Numerical Linear Algebra

Dongwoo Sheen

Department of Mathematics &
Interdisciplinary Program in Computational Science & Technology
Seoul National University
Seoul 151-747, Korea

Email: sheen@snu.ac.kr
http://www.nasc.snu.ac.kr

March 4, 2015; 06:46

Contents

Chapter 1

Introduction

1.1 Difficulties in Computation in Numerical Linear Algebra Problems

Numerical methods for solving large-scale linear algebraic problems are necessary to solve various partial/ordinary differential equations by the finite element method, finite difference method, spectral method, and etc. Also in order to obtain numerical solutions to most optimization problems, such methods are essential.

Difficulties in practical computation of large linear systems arise from the following observations:

- usually the computational costs are too expensive;
- there will be possible loss in accuracy with a fixed number of digits computation;
- the methods are not applicable to different problems.

The main questions in numerical methods for linear systems are

- How fast is the numerical method in the sense of operation counts (flops: 1 flop=1 multiplication + 1 addition)?
- What is the accuracy? Can *a priori* and *a posteriori* estimates be given?
- What is the coverage of the method?

1.2 Examples

We begin by giving a short illustration in operation counting for the computation of $\det(A)$, for a given $n \times n$ matrix A. From the formula in elementary linear algebra,

$$\det(A) = \sum_{\sigma} \text{sign}(\sigma) a_{1\sigma_1} \cdots a_{n\sigma_n}, \tag{1.1}$$

where the summation is taken for all permutation $\sigma : \{1, \cdots, n\} \mapsto \{1, \cdots, n\}$ and the signature function is defined as usual:

$$\text{sign}(\sigma) = \begin{cases} 1, & \sigma \text{ is an even permutation,} \\ -1, & \sigma \text{ is an odd permutation.} \end{cases} \tag{1.2}$$

Moreover, the determinant of A can be computed using cofactor matrices in the following manner:

$$\det(A) = \sum_{k=1}^{n} (-1)^{1+k} a_{1k} \det(A_{1k}), \tag{1.3}$$

where A_{1k} is the submatrix of size $(n-1) \times (n-1)$ obtained by deleting the 1st row and kth column from A, which is identical to the $(k,1)$-cofactor matrix of A. The determinants of A_{1k}'s then can be computed by using the formula (1.3), recursively, until the size of cofactor matrices become 1×1. Indeed, denote by $f(n)$ the number of calculating $\det(A)$ of size $n \times n$ by using formula (1.3), if $\det(A_{1k}, k = 1, \cdots, n$, are known. Certainly $f(1) = 1$. Neglect the flops required for the multiplication by $(-1)^{1+k}$ since they are essentially absorbed in the operation $+$ or $-$. Then the operations for the calculation of the determinants can be described as follows:

1. the flops for the multiplication between a_{1k} and $\det(A_{1k})$: which is equal to n multiplications

2. the flops for the multiplication of $\det(A_{1k})$, which is of size $(n-1)\times(n-1)$: there are n such $\det(A_{1k}), k = 1, \cdots, n$.

Thus, recursively the total number of multiplications needed is as follows: for $n \geq 5$,

$$\begin{aligned} f(n) &= n + nf(n-1) \\ &= n + n[n-1 + (n-1)f(n-2)] \\ &= n + n[n-1 + (n-1)\{n-2 + (n-2)f(n-3)\}] \\ &= n + n[n-1 + (n-1)\{n-2 + (n-2)f(n-3)(\cdots(2+2f(1))\cdots)\}] \\ &= n + n(n-1) + n(n-1)(n-2) + \cdots + n(n-1)(n-2)\cdots 2 + n\,! \\ &= n + n(n-1) + n(n-1)(n-2) + \cdots + n(n-1)(n-2)\cdots 3 + 2(n\,!) \\ &> (2 + \frac{2}{3})(n\,!). \end{aligned}$$

owing to (1.4). Imagine how huge the number of operations are need by using this sort of elementary rule to calculate the determinant of a matrix of $1,000,000 \times 1,000,000$, where in actual computation such big a size of matrix will occur frequently.

Another example is to compute the inverse of a nonsingular matrix A. Cramer's rule implies

$$A^{-1} = \frac{1}{\det(A)} \left((-1)^{j+k} \det(A^{jk}) \right),$$

where A^{jk} denotes the (j,k)-cofactor matrix of A. If one uses the above idea to compute the determinants, the total flops will be $n\,! + n^2 \cdot (n-1)\,! = (n+1)\,!$, which will be impossible in practical computing.

Later in this chapter, we will see the flops will be substantially reduced by decomposing the matrix A into a product of lower and upper triangular matrices which results from a Gaussian elimination procedure.

Proposition 1.1. *Let $n \geq 5$. Let $g(n) = n + n(n-1) + n(n-1)(n-2) + \cdots + n(n-1)(n-2)\cdots 3$. Then the following holds:*

$$\frac{2}{3} n\,! < g(n) < \frac{23}{24} n\,! . \tag{1.4}$$

Proof. Notice that

$$g(n) = n\,! \left[\frac{1}{2\,!} + \frac{1}{3\,!} + \frac{1}{4\,!} + \cdots + \frac{1}{(n-2)\,!} + \frac{1}{(n-1)\,!} \right].$$

First, we have

$$\begin{aligned}&\frac{1}{2\,!}+\frac{1}{3\,!}+\frac{1}{4\,!}+\cdots+\frac{1}{(n-2)\,!}+\frac{1}{(n-1)\,!}\\&\quad>\frac{1}{2\,!}+\frac{1}{3\,!}=\frac{2}{3}.\end{aligned}$$

