $\frac{\|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\|}{\|\mathbf{x}^{(k)}\|} \leq tol.$
- $\|r(\mathbf{x}^{(k)})\| \leq tol.$
- $k \geq k_{max},$

where $tol$ and $k_{max}$ are two given parameters.

### 4.3.2 Pseudo-code of the Gauss-Seidel iterative method:

```
[x,k] = my_gs(n,a,b,x0,tol)
error = 1e5; x=x0; k=0;
while error > tol
for i=1:n
  x(i) = b(i);
  for j=1:n
    if j~= i
       x(i) = x(i) - a(i,j)*x(j);
  end
  x(i) = x(i)/a(i,i);
end
error = norm(x-x0);   %defaut is 2-norm
x0=x; k = k+1;        % replace the old value, add the counter.
end    %end while
```

### 4.3.3 The Gauss-Seidel iterative method for 2-point boundary value problem

Following section 1.3, we have

$$\frac{U_{i-1} - 2U_i + U_{i+1}}{h^2} = f(x_i), \quad i = 1, 2, \cdots, n-1.$$

If we use the same ordering for the equations and unknowns, then the diagonals are always $-2/h^2$, the Jacobi iteration is simply

$$U_i^{(k+1)} = \frac{U_{i-1}^{(k)} + U_{i+1}^{(k)}}{2} + \frac{h^2}{2} f(x_i, y_j), \qquad i = 1, 2, \cdots, n-1.$$

No matrix is formed (or only matrix-vector multiplication is needed), no ordering is necessary (assuming that the equations and unknowns have the same ordering). For the Gauss-Seidel iteration, from $k$-th to $k+1$-th iteration, we can use the following

$$U_i^{(k+1)} = U_i^{(k)}, \qquad i = 1, 2, \cdots, n-1,$$
$$U_i^{(k+1)} = \frac{U_{i-1}^{(k+1)} + U_{i+1}^{(k+1)}}{2} + \frac{h^2}{2} f(x_i), \qquad i = 1, 2, \cdots, n-1.$$

If $U_i^{(k+1)}$ has not been updated, then it use the value from $k$-th iteration, otherwise it uses the most updated one.

### 4.3.4 The Gauss-Seidel iterative method for the finite difference method for Poisson equation

For the Poisson equation $u_{xx} + u_{yy} = f(x, y)$, if we use the standard 5-point central finite difference scheme

$$\frac{U_{i-1,j} + U_{i+1,j} - 4U_{ij} + U_{i,j-1} + U_{i,j+1}}{h^2} = f(x_i, y_j)$$

and the same ordering for the equations and unknowns, then the Jacobi iteration is

$$U_{ij}^{(k+1)} = \frac{U_{i-1,j}^{(k)} + U_{i+1,j}^{(k)} + U_{i,j-1}^{(k)} + U_{i,j+1}^{(k)}}{4} + \frac{h^2}{4} f(x_i, y_j),$$
$$i = 1, 2, \cdots, n-1, \quad j = 1, 2, \cdots, n-1,$$

if the solution is prescribed along the boundary (Dirichlet BC). Again, no matrix is needed, no ordering is necessary. We do not need to transform the two dimensional array to a one dimensional one. The implementation is rather simple.

## 4.4 The successive over-relaxation (SOR($\omega$)) iterative method

The Jacobi and Gauss-Seidel methods can be quite slow. The SOR($\omega$) iterative method is an acceleration method by choosing appropriate parameter $\omega$. The SOR($\omega$) iterative method is

$$\mathbf{x}^{(k+1)} = (1-\omega)\mathbf{x}^k + \omega \tilde{\mathbf{x}}_{GS}^{k+1}. \tag{4.4.1}$$

The component form is

$$x_i^{k+1} = (1-\omega)x_i^k + \omega \left( \left( b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} \right) / a_{ii} \right). \tag{4.4.2}$$

Note that, it is incorrect to get G-S result first, then do the linear interpolation.

The idea of the SOR method is to interpolate $\mathbf{x}^k$ and $\mathbf{x}_{GS}^{k+1}$ to get a better approximation. When $\omega \leq 1$, the new point of $(1-\omega)\mathbf{x}^k + \omega\tilde{\mathbf{x}}_{GS}^{k+1}$ is between $\mathbf{x}^k$ and $\mathbf{x}_{GS}^{k+1}$, and this it is called interpolation. The iterative method is called under relaxation. When $\omega > 1$, the new point of $(1-\omega)\mathbf{x}^k + \omega\tilde{\mathbf{x}}_{GS}^{k+1}$ is outside $\mathbf{x}^k$ and $\mathbf{x}_{GS}^{k+1}$, and this it is called extrapolation. The iterative method is called over relaxation. Since the approach is used at every iteration, it is called successive over relaxation (SOR) method.

To derive the matrix-vector form of SOR($\omega$) method, we write its component form as

$$a_{ii}x_i^{k+1} + \omega\sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} = a_{ii}(1-\omega)x_i^k + \omega b_i - \omega\sum_{j=i+1}^{n} a_{ij}x_j^{(k)}. \tag{4.4.3}$$

This is equivalent to

$$(D-\omega L)\mathbf{x}^{(k+1)} = \left((1-\omega)D + \omega U\right)\mathbf{x}^{(k)} + \omega\mathbf{b}.$$

Thus the iteration matrix and constant vector of the SOR($\omega$) method are

$$R_{SOR}(\omega) = (D-\omega L)^{-1}\left((1-\omega)D + \omega U\right), \qquad c_{SOR} = \omega(D-\omega L)^{-1}. \tag{4.4.4}$$

## 4.5 Convergence of basic iteration methods $\mathbf{x}^{(k+1)} = R\mathbf{x}^{(k)} + \mathbf{c}$

Using an iterative method, we will get a vector sequence of $\{\mathbf{x}^{(k)}\}$ and we know how to tell whether it is convergent or not. However, for an iterative method, we need to consider all possible initial guesses and constant vector $\mathbf{c}$.

If the vector sequence of $\{\mathbf{x}^{(k)}\}$ converges to $\mathbf{x}^*$, then by taking limit on both sides of the iterative scheme, we have

$$\mathbf{x}^* = R\mathbf{x}^* + \mathbf{c}. \tag{4.5.1}$$

The above equality is called the consistency condition.

**Definition 4.5.1** *The iteration methods $\boldsymbol{x}^{(k+1)} = R\boldsymbol{x}^{(k)} + \boldsymbol{c}$ is convergent if for any initial guess $\boldsymbol{x}^{(0)}$ and constant vector $\mathbf{c}$, the vector sequence of $\{\boldsymbol{x}^{(k)}\}$ converges to the solution of the system of equations $\boldsymbol{x}^* = R\boldsymbol{x}^* + \boldsymbol{c}$.*

Now we discuss a few sufficient conditions that guarantee convergence of a basic iterative method.

**Theorem 4.5.1** *If there is an associated matrix norm such that $\|R\| < 1$, then the iteration method $\boldsymbol{x}^{(k+1)} = R\boldsymbol{x}^{(k)} + \boldsymbol{c}$ converges.*

**Proof:** Let $\mathbf{e}^{(k)} = \mathbf{x}^{(k)} - \mathbf{x}^*$, from the iterative method $\mathbf{x}^{(k+1)} = R\mathbf{x}^{(k)} + \mathbf{c}$ and the consistency condition $\mathbf{x}^* = R\mathbf{x}^* + \mathbf{c}$, we have

$$\mathbf{e}^{(k+1)} = R\mathbf{e}^{(k)}$$

$$0 \leq \|\mathbf{e}^{(k+1)}\| = \|R\mathbf{e}^{(k)}\| \leq \|R\|\|\mathbf{e}^{(k)}\| \leq \|R\|\|R\|\|\mathbf{e}^{(k-1)}\| \leq \cdots \leq \|R\|^{k+1}\|\mathbf{e}^{(0)}\|$$

Thus we conclude that $\lim_{k\to\infty} \|\mathbf{e}^{(k)}\| = 0$, or equivalently, $\lim_{k\to\infty} \mathbf{x}^{(k)} = \mathbf{x}^{(*)}$.

**Example:** Given

$$R = \begin{pmatrix} 0.9 & 0.05 \\ -0.8 & -0.1 \end{pmatrix}$$

Does the iterative method $\mathbf{x}^{(k+1)} = R\mathbf{x}^{(k)} + \mathbf{c}$ converge?

We can easily get $\|R\|_1 = 1.7$, which leads to no conclusion; and $\|R\|_\infty\| = 0.95 < 1$, which lead to the conclusion that the iterative method converges.

### 4.5.1 Convergence speed

In the theorem above, the $k$-th error depends on the initial one that we do not know. The following error estimate does not need the initial error.

**Theorem 4.5.2** *If there is an associated matrix norm such that $\|R\| < 1$, we have the following error estimate for the iteration method $\boldsymbol{x}^{(k+1)} = R\boldsymbol{x}^{(k)} + \boldsymbol{c}$.*

$$\|\boldsymbol{e}^{(k+1)}\| \leq \frac{\|R\|^k}{1 - \|R\|} \|\boldsymbol{x}^{(1)} - \boldsymbol{x}^{(0)}\|. \tag{4.5.2}$$

Proof: From the iterative method $\mathbf{x}^{(k+1)} = R\mathbf{x}^{(k)} + \mathbf{c}$, we also have f $\mathbf{x}^{(k)} = R\mathbf{x}^{(k-1)} + \mathbf{c}$. Subtracting the two, we get

$$\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)} = R(\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}) = R^2(\mathbf{x}^{(k-1)} - \mathbf{x}^{(k-2)} = \cdots = R^k(\mathbf{x}^{(1)} - \mathbf{x}^{(0)}).$$

Since

$$\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)} = \mathbf{x}^{(k+1)} - \mathbf{x}^{(*)} + \mathbf{x}^* - \mathbf{x}^{(k)} = \mathbf{e}^{(k+1)} - \mathbf{e}^{(k)} = (R - I)\mathbf{e}^{(k)}.$$

Combining the two equalities above we get

$$-(I - R)\mathbf{e}^{(k)} = R^k \left(\mathbf{x}^{(1)} - \mathbf{x}^{(0)}\right).$$

This leads to

$$\|\mathbf{e}^{(k)}\| = \|(I - R)^{-1}R^k \left(\mathbf{x}^{(1)} - \mathbf{x}^{(0)}\right)\|.$$

Finally from the Banach's lemma, we have

$$\|\mathbf{e}^{(k)}\| \leq \frac{\|R\|^k}{1 - \|R\|} \|\mathbf{x}^{(1)} - \mathbf{x}^{(0)}\|.$$

### 4.5.2 Other sufficient conditions using the original matrix $A$

**Theorem 4.5.3** *If $A$ is strictly row diagonally dominant matrix, then both Jacobi and Gauss-Seidel methods converge. The Gauss-Seidel method converges faster in the sense that*

$$\|R_{GS}\|_\infty \leq \|R_J\|_\infty < 1$$

**Proof:** The proof of the first part is easy. For the Jacobi method, we have $R = D^{-1}(L+U)$, thus

$$\|R_J\|_\infty = \max_i \sum_{j=1, j\neq i}^{n} \left| \frac{a_{ij}}{a_{ii}} \right| < 1.$$

The proof for the Gauss-Seidel method is not trivial and long, we refer the readers to the book [J. W. Demmel] on page 287-288.

For general matrices, it is unclear whether the Jacobi or Gauss-Seidel method converges faster even if they both converge.

**Theorem 4.5.4** *If $A$ is a symmetric positive definite (SPD) matrix, then the SOR($\omega$) method converges for $0 < \omega < 2$.*

Again we refer the readers to the book [J. W. Demmel] on page 290-291.

**Theorem 4.5.5** *If $A$ is a weakly row diagonally dominant matrix,*

$$\sum_{j=1, j\neq i}^{n} |a_{ij}| \leq |a_{ii}|, \qquad , i = 1, 2, \cdots, n,$$

*with at least on inequality is strictly and $A$ is irreducible, that is, there is no permutation matrix such that*

$$P^T A P = \begin{pmatrix} A_{11} & A_{12} \\ 0 & A_{22} \end{pmatrix}$$

*then both Jacobi and Gauss-Seidel methods converge. The Gauss-Seidel method converges faster in the sense that*

$$\|R_{GS}\|_\infty \leq \|R_J\|_\infty < 1.$$

**Lemma 4.5.1** *$A$ is irreducible if the graph of the matrix is strongly connected.*

### 4.5.3 A sufficient and necessary condition

The spectral radius of a matrix $A$ is defined as

$$\rho(A) = \max_i |\lambda_i(A)| \tag{4.5.3}$$

Note that from $\rho(A) \leq \|A\|$ from any associated matrix norms. The proof is quite simple. Let $\lambda_{i^*}$ be the eigenvalue of $A$ such that $\rho(A) = |\lambda_{i^*}|$ and $\mathbf{x}^* \neq \mathbf{0}$ is the corresponding eigenvector, then we have $A\mathbf{x}^* = \lambda_{i^*}\mathbf{x}^*$. Thus we have $|\lambda_{i^*}|\|\mathbf{x}^*\| \leq \|A\|\|\mathbf{x}^*\|$. Since $\|\mathbf{x}^*\| \neq 0$, we get $\rho(A) \leq \|A\|$.

**Theorem 4.5.6** *An iteration method $\boldsymbol{x}^{(k+1)} = R\boldsymbol{x}^{(k)} + \boldsymbol{c}$ converges for arbitrary $\boldsymbol{x}^{(0)}$ and $\boldsymbol{c}$ if and only if $\rho(R) < 1$.*

**Proof: Part A:** If the iterative method converge, then $\rho(R) < 1$. This can be done using counter proof method. Assume that $\rho(R) > 1$, then let $A\mathbf{x}^* = \lambda_{i^*}\mathbf{x}^*$, with $\rho(A) = |\lambda_{i^*}| > 1$ and $\|\mathbf{x}^*\| \neq 0$. If we set $\mathbf{x}^{(0)} = \mathbf{x}^*$ and $\mathbf{c} = \mathbf{0}$, then we have $\mathbf{x}^{(k+1)} = \lambda_{i^*}^{k+1}\mathbf{x}^*$ which dost not have a limit since $\lambda_{i^*}^{k+1} \to \infty$. The case of $\rho(R) = 1$ is left as an exercise.

**Proof: Part B:** If $\rho(R) < 1$, then the iterative method converges for arbitrary $\mathbf{x}^{(0)}$ and $\mathbf{c}$. They key in the proof is to find a matrix norm such that $\|R\| < 1$.