Next, we get

$$\begin{aligned}&\frac{1}{2\,!}+\frac{1}{3\,!}+\frac{1}{4\,!}+\cdots+\frac{1}{(n-2)\,!}+\frac{1}{(n-1)\,!}\\&\quad=\left(1-\frac{1}{2}\right)+\left(\frac{1}{2}-\frac{1}{3}\right)+\frac{1}{2\,!}\left(\frac{1}{3}-\frac{1}{4}\right)+\cdots\\&\qquad+\frac{1}{(n-4)\,!}\left(\frac{1}{n-2}-\frac{1}{n-3}\right)+\frac{1}{(n-3)\,!}\left(\frac{1}{n-1}-\frac{1}{n-2}\right)\\&\quad=\left(1-\frac{1}{2}\right)+\left(\frac{1}{2}-\frac{1}{3}\right)+\frac{1}{2\,!}\frac{1}{3}-\frac{1}{4}\left(\frac{1}{2\,!}-\frac{1}{3\,!}\right)-\frac{1}{5}\left(\frac{1}{3\,!}-\frac{1}{4\,!}\right)-\cdots\\&\qquad-\frac{1}{n-2}\left(\frac{1}{(n-3)\,!}-\frac{1}{(n-4)\,!}\right)+\frac{1}{(n-3)\,!}\frac{1}{(n-1)}\\&\quad<\left(1-\frac{1}{2}\right)+\left(\frac{1}{2}-\frac{1}{3}\right)+\frac{1}{2\,!}\frac{1}{3}+\frac{1}{(n-3)\,!}\frac{1}{(n-1)}\le\frac{23}{24},\end{aligned}$$

for $n \ge 5$. This proves the proposition. □

1.3 FORTRAN=FORmula TRANslator

FORTRAN was designed by John Backus (1924–2007) in 1954, for mathematicians and scientists, and remains the preeminent programming language in these areas today. FORTRAN first appeared in the IBM Preliminary Report,"Specifications for the IBM Mathematical FORmula TRANslating System, FORTRAN" [1], and then implemented on IBM 704.

FORTRAN is the first high-level programming language that translates mathematical formulae or logical statements into binary machine codes which can be executed thorough central processor units (CPU) using memory. High-level languages allow scientists and engineers to work with their computers although they do not have to understand how the machines actually work. If one would control computers at the machine level, learning the machines assembly language is necessary.

1.3.1 First Fortran program

The first test program in Fortran is to sum $\frac{1}{j*(j+1)}$ from $j = 1$ to $j = 999$. Save the following file with any file name with ending .f90, for instance, "summation_implicit.f90".

F90 Program 1.1: Sum $\frac{1}{j*(j+1)}$ from $j = 1$ to $j = 999$.

```
1 program summation
2   sum = 0.
3   do j = 1, 999
4      sum = sum + 1./real(j*(j+1))
5   end do
6   print*, " sum = ", sum
7 end program summation
```

Then at the command prompt, type

```
$ f90 summation_implicit.f90
```

Here, the command "f90" is the Fortran compiler. Depending on your machine, the command for Fortran compiler may differ. For instance, if you use intel fortran compiler, it may be "ifc", instead, while if you use gfortran compiler, it is "gfortran".

If the compilation process is successful, you should find a new file, "a.out", which is the executable binary file. Then, at the command prompt, type "a.out" as follows.

```
$ a.out
```

You will see the following output.

```
 sum = 0.999000251
```

Fortran was introduced in the 1950's, and until it was possible to type on the console terminal in the late 1970's, informing the Fortran code to computers was only possible by means of using some other methods, for instance, by punching the instruction information of the code to cards and the leading the cards to the reader. Hence it was at that time convention to use implicit agreement that the variables beginning with "j,k,l,m,n" are integers, and those beginning with all the other characters are real variables. Hence, the variables "j" and "sum" are implicitly understood as an integer and real variables. In some cases, when one would like to circumvent the implicit

variable convention, the program begins with "implicit none" and declare "variables" explicitly in the following lines. In many cases, circumventing the implicit variable convention are safer. In this case, one may modify the code as follows:

F90 Program 1.2: Sum $\frac{1}{j*(j+1)}$ from $j = 1$ to $j = 999$.

```
1  program summation
2    implicit none
3    real:: sum
4    integer:: j
5
6    sum = 0.
7    do j = 1, 999
8       sum = sum + 1./real(j*(j+1))
9    end do
10   print*, " sum = ", sum
11 end program summation
```

Save the above file with the file name "summation_single.f90", and type

```
$ f90 summation_single.f90
```

One will see the exact identical output as the previous one.

```
sum =   0.999000251
```

Observe that the program computes

$$\sum_{j=1}^{999} \frac{1}{j*(j+1)},$$

which is

$$\sum_{j=1}^{999} \left(\frac{1}{j} - \frac{1}{j+1} \right) = 1 - \frac{1}{100} = 0.999.$$

The exact summation is $1-0.001 = 0.999$. Well, we see the difference between the exact summation and the numerical solution is 0.000000251. This error is generated by calculating the summation using the single precision for the representation of real number by using four bytes. Here, one byte means eight bits. Next, let us try to calculate the sum in double precision, which uses eight bytes instead. Save the following filw with name "summation_double.f90" and compile it and run the executible file "a.out".

F90 Program 1.3: Sum $\frac{1}{j*(j+1)}$ from $j = 1$ to $j = 999$.

```
program summation
  implicit none
  real(8):: sum
  integer:: j

  sum = 0.d0
  do j = 1, 999
     sum = sum + 1.d0/real(j*(j+1))
  end do
  print*, " sum = ", sum
end program summation
```

Then, we will get the following output.

```
sum =   0.99900000000000067
```

Well, this time we have the summation correct up to fifteen digits after the decimal point.

Indeed, up to the 1970's people often computed using single precision since the memory was expensive at that time. But these days memories are cheap enough, and we recommend that most computation be done by using double precision. In our lecture, we will use double precision only.

In Fortran 90 or after the KIND types are available. Try to save the following file with "prec.f90" in the same directory and try to use your own precision and write fortran program flexible to changing the precision globally with minimal chages.