From liner algebra Jordan's theorem we know that for any square matrix $R$, it is similar to a Jordan canonical form, that is, there is a nonsingular matrix $S$ such that

$$S^{-1}RS = \begin{pmatrix} J_1 & & & & \\ & J_2 & & & \\ & & \ddots & & \\ & & & \ddots & \\ & & & & J_p \end{pmatrix}, \qquad \begin{pmatrix} \lambda_i & 1 & & & \\ & \lambda_i & 1 & & \\ & & \ddots & \ddots & \\ & & & \ddots & 1 \\ & & & & \lambda_i \end{pmatrix},$$

Note that $\|S^{-1}RS\|\infty = \rho(R) < 1$ if all Jordan blocks are 1 by 1 matrix, otherwise $\|S^{-1}RS\|\infty \leq \rho(R) + 1$. Since $\rho(R) < 1$, we can find $< 0\epsilon$ such that $\rho(R) + \epsilon < 1$, say $\epsilon = (1 - \rho(R))/2$. Consider a particular Jordan Block $J_i$ and assume it is a $k$ by $k$ matrix. Let

$$D_i(\epsilon) = \begin{pmatrix} 1 & & & & \\ & \epsilon & & & \\ & & \ddots & & \\ & & & \ddots & \\ & & & & \epsilon^{k-1} \end{pmatrix}, \qquad D_i(\epsilon)J_iD_i^{-1}(\epsilon) = \begin{pmatrix} \lambda_i & \epsilon & & & \\ & \lambda_i & \epsilon & & \\ & & \ddots & \ddots & \\ & & & \ddots & \epsilon \\ & & & & \lambda_i \end{pmatrix},$$

Using this diagonal matrix, we can get

$$D(\epsilon) = \begin{pmatrix} D_1(\epsilon) & & & & \\ & D_1(\epsilon) & & & \\ & & \ddots & & \\ & & & \ddots & \\ & & & & D_p(\epsilon) \end{pmatrix},$$

Thus $\|D^{-1}(\epsilon)S^{-1}RSD\|_\infty = \rho(R) + \epsilon < 1$.

Finally we define a new matrix norm as

$$\|R\|_{new} = \|D^{-1}(\epsilon)S^{-1}RSD\|_\infty \tag{4.5.4}$$

It can easily show that the definition above is indeed an associate matrix norm and $\|R\|_{new} = \rho(R) + \epsilon < 1$, we conclude that the iterative method converges for any initial guess and vector $\mathbf{c}$.

## 4.6 Discussion for the Poisson equations, what is the best $\omega$?

For the finite difference methods for Poisson equations in 1D and 2D, we can find the eigenvalues of the coefficient matrix, which also leads to eigenvalues of the iteration matrix (Jacobib, Gauss Seidel, SOR$\omega$). For 1D model problem $u''(x) = f(x)$ with the Dirichlet boundary condition $u(0)$ and $u(1)$ are prescribed, the coefficient matrix is

$$A_{FD} = \frac{1}{h^2}\begin{bmatrix} -2 & 1 & & & \\ 1 & -2 & 1 & & \\ & 1 & -2 & 1 & \\ & & \ddots & \ddots & \ddots \\ & & & 1 & -2 \end{bmatrix}, \qquad A = \begin{bmatrix} -2 & 1 & & & \\ 1 & -2 & 1 & & \\ & 1 & -2 & 1 & \\ & & \ddots & \ddots & \ddots \\ & & & 1 & -2 \end{bmatrix}.$$

The matrix is weakly row diagonally dominant (not strictly) and irreducible; $-A$ is symmetric positive definite; or $A$ is symmetric negative definite.

**Theorem 4.6.1** *For the above $n$ by $n$ matrix, we have*

$$\rho(R_J) = \max_{1\le i\le n}\left|1 + \frac{\lambda_i(A)}{2}\right|, \tag{4.6.1}$$

*where $\lambda_i(A)$, $i = 1, 2, \cdots, n$ are the eigenvalues of the matrix $A$, $R_J$ is the iteration matrix of the Jacobi method.*

**Proof:** We know that $R_j = D^{-1}(L+U)$ and $D = -2I$. Let $\lambda$ be an eigenvalue of $J_R$, then $det(\lambda I - J_R) = 0$, or $det(\lambda I - D^{-1}(L+U)) = 0$, or $det(D^{-1}(\lambda D - (L+U)) = 0$, or $det(D)det(\lambda D - (L+U)) = 0$. Since $det(D) \neq 0$, we have $det(\lambda D - (L+U)) = 0$, or $det((\lambda-1)D + D - (L+U)) = 0$, or $det((\lambda-1)D + A) = 0$, or $det((1-\lambda)D - A) = 0$, or $det(-2(1-\lambda)I - A) = 0$ since $D = -2I$. Thus $-2(1-\lambda)$ is an eigenvalue of $A$, or

$$\lambda_i(R_J) = 1 + \frac{\lambda_i(A)}{2}, \quad i = 1, 2, \cdots, n.$$

From the relation above, we have the theorem right way.

The following lemma gives the eigenvalues of a tri-diagonal matrix.

**Lemma 4.6.1** *Let $A$ be the following $n$ by $n$ matrix*

$$A = \begin{bmatrix} \alpha & \beta & & \\ \beta & \alpha & \beta & \\ & \ddots & \ddots & \ddots \\ & & \beta & \alpha \end{bmatrix}.$$

*The eigenvalues of $A$ are the following*

$$\lambda_k = \alpha + 2\beta \cos\frac{k\pi}{n+1}, \quad k = 1, 2, \cdots, n.$$

*The eigenvector corresponding to $\lambda_k$ is*

$$\chi_{k,j} = \sin\frac{kj\pi}{n+1}, \quad j = 1, 2, \cdots, n.$$

**Proof**: It is easy to check that $A\chi_k = \lambda_k \chi_k$.

When $\alpha = -2$, $\beta = 1$, we have

$$\lambda_k = -2 + 2\cos\frac{k\pi}{n+1}, \quad k = 1, 2, \cdots, n.$$

Thus the spectral radius of the Jacobi iterative method for 1D Poisson problem is

$$\begin{aligned} \rho(R_J) &= \max_{1\le k\le n}\left|1 + \frac{\lambda_k(A)}{2}\right| = \max_{1\le k\le n}\left|1 - \frac{2(1-\cos(k\pi/(n+1))}{2}\right| \\ &= \max_{1\le k\le n}\left|\cos\frac{k\pi}{n+1}\right| = \cos\frac{\pi}{n+1} \sim 1 - \frac{1}{2}\left(\frac{\pi}{n+1}\right)^2. \end{aligned}$$

We can see that as $n$ is getting larger, the spectral radius is getting close to unit indicating slower convergence.

Once we know the spectral radius, we also know roughly the number of iterations needed to reach the desired accuracy. For example, if we wish to have roughly six significant digit, the we should set $\rho(R)^k \le 10^{-6}$ or $k \ge -6\log_{10}(\rho(R))$.

### 4.6.1 Finite difference method for the Poisson equation in two dimensions

In two space dimensions, we have parallel results. The eigenvalues for the matrix $A = h^2 A_{FD}$ of $N$ by $N$ matrix $N = n^2$ are

$$\lambda_{i,j} = -\left(4 - 2\left(\cos\frac{i\pi}{n+1} + \cos\frac{j\pi}{n+1}\right)\right), \quad i,j = 1,2,\cdots n. \tag{4.6.2}$$

The diagonals of the matrix are $-4$ and we have

$$\lambda_{i,j}(R_J) = 1 + \frac{\lambda_{i,j}(A)}{4}, \quad i,j = 1,2,\cdots,n.$$

Thus

$$\begin{aligned}\rho(R_J) &= \max_{1\le i,j\le n}\left|1 + \frac{\lambda_{i,j}(A)}{4}\right| = \max_{1\le i,j\le n}\left|1 - \frac{2(1-\cos(i\pi/(n+1)) + \cos(j\pi/(n+1))}{4}\right| \\ &= \max_{1\le i,j\le n}\left|\cos\frac{i\pi}{n+1} + \cos\frac{j\pi}{n+1}\right| = \cos\frac{\pi}{n+1} \sim 1 - \frac{1}{2}\left(\frac{\pi}{n+1}\right)^2.\end{aligned}$$

We see that the results in 1D and 2D are pretty much the same. To derive the best $\omega$ for the SOR($\omega$) method, we need to derive the eigenvalue relation between the original matrix and iteration matrix. Note that $R_{SOR} = (D - \omega L)^{-1}((1-\omega)D + \omega U)$.

**Theorem 4.6.2** *The optimal $\omega$ for the SOR($\omega$) method for the system of equations derived from the finite difference method for the Poisson equation is*

$$\omega_{opt} = \frac{2}{1 + \sqrt{1 - (\rho(R_j))^2}} = \frac{2}{1 + \sqrt{1 - \cos^2\frac{\pi}{n+1}}} = \frac{2}{1 + \sin\frac{\pi}{n+1}} \sim \frac{2}{1 + \frac{\pi}{n+1}} \tag{4.6.3}$$

Below is the sketch of the proof.

- Step 1. Show the eigenvalue relation between $R_J$ and $R_{SOR}(\omega)$

$$\lambda_{SOR}(\omega) = 1 - \omega + \frac{1}{2}\omega^2\lambda_J^2 \pm \omega\lambda_J\sqrt{1 - \omega + \frac{\omega^2\lambda_J^2}{4}} \tag{4.6.4}$$

- Step 2. Find the extreme value (minimum) of above as a function of $\lambda_J$ which leads to the optimal $\omega$.

We refer the readers to the book [J. W. Demmel] on page 292-293 for the proof.

**Remark 4.6.1**

- *The spectral radius of $R_{SOR}(\omega)$ is*

$$\rho(R_{SOR}) = \begin{cases} 1 - \omega + \frac{1}{2}\omega^2\rho(R_J)^2 + \omega\rho(R_J)\sqrt{1 - \omega + \frac{\omega^2\rho(R_J)^2}{4}}, & 0 < \omega < \omega_{opt}, \\ \omega - 1, & \omega_{opt} < \omega < 2. \end{cases}$$

  *If we plot $\rho(R_{SOR})$ against $\omega$, we can see it is a quadratic curve between $0 < \omega < \omega_{opt}$ and it is flat as $\omega$ is getting closer to $\omega_{opt}$ which means it is less sensitive in the neighborhood of $\omega_{opt}$; while the second piece is a linear function. Thus, we would rather choose $\omega$ larger than smaller.*

- *The optimal $\omega$ is only for the Poisson equation, not for other elliptic problems. However, it gives a good indication of the best $\omega$ is the diffusion is dominant in reference to the mesh size.*

# Chapter 5

# Computing algebraic eigenvalues and eigenvectors

## 5.1 Preliminary

Given a square matrix $A \in R^{n,n}$, if we can find a number $\lambda \in C$ and $\mathbf{x} \neq 0$ such that $A\mathbf{x} = \lambda\mathbf{x}$, then $\lambda$ is called an ***eigenvalue*** of $A$, $\mathbf{x}$ is called a corresponding ***eigenvector*** to $\lambda$. Note that if $A\mathbf{x} = \lambda\mathbf{x}$, then $A(cx\mathbf{x}) = \lambda(c\mathbf{x})$ for any non-zero cnstant $c$, in other words, eigenvector can differ by a constant. Often we prefer to use an eigenvector with unit length ( $\|\mathbf{x}\| = 1$). We call $(\lambda, \mathbf{x})$ an ***eigen-pair*** if $A\mathbf{x} = \lambda\mathbf{x}$ ($\mathbf{x} \neq 0$).

For an eigen-pair $(\lambda, \mathbf{x})$, we have $A\mathbf{x} - \lambda\mathbf{x} = \mathbf{0}$. This means that $(\lambda I - A)\mathbf{x} = \mathbf{0}$ has non-zero (or not unique) solutions. This indicates that $(\lambda I_A)$ is singular, or $det(\lambda I_A) = 0$. Thus $\lambda$ must be a root of the ***characteristic polynomial*** of the matrix $A$, $det(\lambda I - A) = \lambda^n + a_{n-1}\lambda^{n-1} + \cdots + a_1\lambda + a_0$.

There are $n$ eigenvalues for an $n$ by $n$ square matrix. The eigenvalues can be real, complex numbers, repeated roots. If the matrix is real, then the complex eigenvalues are in pairs, that is, if $\lambda = a + bi$ is one eigenvalue, then $\bar{\lambda} = a - bi$ is also an eigenvalue. If $A$ is a real and symmetric matrix, then all the eigenvalues are real numbers.

Different eigenvectors corresponding to different eigenvalues are linear independent. If an eigenvalue $\lambda^*$ has multiplicity $p$, which means that characteristic polynomial has the factor $(\lambda - \lambda^*)^p$, but no the factor $(\lambda - \lambda^*)^{p+1}$, the number is independent of eigenvectors corresponding to $\lambda^*$ is less than or equal to $p$, recall some examples in class. If an $n$ by $n$ square matrix $A$ has $n$ linear independent eigenvectors, then $A$ is diagonalizable, that is, there is a nonsingular matrix $S$, such that $S^{-1}AS = D$, where $D = diag(\lambda_1, \lambda_2, \cdots, \lambda_n)$ is a diagonal matrix.

For convenience of discussion, we will use the following notations. We arrange eigenval-

ues of a matrix $A$ according to

$$|\lambda_1| \geq |\lambda_2| \geq \cdots |\lambda_i| \geq \cdots \geq |\lambda_n|.$$

Thus $\rho(A) = |\lambda_1|$, $\lambda_1$ is called the ***dominant eigenvalue*** (can be more than one), while $\lambda_n$ is called the ***least dominant eigenvalue***.

There are many applications of eigenvalue problems. Below are a few of them

- Frequencies if vibration, resonates, etc.
- Spectral radius and convergence of iterative method
- Discrete form of continuous eigenvalue problems, for example, the Sturm-Louville problem

  $$-(p(x)u'(x))' + q(x)u(x) = \lambda u(x), \quad 0 < x < 1,$$
  $$u(0) = 0, \quad u(1) = 0.$$

  After applying a finite difference or finite element method, we would have $A\mathbf{x} = \lambda\mathbf{x}$. The solutions are the basis for the Fourier series expansion.
- Stability analysis of dynamic systems, or numerical methods.

## 5.2 The Power's method

The idea of the power method is: starting from a non-zero vector $\mathbf{x}^{(0)} \neq \mathbf{0}$ (an approximation to an eigenvector), then form an iteration

$$\mathbf{x}^{(k+1)} = A\mathbf{x}^{(k)} = A^2\mathbf{x}^{(k-1)} = \cdots A^{k+1}\mathbf{x}^{(0)}$$

Then under some conditions, we can extract an eigen-pair information from the sequence!

With the assumption that $\lambda_1$ satisfies $|\lambda_1| > |\lambda_2|$, which is an essential condition, then we can show that $\mathbf{x}^{(k)} \sim C\mathbf{x}_1$, and $x_p^{(k+1)}/x_p^{(k)} \sim \lambda_1$ as $k$ is very large, where $|x_p^{(k+1)}| = \|\mathbf{x}^{(k+1)}\|$.