F90 Program 1.4: Precision declaration module

```
! D. Sheen (http://www.nasc.snu.ac.kr) as of Feb 1, 2015
module prec
  integer, parameter::hpi = 1   !-128 to 127
  integer, parameter::spi = 2   !-32,768 to 32,767
  integer, parameter::dpi = 4   !-2,147,483,648 to 2,147,483,647 (default)
  integer, parameter::qpi = 8   !-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
  integer, parameter::mpi = 4   !-2,147,483,648 to 2,147,483,647 (default)

  integer, parameter::sp=selected_real_kind(p=6,r=37) !standard   = 4
  integer, parameter::dp=selected_real_kind(p=15,r=307) !standard = 8
  integer, parameter::qp=selected_real_kind(p=33,r=4931) !standard=16
  integer, parameter::mp=selected_real_kind(p=15,r=307) !standard = 8

  integer, parameter::spc = kind((1.0, 1.0))
  integer, parameter::dpc = kind((1._dp, 1._dp))
  integer, parameter::qpc = kind((1._qp, 1._qp))
  integer, parameter::mpc = kind((1._mp, 1._mp))

  integer, parameter::lgt = kind(.true.)

end module prec
```

For this purpose, write F90/95-version programs as follows: save the file with name "summation.f90".

F90 Program 1.5: Sum $\frac{1}{j*(j+1)}$ from $j = 1$ to $j = 999$.

```
1  include "prec.f90"
2
3  program summation
4    use prec
5    implicit none
6    real(mp):: sum
7    integer(mpi):: j
8
9    sum = 0._mp
10   do j = 1, 999
11      sum = sum + 1._mp/real(j*(j+1),mp)
12   end do
13   print*, " sum = ", sum
14 end program summation
```

Then, we will get the following output.

```
sum =    0.99900000000000067
```

			123455678901234567890
Kind	Memory	Precision	Smallest value, tiny() Largest value, huge()
real (kind=4)	4 bytes	7 s.f. (Single)	1.1754944E-38 3.4028235E+38
real (kind=8)	8 bytes	15 s.f. (Double)	2.2250738585072014E-308 1.7976931348623157E+308
real (kind=16)	16 bytes	34 s.f. (Quad)	

Here, s.f. means "significant figures".

Kind	Memory	Range
integer (kind=1)	1 byte	-128 to 127
integer (kind=2)	2 bytes	-32768 to 32767
integer (kind=4)	4 bytes	-2147483648 to 2147483647
integer (kind=8)	8 bytes	about $\pm 9.2x10^{18}$

The default kind for integer uses 4 bytes, and hence integer (kind=4).
To assign double and quad precision, one can use the _8 and _16 as follows:

single precision	A = 1.2345678E38 or 1.2345678E38_4
double precision	A = 1.234567890123456E308_8
quad precision	A = 1.2345678901234567890123456789012345E4931_16

Also, for various real types, one can declare as follows:

```
          real ::   a   ! default (single precision)
 real(kind=4) ::   b   ! single precision
 real(kind=8) ::   c   ! double precision
real(kind=16) ::   d   ! quad precision
```

Simply, one can use the following tags for precision

```
A=1.2345678E38                                          ! (Single)
A=1.2345678E38_4                                        ! (Single)
A=1.234567890123456E308_8                               ! (Double)
A=1.23456789012345678901234567890123456E4931_16         ! (Quad)
```

In fortran programs, one may use format statements to read and write in convenient format. The following are frequently used format for read and write statements. Here,

Descriptor	Use
rIw	Integer data
rFw.d	Real data in decimal notation
rEw.d	Real data in scientific notation
Aw	Character data
'x...x'	Character strings
nX	Horizontal spacing (skip spaces)
/	Vertical spacing (skip lines)
Tc	Tab

w	=	positive integer specifying FIELD WIDTH
r	=	positive integer REPEAT COUNT
d	=	non-negative integer specifying number of digits to display to right of deci
x	=	any character
n	=	positive integer specifying number of columns
c	=	positive integer representing column number

1.3.2 The 10 algorithms with the greatest influence on the development and practice of science and engineering in the 20th century

Following is the list (here, the list is in chronological order; however, the articles appear in no particular order):

- Metropolis Algorithm for Monte Carlo
- Simplex Method for Linear Programming
- Krylov Subspace Iteration Methods
- The Decompositional Approach to Matrix Computations
- The Fortran Optimizing Compiler
- QR Algorithm for Computing Eigenvalues
- Quicksort Algorithm for Sorting
- Fast Fourier Transform
- Integer Relation Detection
- Fast Multipole Method

Chapter 2

The Gaussian Elimination

It has been long known that the Gaussian elimination scheme is not invented by Carl F. Gauss. He instead used and reformulated the method. Perhaps Issac Newton contributed much for the development of the method.

2.1 Gaussian elimination - an example

In this chapter we consider the linear system

$$Ax = b, \tag{2.1}$$

where A and b are given an $n \times n$ matrix and a vector in $\mathbb{C}^n$. As a simple example, we consider the following matrix A and b:

$$A = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}, \quad b = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}. \tag{2.2}$$

Let $A^{(1)}$ denote the augmented matrix by gluing the vector b as the fourth column to the matrix A, so that

$$A^{(1)} = \begin{bmatrix} 2 & -1 & 0 & 1 \\ -1 & 2 & -1 & 2 \\ 0 & -1 & 2 & 1 \end{bmatrix}. \tag{2.3}$$

The Gaussian elimination procedure is then to reduce the matrix $A^{(1)}$ to an upper triangular matrix by elementary row operations, which can be interpreted as follows.