**Sketch of the proof:**

While feasible in theory, the idea is not practical in computation because $\mathbf{x}^{(k)} \longrightarrow \mathbf{0}$ if $\rho(A) < 1$ and $\mathbf{x}^{(k)} \longrightarrow \infty$ if $\rho(A) > 1$. The solution is to rescale the vector sequence, which leads to the following power method.

Given $\mathbf{x}^{(0)} \neq \mathbf{0}$, form the following iteration

$$\text{for } k = 1 \text{ until converges}$$

$$\mathbf{y}^{(k+1)} = A\mathbf{x}^{(k)}$$

$$\mathbf{x}^{(k+1)} = \frac{\mathbf{y}^{(k+1)}}{\|\mathbf{y}^{(k+1)}\|_2}$$

$$\mu_{k+1} = (\mathbf{x}^{(k+1)})^T A\mathbf{x}^{(k+1)}$$

$$\text{end}$$

We can use the following stopping criteria: $|\mu_{k+1} - \mu_k| < tol$ or $\|\mathbf{y}^{(k+1)} - \mathbf{y}^{(k)}\| < tol$ or both.

Under some conditions, the sequence of the pair $(\mu_k, \mathbf{x}^{(k)})$ converge to the eigen-pair corresponding to the dominant eigenvalue.

For simplicity of the proof, we assume that $A$ has a complete eigenvectors $\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_n$, $\|\mathbf{v}_i\| = 1$, $A\mathbf{v}_i = \lambda_i \mathbf{v}_i$.

**Theorem 5.2.1** *Assume that* $|\lambda_1| > |\lambda_2| \geq |\lambda_3| >\geq \cdots \geq |\lambda_n|$, *and* $\boldsymbol{x}^{(0)} = \sum_{i=1}^n \alpha_i \boldsymbol{v}_i$ *with* $\alpha_1 \neq 0$, *then the pair* $(\mu_k, \boldsymbol{x}^{(k)})$ *the power method converges to the eigen-pair corresponding to the dominant eigenvalue.*

**Proof**: Note that

$$\begin{aligned} \mathbf{y}^{(k)} &= A\mathbf{x}^{(k-1)} = A\frac{\mathbf{y}^{(k-1)}}{\|\mathbf{y}^{(k-1)}\|_2} = \gamma_k A\mathbf{y}^{(k-1)} \\ &= \gamma_k\gamma_{k-1}A^2\mathbf{y}^{(k-2)} = \cdots \gamma_k\gamma_{k-1}\cdots\gamma_1 A^{k-1}\mathbf{y}^{(1)} = C_k A^k \mathbf{x}^{(0)} \end{aligned}$$

where $\gamma_k\gamma_{k-1}\cdots\gamma_1$ and $C_k$ are some constants. Since $\mathbf{x}^{(k)}$ is parallel to $\mathbf{y}^{(k)}$ and has unity length in 2-norm, we must have

$$\mathbf{x}^{(k)} = \frac{\mathbf{y}^{(k)}}{\|\mathbf{y}^{(k)}\|_2}.$$

On the other hand, we know we have

$$A^k\mathbf{x}^{(0)} = \lambda_1^k \left( \alpha_1 v_1 + \alpha_2 v_2 \left(\frac{\lambda_2}{\lambda_1}\right)^k + \cdots \alpha_n v_n \left(\frac{\lambda_n}{\lambda_1}\right)^k \right)$$

Thus we have

$$\lim_{k\to\infty} \mathbf{x}^{(k)} = \lim_{k\to\infty} \frac{\mathbf{x}^{(k)}}{\|\mathbf{x}^{(k)}\|_2} = \frac{\alpha_1\mathbf{v}_1}{\|\alpha_1\mathbf{v}_1\|_2} = \pm\mathbf{v}_1.$$

and

$$\lim_{k\to\infty} \mu_k = \lim_{k\to\infty} (\mathbf{x}^{(k)})^T A\mathbf{x}^{(k)} = \lambda_1.$$

### 5.2.1 The power method using the infinity norm

If we use different scaling, we can get different power method. Uisng the infinity norm, first we introduce the $x_p$ notation for a vector $\mathbf{x}$. Given a vector $\mathbf{x}$, $x_p$ is the first component such that $|x_p| = \|\mathbf{x}\|_\infty$, and $p$ is the index. For example, if $\mathbf{x} = [\ 2 \quad -1 \quad -5 \quad 5 \quad -5\ ]^T$, then $x_p = -5$ with $p = 3$.

## 5.3 The inverse power method for the least dominant eigenvalue

If $A$ is invertible, then $1/\lambda_i$ are the eigenvalues of $A^{-1}$ and $1/\lambda_n$ will be the dominant eigenvalue of $A^{-1}$. However, the following inverse power method for the least dominant eigenvalue without the need to have the intermediate steps.

Given $\mathbf{x}^{(0)} \neq \mathbf{0}$, form the following iteration

for $k = 1$ until converges

$$A\mathbf{y}^{(k+1)} = \mathbf{x}^{(k)}$$

$$\mathbf{x}^{(k+1)} = \frac{\mathbf{y}^{(k+1)}}{\|\mathbf{y}^{(k+1)}\|_2}$$

$$\mu_{k+1} = (\mathbf{x}^{(k+1)})^T A\mathbf{x}^{(k+1)}.$$

end

With similar conditions ($|\lambda_n| < |\lambda_{n-1}| \leq \cdots \leq |\lambda_1$, the essential condition), one can prove that

$$\lim_{k\to\infty} \mu_k = \lambda_n, \qquad \lim_{k\to\infty} \mathbf{x}^{(k)} = \mathbf{v}_n, \quad \text{with} \quad \|\mathbf{v}_n\|_2 = 1.$$

Note that, at each iteration, we need to solve a linear system of equations with the same coefficient matrix. This is the most expensive part of the algorithm. An efficient implementation is to get the matrix decomposition done outside the loop. Let $PA = LU$, the algorithm can be written as follows.

Given $\mathbf{x}^{(0)} \neq \mathbf{0}$, form the following iteration

$$\begin{aligned}
&\text{for } k = 1 \text{ until converges} \\
&\quad L\mathbf{z}^{(k+1)} = P\mathbf{x}^{(k)} \\
&\quad U\mathbf{y}^{(k+1)} = \mathbf{z}^{(k+1)} \\
&\quad \mathbf{x}^{(k+1)} = \frac{\mathbf{y}^{(k+1)}}{\|\mathbf{y}^{(k+1)}\|_2} \\
&\quad \mu_{k+1} = (\mathbf{x}^{(k+1)})^T A\mathbf{x}^{(k+1)} \\
&\text{end}
\end{aligned}$$

## 5.4 Gershogorin theorem and the shifted inverse power method

If we know a good approximate $\sigma$ to an eigenvalue $\lambda_p$ such that

$$|\lambda_p - \sigma| < \min_{1 \leq i \leq n, i \neq p} |\lambda_i - \sigma|.$$

That is, $\lambda_p - \sigma$ is the least dominant eigenvalue of $A - \sigma I$. We can use the following shifted inverse power method to find the eigenvalue $\lambda_p$ and its corresponding eigenvector Given $\mathbf{x}^{(0)} \neq \mathbf{0}$, form the following iteration

$$\begin{aligned}
&\text{for } k = 1 \text{ until converges} \\
&\quad (A - \sigma I)\mathbf{y}^{(k+1)} = \mathbf{x}^{(k)} \\
&\quad \mathbf{x}^{(k+1)} = \frac{\mathbf{y}^{(k+1)}}{\|\mathbf{y}^{(k+1)}\|_2} \\
&\quad \mu_{k+1} = (\mathbf{x}^{(k+1)})^T A\mathbf{x}^{(k+1)} \\
&\text{end}
\end{aligned}$$

Thus if we can find good approximations of any eigenvalue, we can use the shifted inverse power method to compute it. Now the question is how do we roughly located the eigenvalues of a matrix $A$. Gershgorin theorem provides some useful hints.

**Definition 5.4.1** *Given a matrix $A \in C^{n \times n}$, the circle ( all points within the circle on the complex plane)*

$$|\lambda - a_{ii}| \leq \sum_{j=1, j \neq i}^{n} |a_{ij}| \tag{5.4.1}$$

*is called the i-th Gershgorin circle.*

**Theorem 5.4.1** *Gershgorin Theorem.*

1. *Any eigenvalue have to be in one of Gershgorin circles.*
2. *The union of $k$ Gershgorin circles, which do not intersect with other $n-k$ circles, contains precisely $k$ eigenvalues of $A$.*

Proof: For any eigen-pair $(\lambda, \mathbf{x})$, $A\mathbf{x} = \lambda\mathbf{x}$. Consider the $p$-th component of $\mathbf{x}$ such that $|x_p| = \|\mathbf{x}\|_\infty$, we have

$$\sum_{j=1}^{n} a_{pj}x_j = \lambda_p x_p$$

or

$$(\lambda - a_{pp})x_p = \sum_{j=1, j\neq p}^{n} a_{pj}x_j.$$

From the expression above we get

$$|\lambda - a_{pp}| \leq \sum_{j=1, j\neq p}^{n} |a_{pj}| \left|\frac{x_j}{x_p}\right| \leq \sum_{j=1, j\neq p}^{n} |a_{pj}|, \qquad \text{since } |x_j/x_p| \leq 1.$$

Thus $\lambda$ is in the $p$-th Gershgorin circle and the first part of the theorem is complete.

The proof for the second part is based continuation theory that roots of a polynomial are continuous functions of the coefficients of the polynomials. The theorem is obviously true for the diagonal matrix. If the radius of the Gershgorin circles increase continuously as we change the zero off diagonal entries from 0 to $a_{ij}$, the eigenvalues will move among union of the Gershgorin circles but can not cross to the disjoint ones.

**Example:** Let $A$ be the following matrix

$$A = \begin{pmatrix} -5 & -1 & 0 \\ -1 & 2 & -1/2 \\ 0 & -1 & 8 \end{pmatrix}$$

Use the Gershgorin theorem to roughly locate the eigenvalues.

The three Gershgorin circles are

- $R_1$ : $\quad |z+5| \leq 1.$
- $R_2$ : $\quad |z-2| \leq 1.5.$
- $R_3$ : $\quad |z-8| \leq 1.$

The do not interset with each other, so each circle has one eigenvalue. Since the matrix is a real matrix, and complex eigenvalues have to be in pair, we conclude that all the eigenvalues are real. Thus we get

- the dominant eigenvalue satisfies $7 \leq \lambda_1 \leq 9$,
- the least dominant eigenvalue satisfies $0.5 \leq \lambda_3 \leq 3.5$,
- the middle eigenvalue satisfies $-6 \leq \lambda_1 \leq -4$.

If we wish to find the middle eigenvalue $\lambda_2$ we should choose $\sigma = -5$. Even for the dominant eigenvalue, we would get faster convergence if we shift the matrix by taking $\sigma = 8$ and then apply the shifted power method.

## 5.5 How to find a few or all eigenvalues?

If we know an eigenpair $(\lambda_1, \mathbf{x}_1)$ of a matrix $A$, we can use the ***deflation method*** to reduce $A$ to a one-dimensional lower matrix whose eigenvalues are the same as the rest eigenvalues of $A$. The process is follows. Assume that $\|\mathbf{x}_1\|_2 = 1$, we can form expand $\mathbf{x}$ to form an orthogonal basis of $R^n$: $\{\mathbf{x}_1, \mathbf{x}_2, \cdots \mathbf{x}_n\}$ with $\mathbf{x}_i^T \mathbf{x}_j = \delta_{ij}$. Note that $\delta_{ij} = 0$ if $i \neq j$ and $\delta_{ii} = 1$. Let $Q = [\mathbf{x}_1, \mathbf{x}_2, \cdots \mathbf{x}_n]$, then $Q$ is an orthogonal matrix ( $Q^T Q = QQ^T = I$). We can get

$$\begin{aligned} Q^T AQ &= Q^T [A\mathbf{x}_1, A\mathbf{x}_2, \cdots A\mathbf{x}_n] \\ &= Q^T [\lambda_1 \mathbf{x}_1, A\mathbf{x}_2, \cdots A\mathbf{x}_n] = \begin{pmatrix} \lambda_1 & * \\ 0 & A_1 \end{pmatrix} \end{aligned}$$

Thus the eigenvalues of $A_1$ are also those of $A$, but $A_1$ is a one-dimensional matrix compare with the original matrix $A$. The deflation method is only used if we wish to find a few eigenvalues.

To find all eigenvalues, the $QR$ method for eigenvalues are often used. The idea of the $QR$ method is first to reduce a matrix to a simple form (often upper Hessenberg matrix or tridiagonal matrix) using similarity transformation $S^{-1}AS$ so that the eigenvalues are unchanged. Since the inverses of an orthogonal matrix is its transpose, $S$ is often chosen as orthogonal matrix. Orthogonal matrices also have better stability than other matrixes since $\|Q\mathbf{x}\|_2 = \|\mathbf{x}\|_2$ and $\|QA\|_2 = \|A\|_2$.

**Definition 5.5.1** *Given a unit vector $\boldsymbol{w}$, $\|\boldsymbol{w}\|_2 = 1$, the Household matrix is defined as*

$$P = I - 2\boldsymbol{w}\boldsymbol{w}^T. \tag{5.5.1}$$

It is a simple check that $P = P^T = P^{-1}$. Such a matrix sometimes is also called a reflection matrix or transformation.

**Theorem 5.5.1** *If $\|\boldsymbol{x}\|_2 = \|\boldsymbol{y}\|_2$, then there is a Household matrix $P$ such that $P\boldsymbol{x} = \boldsymbol{y}$.*

Proof: Assume that $P\mathbf{x} = \mathbf{y}$, then we have

$$\left(I - 2\mathbf{w}\mathbf{w}^T\right)\mathbf{x} = \mathbf{y}$$

$$\mathbf{x} - \mathbf{y} = 2\mathbf{w}(\mathbf{w}^T\mathbf{x}).$$

Note that $2(\mathbf{w}^T\mathbf{x})$ is a number, thus $\mathbf{w}$ is parallel to $\mathbf{x} - \mathbf{y}$. Since $\mathbf{w}$ is also a unit vector in 2-norm, we conclude that $\mathbf{w} = (\mathbf{x} - \mathbf{y})/\|\mathbf{x} - \mathbf{y}\|_2$, then it is simple manipulation to show that $P\mathbf{x} = \mathbf{y}$.