Let $G_1 = I - m_1\, e_1^t$, with

$$m_1 = (0, m_{21}, m_{31})^t = \frac{1}{a_{11}^{(1)}}(0, a_{21}^{(1)}, a_{31}^{(1)})^t = (0, -\frac{1}{2}, 0)^t.$$

Then the first step of Gaussian elimination can be understood as multiplying the matrices G_1 and $A^{(1)}$ to obtain $A^{(2)}$ as follows:

$$G_1 A^{(1)} = \begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{2} & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & -1 & 0 & 1 \\ -1 & 2 & -1 & 2 \\ 0 & -1 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 2 & -1 & 0 & 1 \\ 0 & \frac{3}{2} & -1 & \frac{5}{2} \\ 0 & -1 & 2 & 1 \end{bmatrix} := A^{(2)}. \quad (2.4)$$

Next, let $G_2 = I - m_2\, e_2^t$, with

$$m_2 = (0, 0, m_{32})^t = \frac{1}{a_{22}^{(2)}}(0, 0, a_{32}^{(2)})^t = (0, 0, -\frac{2}{3})^t.$$

Then the second step of Gaussian elimination can be understood as multiplying the matrices G_2 and $A^{(2)}$ to obtain $A^{(3)}$ as follows:

$$G_2 A^{(2)} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & \frac{2}{3} & 1 \end{bmatrix} \begin{bmatrix} 2 & -1 & 0 & 1 \\ 0 & \frac{3}{2} & -1 & \frac{5}{2} \\ 0 & -1 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 2 & -1 & 0 & 1 \\ 0 & \frac{3}{2} & -1 & \frac{5}{2} \\ 0 & 0 & \frac{4}{3} & \frac{8}{3} \end{bmatrix} := A^{(3)}. \quad (2.5)$$

Definition 2.1. *A matrix A is called a lower (respectively, upper) triangular if $a_{jk} = 0$ for all $j < k$ (respectively, $j > k$). Moreover, if all the diagonal elements of such matrices are equal to 1, such matrices are said to be unit lower, or upper triangular.*

$$\mathbf{L} = \begin{pmatrix} 1 & & & 0 \\ & 1 & & \\ & & \ddots & \\ * & & & 1 \end{pmatrix} \text{ and } \mathbf{U} = \begin{pmatrix} u_{11} & & & * \\ & u_{22} & & \\ & & \ddots & \\ 0 & & & u_{nn} \end{pmatrix}$$

Proposition 2.1. *If A and B are (unit) lower triangular, then so is AB.*

Proof. Exercise □

Exercise 2.1. *Prove Proposition 2.1.*

Definition 2.2. *$G_j = I - g e_j^t$ is called a Gaussian transformation matrix with Gauss vector $g = (0, 0, \cdots, g_{j+1}, \cdots, g_n)^t$, where e_j is the jth standard unit vector.*

$$\mathbf{G_j} = \begin{pmatrix} 1 & & & & & \\ & \ddots & & & & \\ & & 1 & & & \\ & & -g_{j+1} & 1 & & \\ & & \vdots & & \ddots & \\ & & -g_n & & & 1 \end{pmatrix}.$$

Note that the matrix $A^{(3)}$ is upper triangular.

Summarizing the above procedure, one can write in the form

$$A^{(3)} = G_2 A^{(2)} = G_2 G_1 A^{(1)}, \tag{2.6}$$

from which one has the following decomposition of $A^{(1)}$ as a multiplication of a lower triangular matrix L and a unit upper triangular matrix U:

$$A^{(1)} = LU = \left[G_1^{-1} G_2^{-1}\right] A^{(3)}, \tag{2.7}$$

where G_j^{-1} is the inverse of G_j.

Exercise 2.2. *It is immediate to see that*

$$G_j^{-1} = I + m_j\, e_j^t, \quad \textit{for } j = 1, 2. \tag{2.8}$$

Exercise 2.3. *Check that $L = G_1^{-1} G_2^{-1}$ is a lower triangular matrix with the diagonal entry being 1. Moreover, show that*

$$L = \begin{bmatrix} 1 & 0 & 0 \\ m_{21} & 1 & 0 \\ m_{31} & m_{32} & 1 \end{bmatrix}.$$

Exercise 2.4. *Verify that the above procedure together with Exercises 2.2 and 2.3 is valid for any $n \times n$ matrices, as long as*

$$a_{jj}^{(j)} \neq 0 \textit{ for } j = 1, \cdots, n-1. \tag{2.9}$$

Exercise 2.5. *Let L and U be (unit) lower and upper triangular matrices.*

- *Show that L^{-1} and U^{-1} are also lower and upper triangular matrices. if $l_{jj} \neq 0, u_{jj} \neq 0\ \forall j$*
- *How many flops are needed to calculate L^{-1} and U^{-1}?*

- *If $A = LU$, then $A^{-1} = U^{-1}L^{-1}$ Assuming that L^{-1} and U^{-1} are already computed (as above). How many flops will be required to compute $U^{-1}L^{-1}$?*

From (2.5) it follows that

$$\begin{array}{rcrcrcl} 2x_1 & - & x_2 & & & = & 1, \\ & & \frac{3}{2}x_2 & - & x_3 & = & \frac{5}{2}, \\ & & & & \frac{4}{3}x_3 & = & \frac{8}{3}. \end{array} \tag{2.10}$$

Thus the solution is easily obtained by looking at the equations (2.10) in backward order:

$$x_3 = 2, \quad x_2 = 3, \quad x_1 = 2.$$

2.1.1 Gaussian elimination algorithm and a Fortran example

The Gaussian algorithm (2.6), even for general n, is essentially given as in the form:

ALGORITHM 2.1. *Gaussian elimination (without partial pivoting)*

```
1 do j=1,n-1
2   do k=j+1, n
3     m = a(k,j)/a(j,j)
4     a(k,j:n) = a(k,j:n) - m*a(j,j:n)
5     b(k) = b(k) - m*b(j)
6   enddo
7 enddo
```