**Example**: Find a Householder matrix $P$ such that

$$P\begin{pmatrix} 3 \\ 0 \\ 4 \end{pmatrix} = \begin{pmatrix} \alpha \\ 0 \\ 0 \end{pmatrix}$$

Note that in this example, we need to find both $\alpha$ and $P$. Since the orthogonal transformation does not change the 2-norm, we should have

$$\sqrt{3^2 + 4^2} = \alpha^2, \qquad \Longrightarrow \alpha = \pm 5,$$

$$\mathbf{w} = (\mathbf{x} - \mathbf{y})/\|\mathbf{x} - \mathbf{y}\|_2 = \frac{1}{\|\mathbf{x} - \mathbf{y}\|_2}\begin{pmatrix} 3 \pm \alpha \\ 0 - 0 \\ 4 - 0 \end{pmatrix}$$

To avoid possible cancellation, we should choose the opposite sign, that $\alpha = -5$, and $\mathbf{w} = [8 \;\; 0 \;\; 4]^T/\sqrt{80}$.

### 5.5.1 The $QR$ decomposition of a matrix $A$

Start from $A_0 = A$.

for $k = 0, 1, \cdots$ until converge

$A_k = Q_k R_K$

$A_{k+1} = R_k Q_k$.

end

**Theorem 5.5.2** *If $A \in R^{n\times n}$, and $|\lambda_1| \geq |\lambda_2| \geq \cdots \geq |\lambda_n|$, then*

- *$A_{i+1} \sim A_i \sim A$, that is, all $A_k$ have the same eigenvalues.*
- $$\lim_{k\to\infty} A_k = R_A = \begin{pmatrix} R_{11} & R_{12} & \cdots & R_{1p} \\ & R_{22} & \cdots & \cdots \\ & & \ddots & \vdots \\ & & & R_{pp} \end{pmatrix}$$

*where $R_A$ is a block upper triangular matrix whose diagonals $R_{ii}$ is either a 1 by 1 or 2 by 2 matrix (corresponding to complex eigenvalues in pair).*

**Proof** of the first part: $A_{k+1} = R_k Q_k = Q_k^T A_k R_k$.

**Shifted $QR$ method:**

Start from $A_0 = A$.

for $k = 0, 1, \cdots$ until converge

$A_k - \sigma_k I = Q_k R_K$

$A_{k+1} = R_k Q_k + \sigma_k I$.

end

Stop criteria: $\max_{3 \le i \le n, 1 \le j \le i-2} |a_{ij}| < tol$.

**Double shifted $QR$ method:** If $\sigma$ is chosen as a complex number, then we should use the Double shifted $QR$ method.

Start from $A_0 = A$.

for $k = 0, 1, \cdots$ until converge

$A_k - \sigma_k I = Q_k R_k$

$A_{k+1} = R_k Q_k + \sigma_k I$.

$A_{k+1} - \bar{\sigma}_k I = Q_{k+1} R_{k+1}$

$A_{k+2} = R_{k+1} Q_{k+1} + \bar{\sigma}_k I$.

end

$QR$ method for finding all eigenvalues are quite expensive. To reduce the computational cost. Often we use the similarity transformation via Householder matrix first to reduce the original matrix to an upper Hessenberg matrix (tri-diagonal if the matrix is symmetric) first, then apply the $QR$ method. This requires $n-2$ steps: $P_{n-1} P_{n-3} \cdots P_2 P_1 A P_1 P_2 \cdots P_{n-3} P_{n-2}$, where

$$P_1 = \begin{pmatrix} 1 & 0 \\ 0 & \bar{P}_1 \end{pmatrix}, \qquad \bar{P}_1 \begin{pmatrix} a_{21} \\ a_{31} \\ \vdots \\ a_{n1} \end{pmatrix} = \begin{pmatrix} \alpha \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

for example. The reason to choose $P_1$ to keep the first row of $A$ unchanged when we multiply $P_1$ from the left is to ensure that the first columns of $P_1 A$ unchanged when we multiply $P_1$ from the right so that those zeros will remain.

# Chapter 6

# Least squares and SVD solutions

In this chapter, we discuss numerical method for solving arbitrary linear system of equations.

## 6.1 Least squares solutions

Consider $A\mathbf{x} = \mathbf{b}$, where $A \in R^{m,\times n}$, $m \geq n$, and $rank(A) = n$. One motivation is the curve fitting example in which we have many observed data, but we find a simple polynomial to fit the data. Below are some examples:

$$\begin{pmatrix} 2 \\ 1 \\ 4 \\ -1 \end{pmatrix} \begin{pmatrix} x \end{pmatrix} = \begin{bmatrix} 0 \\ 0 \\ 4 \\ 2 \end{bmatrix}$$

From the first two equations, the solution should be $x = 0$; from the third equation, the solution should be $x = 1$; from the last equation, the solution should be $x = -2$; In other words, we can not find a single $x$ that satisfy all the equations. The system of equations is called ***over-determined***. In general, there is no classical solution to an over-determined system of equation. We need to find the 'best' solution.

By the best solution we mean that to minimize the error in some norm. Since the residual is computable, one approach is to minimize the residual in 2-norm of all possible choices:

The solution $\mathbf{x}^*$ is the 'best solution' in 2-norm if it satisfies

$$\|\mathbf{b} - A\mathbf{x}^*\|_2 = \min_{\mathbf{x} \in R^n} \|\mathbf{b} - A\mathbf{x}\|_2. \tag{6.1.1}$$

If we use different norms rather than 2-norm, it will leads to different algorithm and different applications. But 2-norm is the one that is used the most and it is the simplest one.

Note that, an equivalent definition is the following:

$$\|\mathbf{b} - A\mathbf{x}^*\|_2^2 = \min_{\mathbf{x}\in R^n} \|\mathbf{b} - A\mathbf{x}\|_2^2. \tag{6.1.2}$$

The above definition gives way to compute the 'best solution' as the global minimum of a multi-variable function of its component $x_1,\ x_2, \cdots, x_n$. This can be seen from the following:

$$\phi(\mathbf{x}) = \|\mathbf{b} - A\mathbf{x}\|_2^2 = (\mathbf{b} - A\mathbf{x})^T (\mathbf{b} - A\mathbf{x}) = \mathbf{b}^T\mathbf{b} - \mathbf{b}^T A\mathbf{x} - \mathbf{x}^T A^T\mathbf{b} + \mathbf{x}^T A^T A\mathbf{x}.$$

Note that $\mathbf{b}^T A\mathbf{x} = \mathbf{x}^T A^T\mathbf{b}$, one can easily get the gradient $\varphi(\mathbf{x})$ which is

$$\nabla\phi(\mathbf{x}) = 2\left(A^T A\mathbf{x} - A^T\mathbf{b}.\right)$$

Since the columns of $A$ are linearly independent ($rank(A) = n$), we have $\mathbf{x}^T A^T A\mathbf{x} = \|A\mathbf{x}\|^2 > 0$ for any non-zero $\mathbf{x}$, and $A^T A$ is symmetric positive definite. The only critical point of $\phi(\mathbf{x})$ is the solution of the following ***normal equation***

$$A^T A\mathbf{x} = A^T\mathbf{b} \tag{6.1.3}$$

whose unique solution is the least squares solution which minimizes the 2-norm of the residual in $R^n$ space.

The normal equation approach not only provides a numerical method, but also shows that the least squares solution is unique under the condition $rank(A) = n$, that is, the columns of $A$ are lienarly independent.

A serious problem with the normal equation approach is the possible ill-conditioned system. Note that if $m = n$, then $cond_2(A^T) = (cond_2(A))^2$. A more accurate method is the QR method for the least squares solution. For the following least squares problem

$$\begin{pmatrix} R \\ 0 \end{pmatrix}\mathbf{x} = \begin{pmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \end{pmatrix}$$

We have

$$\|\mathbf{b} - A\mathbf{x}\|_2^2 = \|\mathbf{b}_1 - R\mathbf{x}\|_2^2 + \|\mathbf{b}_2\|_2^2$$

and

$$\min_{\mathbf{x}\in R^n} \|\mathbf{b} - A\mathbf{x}\|_2^2 = \min_{\mathbf{x}\in R^n} \|\mathbf{b}_1 - R\mathbf{x}\|_2^2 + \|\mathbf{b}_2\|_2^2 = \|\mathbf{b}_2\|_2^2$$

when $\mathbf{b}_1 - R\mathbf{x} = 0$ or $\mathbf{x} = R^{-1}\mathbf{b}_1$. Particularly, if $R$ is an upper triangular matrix, we can use the backward substitution to solve the tri-diagonal system of equations efficiently.

In the QR method for the least squares, the idea is to reduce the original problem to the problem above using orthogonal, particularly, the Householder, transformation which

keeps the least squares solution unchanged. From the process of the QR algorithm, we can apply a sequence of Householder matrices that reduce $A$ to an upper triangular form

$$P_n P_{n-1} \cdots P_1 A = \begin{pmatrix} R \\ 0 \end{pmatrix}$$

or $QA = \begin{pmatrix} R^T & 0 \end{pmatrix}^T$, then from $A\mathbf{x} = \mathbf{b}$, we get $QA\mathbf{x} = Q\mathbf{b}$, or

$$\begin{pmatrix} R \\ 0 \end{pmatrix} \mathbf{x} = \begin{pmatrix} \tilde{\mathbf{b}}_1 \\ \tilde{\mathbf{b}}_2 \end{pmatrix}$$

In practice, we can apply the Householder matrix directly to the augmented matrix $[A \vdots b]$ as in the Gaussian elimination method.

**An example:**

## 6.2 Singular value decomposition (SVD), and SVD solution of $A\mathbf{x} = \mathbf{b}$

Singular value decomposition can be used to solve $A\mathbf{x} = \mathbf{b}$ for arbitrary $A$ and $\mathbf{b}$.

**Theorem 6.2.1** *Given any matrix $A \in C^{m\times n}$, there are two orthogonal matrices $U \in C^{m\times m}, UU^H = U^H U = I$ and $V \in C^{n\times n}, VV^H = V^H V = I$ such that*

$$A = U \sum V^H, \qquad \text{where} \qquad \sum = \begin{pmatrix} \sigma_1 & & & & \\ & \sigma_2 & & & \\ & & \ddots & & \\ & & & \sigma_p & \\ & & & & \boldsymbol{0} \end{pmatrix}_{m\times n} \tag{6.2.1}$$

*$\sigma_1, \sigma_2, \cdots, \sigma_p > 0$ are called the singular values of $A$. Note that they are positive numbers, $p = rank(A)$. Furthermore*

- *$\sigma_i = \sqrt{\lambda_i(A^H A)} = \sqrt{\lambda_i(AA^H)}$, square root of non-zero eigenvalues of $A^H A$ or $AA^H$.*
- *$\|A\|_2 = \max_{1\le i\le p} \sigma_i(A)$.*

Note that we often arrange singular values according to $\sigma_1 \ge \sigma_2 \ge \cdots \ge \sigma_p > 0$. The proof process also gives a way for such a decomposition (constrictive).

Proof: Let $\sigma_1 = \|A\|_2$, then there is an $\mathbf{x}$, $\|\mathbf{x}_1\|_2 = 1$ such that $A^H A\mathbf{x}_1 = \sigma_1^2 \mathbf{x}_1$. Let $\mathbf{y}_1 = A\mathbf{x}_1/\sigma_1$, we have $\|\mathbf{y}_1\|_2 = \|A\mathbf{x}_1\|_2/\sigma_1 = 1$.

Next we expand $\mathbf{x}_1$ to form an orthogonal basis in $R^{n\times n}$ to form an orthogonal matrix $V$

$$V = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n] = [\mathbf{x}_1,\ V_1]\,, \qquad V^H V = V^H V = I.$$

We also expand $\mathbf{y}_1$ to form an orthogonal basis in $R^{m\times m}$ to form an orthogonal matrix $U$

$$U = [\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_n] = \left[\ \mathbf{y}_1,\ \ U_1\ \right], \qquad U^H U = U^H U = I.$$

Then we have

$$U^H A V = \begin{pmatrix} \mathbf{y}_1^H \\ U_1^H \end{pmatrix} \begin{pmatrix} A\mathbf{x}_1 & AV_1 \end{pmatrix} = \begin{pmatrix} \sigma_1 & \mathbf{0} \\ \mathbf{0} & U_1^H A V_1 \end{pmatrix}$$

This is because $\mathbf{y}_1^H A\mathbf{x}_1 = \mathbf{x}_1^H A^H A\mathbf{x}_1/\sigma_1 = \mathbf{x}_1^H \mathbf{x}_1 \sigma_1^2/\sigma_1 = \sigma_1$; $U_1^H A\mathbf{x}_1 = \sigma_1 U_1^H \mathbf{y}_1 = \mathbf{0}$; $\mathbf{y}_1^H AV_1 = (A\mathbf{y}_1)^H V_1 = \mathbf{x}_1^H A^H AV_1 = \sigma_1^2 \mathbf{x}_1^H V_1 = \mathbf{0}$. Thus from the mathematics induction principle, we can continue this process to get the SVD decomposition.

## Pseudo-inverse of a matrix $A$

From the SVD decomposition of a matrix $A$, we can literally find the 'inverse' of the matrix to get its pseudo-inverse

$$A^+ = V\sum{}^+ U^H, \qquad \text{where} \qquad \sum{}^+ = \begin{pmatrix} \frac{1}{\sigma_1} & & & & \\ & \frac{1}{\sigma_2} & & & \\ & & \ddots & & \\ & & & \frac{1}{\sigma_p} & \\ & & & & \mathbf{0} \end{pmatrix}_{n\times m} \tag{6.2.2}$$

Particularly, if $m = n$ and $det(A) \neq 0$, then $A^+ = A^{-1}$. The pseudo-inverse matrix $A^+$ of a matrix $A$ has the following properties:

- $AA^+A = A$, note that $A^+A \neq I$.
- $A^+AA^+ = A^+$.
- $A^+A = (A + A)^H$. If $rank(A) = n$, then $A^+ = (A^H A)^{-1}A^H$.
- $AA^+ = (AA^+)^H$. If $rank(A) = m$, then $A^+ = A^H(AA^H)^{-1}$.

## Solving $A\mathbf{x} = \mathbf{b}$ of arbitrary matrix $A$

The SVD solution of $A\mathbf{x} = \mathbf{b}$ is simply $\mathbf{x}^* = A^+\mathbf{b}$. It has the the following properties:

- If $m = n$ and $det(A) \neq 0$, then $\mathbf{x}^* = A^+ b = A^{-1}\mathbf{b}$.
- If $rank(A) = n$, then $\|A\mathbf{x}^* - \mathbf{b}\|_2 = \min_{\mathbf{x}\in R^n} \|A\mathbf{x}^* - \mathbf{b}\|_2$, that is, $x^*$ is the least squares solution.
- If there is more than one classical solution, then $\mathbf{x}^*$ is the one with minimal 2-norm, that is

$$\|\mathbf{x}^*\|_2 = \min_{\|A\mathbf{x}^*-\mathbf{b}\|_2=\|A\mathbf{x}-\mathbf{b}\|_2} \{\|\mathbf{x}\|_2\}, \qquad \|A\mathbf{x}^* - \mathbf{b}\|_2 = \min_{\mathbf{x}\in R^n} \|A\mathbf{x} - \mathbf{b}\|_2$$