In order to apply the Gaussian algorithm to a concrete example, let us consider the following one-dimensional elliptic problem: find $u(x)$ such that

$$-u''(x) = f(x), \quad x \in (0,1); \qquad u(0) = u(1) = 0. \tag{2.11}$$

Let $h = \frac{1}{N}$, and $x_j = jh, j = 0, \cdots, N$. The numerical approximation by a standard finite difference scheme is given by finding $U(x_j), j = 1, \cdots, N-1$, such that

$$-U(x_{j-1}) + 2U(x_j) - U(x_{j+1}) = h^2 f(x_j), j = 1, \cdots, N-1, \tag{2.12}$$

with $U(x_0) = U(x_N) = 0$. Denote by A the $(N-1)\times(N-1)$ tridiagonal matrix with $a_{jj} = 2$, $j = 1, \cdots N-1$, and $a_{j,j+1} = a_{j+1,j} = -1$, $j = 1, \cdots N-2$. Let $\mathbf{u}$ and $\mathbf{b}$ be the $N-1$ dimensional vectors $(U(x_1), U(x_2), \cdots, U(x_{N-1}))^t$ and $h^2\,(f(x_1), f(x_2), \cdots, f(x_{N-1}))^t$. Then we are led to solve

$$A\mathbf{u} = \mathbf{b}.$$

In our example, let us choose in particular $\mathbf{b} = e_1$, the first standard unit vector. Let $n = 100$.

A complete fortran program is as follows:

F90 Program 2.1: Gaussian elimination without pivoting

```
! Gaussian elimination without partial pivot
! Written by D. Sheen (http://www.nasc.snu.ac.kr) 2003. 3.
include "prec.f90"

program gausselim
  use prec
  implicit none

  integer(mpi):: n, j, k
  real(mp), allocatable :: a(:,:), b(:), savea(:,:), saveb(:)
  real(mp) :: m

  write(6,*) "What is the dimension n?"
  read(5,*) n ! n =10, 100, 1000
  n = n-1 ! Guess why do we put this line!
  allocate(a(1:n, 1:n), b(1:n), saveb(1:n), savea(1:n,1:n) )

  a = 0._mp
  do j = 1, n
     a(j,j) = 2._mp
  enddo
  do j = 1, n-1
     a(j,j+1) = -1._mp
     a(j+1,j) = -1._mp
  enddo

  b = 0._mp
  b(1) = 1._mp

  savea = a  ! Save the matrix a to test if the soler is correct
  saveb = b  ! Save the vector b to test if the soler is correct

  do j =1, n-1
     do k = j+1, n
        m = a(k,j)/ a(j,j)
        a(k,j:n) = a(k,j:n) - m*a(j,j:n)
        b(k) = b(k) - m*b(j)
     end do
  end do

```

```
41 ! Now solve U x = b by back substitution
42
43   do j = n, 1, -1
44      b(j) = ( b(j) - dot_product( a(j, j+1:n), b(j+1: n) ) ) / a(j,j)
45   end do
46
47 ! Now check if the solver is correct
48
49   do j = 1, n
50      if ( abs(saveb(j) - dot_product(savea(j,1:n), b(1:n))) > 5.e-13_mp ) then
51         print*, "The computed solution seems to be wrong at", j
52         stop "The solution is wrong"
53      end if
54   end do
55
56   write(6,*) "The solution x is as follows:"
57   write(6,91) b
58   write(66,91) b
59 91 format(5g12.3)
60
61 end program gausselim
```

Save F90 Prog. 2.1 with the file name "gausselim.f90" and compile with the command

```
$ f90 gausselim.f90
```

followed by ENTER Key. Then, you will have the executable program a.out, which can be run by typing in the command line

```
$ a.out
```

followed by ENTER Key. Then, you will see the message

```
What is the dimension n?
```

Then you can type in

```
10
```

followed by ENTER Key. You will see the results of the program.

```
 The solution x is as follows:
    0.90000         0.80000         0.70000         0.60000         0.50000
    0.40000         0.30000         0.20000         0.10000
```

Instead of writing all the procedure in one file like the above F90 Prog. 2.1, professional programmers usually use subprograms that consist of main program, subroutine and function subprograms. Many basic subroutines and

functions are often reused repeatedly, and are supposed to be common to other calling programs. The following program is a compact one which uses a kind of standard subprograms.

F90 Program 2.2: Main program for Gaussian elimination without pivoting

```
! Gaussian elimination without partial pivot
! Written by D. Sheen (http://www.nasc.snu.ac.kr) 2003. 3.
include "prec.f90"

program gausselim_main
  use prec
  implicit none

  integer(mpi):: n, j, k
  real(mp), allocatable :: a(:,:), b(:), savea(:,:), saveb(:)

  write(6,*) "What is the dimension n?"
  read(5,*) n ! n =10, 100, 1000
  n = n-1 ! Guess why do I put this line!
  allocate(a(1:n, 1:n), b(1:n), saveb(1:n), savea(1:n,1:n) )

  call build_a(a, n)
  call build_b(b, n)

  savea = a  ! Save the matrix a to test if the soler is correct
  saveb = b  ! Save the vector b to test if the soler is correct

  do j =1, n-1
     call gauss_tran(a(j:n,j:n),b(j:n),n,j)
  end do

  call back_subs(a,b,n) ! Solve U x = b by back substitution
  call check_axb(savea, b, saveb, n) ! Check if the solver is correct

  write(6,*) "The solution x is as follows:"
  write(6,91) b
  write(66,91) b
91 format(5g12.3)

end program gausselim_main
```

F90 Program 2.3: Build the matrix A for Gaussian elimination without pivoting

```
subroutine build_a(a, n)
  use prec
  implicit none
  integer(mpi):: n, j
  real(mp):: a(1:n,1:n)

  a = 0._mp
  do j = 1, n
     a(j,j) = 2._mp
  enddo
  do j = 1, n-1
```

```
12        a(j,j+1) = -1._mp
13        a(j+1,j) = -1._mp
14      enddo
15    end subroutine build_a
```

F90 Program 2.4: Build the right side b for Gaussian elimination without pivoting

```
1   subroutine build_b(b, n)
2     use prec
3     implicit none
4     integer(mpi):: n
5     real(mp):: b(1:n)
6
7     b = 0._mp
8     b(1) = 1._mp
9
10  end subroutine build_b
```

F90 Program 2.5: Gauss transformation

```
1   subroutine gauss_tran(a,b,n,j)
2     use prec
3     implicit none
4     integer(mpi)::n,j,k
5     real(mp):: m
6     real(mp), intent(inout):: a(j:n,j:n), b(j:n)
7
8     do k = j+1, n
9        m = a(k,j)/a(j,j);  a(k,j) = m  !overwriting lower part of A with L
10       a(k,j+1:n) = a(k,j+1:n) - m*a(j,j+1:n)
11       b(k) = b(k) - m*b(j)
12    end do
13  end subroutine gauss_tran
```

F90 Program 2.6: Back Substitution

```
1   subroutine back_subs(a,b,n)
2     use prec
3     implicit none
4     integer(mpi):: n,j
5     real(mp), intent(inout):: a(1:n,1:n), b(1:n)
6
7     do j = n, 1, -1
8        b(j) = ( b(j) - dot_product( a(j, j+1:n), b(j+1: n) ) ) / a(j,j)
9     end do
10
11  end subroutine back_subs
```

F90 Program 2.7: Check Ax=b

```
1   subroutine check_axb(savea, x, saveb, n)
2     use prec
3     implicit none
4     integer(mpi):: n, j
```

```
5    real(mp):: savea(1:n,1:n), x(1:n), saveb(1:n)
6    do j = 1, n
7       if ( abs(saveb(j) - dot_product(savea(j,1:n), x(1:n))) > 5.e-13_mp ) then
8          print*, "The computed solution seems to be wrong at", j
9          stop "The solution is wrong"
10      end if
11   end do
12 end subroutine check_axb
```

Save the programs F90 Prog. 2.2, F90 Prog. 2.3, F90 Prog. 2.4, F90 Prog. 2.5, F90 Prog. 2.6, F90 Prog. 2.7 with the file names gausselim_main.f90, build_a.f90, build_b.f90, gauss_tran.f90, back_subs.f90, and check_axb.f90.

Then in oder to compile, type in the command line the following:

```
$ f90 gausselim_main.f90 build_a.f90 build_b.f90 gauss_tran.f90 \
    back_subs.f90 check_axb.f90
```

followed by ENTER Key.

Exercise 2.6. *Veryfy by a program that confirms the $A = LU$ as long as* (2.9) *holds. More precicely, write a program for general $n \times n$ matrix A to find L and U, from which you can check $A = LU$. In actual checking, use $n = 3$ and A given as in* (2.2).

Exercise 2.7. *In the above program, observe an interesting property from the solutions when you have inputs with $N = 10, 20, 100$. See the solutions are given by $U(x_j) = 1 - jh$, for $j = 1, \cdots, N-1$, where $h = \frac{1}{N}$. Explain why this happens. Try to do with your best imagination!*

2.2 LU-decomposition with partial pivoting

2.2.1 Examples needed partial pivoting

Why do we need pivoting? (This part will be rewritten.)

Example 2.1. *$\beta = 10, t = 2$ (decimal, rounding after 3^{rd} decimal point.)*

$$\begin{pmatrix} 0.1 & 0.94 \\ 10. & 0.1 \end{pmatrix} X = \begin{pmatrix} 1.0 \\ 10. \end{pmatrix}, X = \begin{pmatrix} 0.99041534 \\ 0.958466 \end{pmatrix}$$

The computation by Gauss elimination (without pivoting) yields

$$\begin{pmatrix} 0.1 & 0.94 & 1.0 \\ 0. & 0.1 - 94. & -90. \end{pmatrix} \rightarrow \begin{pmatrix} 0.1 & 0.94 & 1.0 \\ 0. & -94. & -90. \end{pmatrix},$$

from which

$$X = \begin{pmatrix} 1.4 \\ 0.95 \end{pmatrix}$$

Instead, the computation by Gauss elimination with partial pivoting yields

$$\begin{pmatrix} 10 & 0.1 & 10. \\ 0. & 0.94 & 1.0 - 0.01 \end{pmatrix} \rightarrow \begin{pmatrix} 10 & 0.1 & 10. \\ 0. & 0.94 & 0.9 \end{pmatrix}$$

from which

$$X = \begin{pmatrix} 1.0 \\ 0.96 \end{pmatrix}$$

Example 2.2. *Another example that requires partial pivoting is as follows:*

$$\begin{pmatrix} 0. & 1.00 \\ 2.00 & 0. \end{pmatrix} \tag{2.13}$$

2.2.2 Gaussian elimination with partial pivoting

In the previous subsection the Gaussian elimination procedure is interpreted as multiplying by the Gauss transformation matrix G_j to the left of $A^{(j)}$ at j-th step for $j = 1, \cdots, n-1$ so that the resulting upper triangular matrix $U = G_{n-1} \cdots G_2 G_1 A$ is obtained. However, Examples 2.1 and 2.2 show that one needs to swap the j-th with j_s-th row where $|a_{j_s j}| \geq |a_{kj}|$ for $k \geq j$. This step is called *partial pivoting* and amounts to multiply to the left of $A^{(j)}$ by the permutation matrix P_j given by

$$\mathbf{P_j} = \begin{pmatrix} 1 & & & & & \\ & \ddots & & & & \\ & & 1 & & & \\ & & & 1 & & \\ & & \vdots & & \ddots & \\ & & & & & 1 \end{pmatrix}.$$

The j-th step is then complete by multiplying G_j to the resulting matrix $P_j A^{(j)}$. Therefore the whole procedure of Gaussian elimination with partial pivoting corresponds to obtaining the following upper triangular matrix

$$G_{n-1} P_{n-1} G_{n-2} P_{n-2} \cdots G_1 P_1 A = U,$$

which is stated as in the following algorithm in which the j-th column below the diagonal at j-th step is filled with the multiplier vectors used for G_k.

Let $a(1:m, 1:n)$ denote an $m \times n$ matrix. Then $a(j:m, j)$ will denote the $m-j+1$ dimensional vector part of jth column $a(1:m, j)$. Also $a(j, :)$ will denote $a(j, 1:n)$ and $a(j_1:j_2, :)$ will denote $a(j_1:j_2, 1:n)$ and so on. Denote

$$P = P_{n-1}P_{n-2}\cdots P_1 \qquad \text{and } L = P\left[G_{n-1}P_{n-1}G_{n-2}P_{n-2}\cdots G_1P_1\right]^{-1}. \tag{2.14}$$

Lemma 2.1. *The matrix L defined by (2.14) is a unit lower triangular matrix.*

Proof. Exercise. □

Exercise 2.8. *Prove Lemma 2.1.*

We thus have the LU-decomposition of A in the form

$$PA = LU, \tag{2.15}$$

where

$$L = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ l_{21} & 1 & 0 & \cdots & 0 \\ l_{31} & l_{32} & 1 & \cdots & 0 \\ \cdot & \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdot & \cdots & \cdot \\ l_{n1} & l_{n2} & l_{n3} & \cdots & 1 \end{bmatrix}, \quad U = \begin{bmatrix} u_{11} & u_{12} & u_{13} & \cdots & u_{1n} \\ 0 & u_{22} & u_{23} & \cdots & u_{2n} \\ 0 & 0 & u_{33} & \cdots & u_{3n} \\ \cdot & \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdot & \cdots & \cdot \\ 0 & 0 & 0 & \cdots & u_{nn} \end{bmatrix} \tag{2.16a}$$

Indeed, we have the following theorem.

Theorem 2.1. *An $n \times n$ invertible matrix has an LU-decomposition $PA = LU$ where L is a unit lower triangular matrix, U is an upper triangular matrix, and $P = P_{n-1}\cdots P_1$ with P_j being the j-th row exchange permutation. The j-th column of L below the diagonal is a permuted version of G_j by $P_{j+1}, \cdots, P_{n-1}$. If $G_j = I - \tilde{l}^{(j)}e_j^t$ with the Gauss vector $\tilde{l}^{(j)} = (\underbrace{0, \cdots, 0}_{j}, \tilde{l}_{j+1}, \cdots, \tilde{l}_n)^T$, then $L(j+1:n, j) = l(j+1:n)$, where $l = P_{n-1}\cdots P_{j+1}\tilde{l}^{(j)}$.*

Proof. Set $\tilde{G}_{n-1} = G_{n-1}$ and $\tilde{G}_j = P_{n-1}\cdots P_{j+1}G_jP_{j+1}\cdots P_{n-1}$ for $j = n-2, \cdots, 1$. Then,

$$\begin{aligned}\tilde{G}_{n-1}\cdots\tilde{G}_1PA &= G_{n-1}(P_{n-1}G_{n-2}P_{n-1})\cdots(P_{n-1}\cdots P_2G_1P_2\cdots)PA\\ &\vdots\\ &= (G_{n-1}P_{n-1})(G_{n-2}P_{n-2})(G_{n-3}P_{n-3})\cdots(G_1P_1)A\\ &= U.\end{aligned}$$

Since P_j is a permutation to swap j-th row and j_s-th row, $P_j(1:j-1, 1:j-1) = I_{j-1}$, which implies that $e_j^tP_{j+1}\cdots P_{n-1} = e_j^t$. Thus

$$\begin{aligned}\tilde{G}_j &= P_{n-1}\cdots P_{j+1}(I - \tilde{l}^{(j)}e_j^t)P_{j+1}\cdots P_{n-1}\\ &= I - (P_{n-1}\cdots P_{j+1})\tilde{l}^{(j)}e_j^t,\end{aligned}$$

which implies that $\tilde{G}_j$ is a Gauss transform with Gauss vector $\tilde{g}^{(j)} = P_{n-1}\cdots P_{j+1}\tilde{l}^{(j)}$. Notice that $L = (\tilde{G}_{n-1}\cdots\tilde{G}_1)^{-1}$. Hence $PA = LU$. □

The above procedure with partial pivoting can be summarized as the following simple algorithm.

ALGORITHM 2.2. *Gaussian Elimination (with partial pivoting)*

```
 1 do j=1,n-1
 2    jsave = MAXLOC(ABS(a(j:n,j),1))
 3    js = jsave(1) + j-1
 4    swap a(j,:) and a(js,:)
 5    swap b(j) and b(js)
 6    do k=j+1, n
 7       m = a(k,j)/a(j,j)
 8       a(k,k:n) = a(k,k:n) - m*a(j,k:n)
 9       b(k) = b(k) - m*b(j)
10    enddo
11 enddo
```

2.2.3 Fortran example with partial pivoting

We begin by modifying F90 Prog. 2.3 as in the following subroutine:

F90 Program 2.8: Build the matrix A for Gaussian elimination with partial pivoting

```
subroutine build_pivot_a(a, n)
  use prec
  implicit none
  integer(mpi):: n, j
  real(mp):: a(1:n,1:n)

  a = 0._mp
  do j = 1, n
     a(j,j) = 2._mp
  enddo
  do j = 1, n-1
     a(j,j+1) = -1._mp
     a(j+1,j) = -1._mp
     a(j+1, 1) = dble(j+1)**3
  enddo
end subroutine build_pivot_a
```

Then, in order to call the subroutine "build_pivot_a.f90," you have to modify the main program F90 Prog. 2.2 as in the following subroutine:

F90 Program 2.9: Main program for Gaussian elimination without pivoting

```
! Gaussian elimination without partial pivot
! Written by D. Sheen (http://www.nasc.snu.ac.kr) 2003. 3.
include "prec.f90"

program gausselim_tmp_main
  use prec
  implicit none

  integer(mpi):: n, j, k
  real(mp), allocatable :: a(:,:), b(:), savea(:,:), saveb(:)

  write(6,*) "What is the dimension n?"
  read(5,*) n ! n =10, 100, 1000
  n = n-1 ! Guess why do I put this line!
  allocate(a(1:n, 1:n), b(1:n), saveb(1:n), savea(1:n,1:n) )

  call build_pivot_a(a, n)
  call build_b(b, n)

  savea = a  ! Save the matrix a to test if the soler is correct
  saveb = b  ! Save the vector b to test if the soler is correct

  do j =1, n-1
     call gauss_tran(a(j:n,j:n),b(j:n),n,j)
  end do

  call back_subs(a,b,n) ! Solve U x = b by back substitution
  call check_axb(savea, b, saveb, n) ! Check if the solver is correct
```

```
29
30    write(6,*) "The solution x is as follows:"
31    write(6,91) b
32    write(66,91) b
33  91 format(5g12.3)
34
35  end program gausselim_tmp_main
```

The only change was made in the line 15 in the program F90 Prog. 2.10, compared to F90 Prog. 2.2. Save the files F90 Prog. 2.10 and F90 Prog. 2.8 with modified filenames, "gausselim_tmp_main.f90" and "build_pivot_a.f90," respectively. Then in oder to compile, type in the command line the following:

```
$ f90 gausselim_tmp_main.f90 build_pivot_a.f90 build_b.f90 gauss_tran.f
    back_subs.f90 check_axb.f90
```

followed by ENTER Key. Then run the program "a.out" with $n = 10, 20, 30, 40,$ and so on. You will find that the program will give answer correctly for $n = 10$ and $n = 20$. But, with $n \geq 30$, the Gaussian elimination without pivoting will give wrong solutions. Indeed, the program begins to show wrong solution with $n = 25$.

We thus have to modify the subroutine "gausselim_tmp_main.f90" as in the following program "gausselim_pivot_main.f90"

F90 Program 2.10: Main program for Gaussian elimination with partial pivoting

```
1  ! Gaussian elimination without partial pivot
2  ! Written by D. Sheen (http://www.nasc.snu.ac.kr) 2003. 3.
3  include "prec.f90"
4  program gausselim_pivot_main
5    use prec
6    implicit none
7    integer(mpi):: n, j
8    real(mp), allocatable :: a(:,:), b(:), savea(:,:), saveb(:)
9    logical:: pivot
10
11   write(6,*) "What is the dimension n?"
12   read(5,*) n ! n =10, 100, 1000
13   n = n-1 ! Guess why do I put this line!
14   allocate(a(1:n, 1:n), b(1:n), saveb(1:n), savea(1:n,1:n) )
15
16   write(6,*) "Want to use partial pivotting? Answer by .true. or .false."
17   read(5,*) pivot
18
19   call build_pivot_a(a, n)
20   call build_b(b, n)
21
22   savea = a  ! Save the matrix a to test if the soler is correct
23   saveb = b  ! Save the vector b to test if the soler is correct
24
```

```
25    do j =1, n-1
26       if(pivot) call partial_pivot(a(j:n,j:n),b(j:n),n,j)
27       call gauss_tran(a(j:n,j:n),b(j:n),n,j)
28    end do
29
30    call back_subs(a,b,n) ! Solve U x = b by back substitution
31    call check_axb(savea, b, saveb, n) ! Check if the solver is correct
32
33    write(6,*) "The solution x is as follows:"
34    write(6,91) b
35    write(66,91) b
36 91 format(5g12.3)
37
38 end program gausselim_pivot_main
```

with adding the program "partial_pivot.f90"

F90 Program 2.11: Partial pivoting for Gaussian elimination

```
1  subroutine partial_pivot(a,b,n,j)
2    use prec
3    implicit none
4
5    integer(mpi):: n, j, ksave, k
6    real(mp), intent(inout):: a(j:n,j:n), b(j:n)
7    real(mp) :: tmp(j:n)
8
9    ksave = maxloc(abs(a(j:n,j)),1)
10   k = ksave + j -1
11   tmp(:) = a(j,:)
12   a(j,:) = a(k,:)
13   a(k,:) = tmp(:)
14
15   tmp(j) = b(j)
16   b(j) = b(k)
17   b(k) = tmp(j)
18
19 end subroutine partial_pivot
```

Finally, compile with the following command:

```
$ f90 gausselim_pivot_main.f90 build_pivot_a.f90 build_b.f90 partial_pi
gauss_tran.f90 back_subs.f90 check_axb.f90
```

followed by ENTER Key. The program will be running with fairly large n.

Remark 2.1. *In F90 Prog. 2.11 "maxloc(abs(a(j:n,j)),1)" returns the location ordinal number of the maximum value among the absolute values of the array "a(j:n,j)." The last argument "1" means that the maximum is sought among the values of a varying the first dimensional index of the array, here j through n, with the second index j fixed. Indeed, either array "a(j:n,j)" or*

"a(j,j:n)" is of one dimension, and thus you have the only choice 1 *for this argument; otherwise, it won't be complied.*

If you have a two dimensional array "a(j:m, k:n)", "maxloc(abs(a(j:m,k:n)),1)" will return $n-k+1$ *values of the maximum value locations from* $a(j:m,\ell)$*, with varying the first dimensional index, for each fixed second dimensional index* $\ell = k, \cdots, n$*, while "maxloc(abs(a(j:m,k:n)),2)" will return* $m-j+1$ *values of the maximum value locations from* $a(\ell, k:n)$ *with varying the second dimensional index, for each fixed first dimensional index* $\ell = j, \cdots, m$*.*

2.3 Direct triangular decompositions

2.3.1 Doolittle's algorithm for LU-decomposition

$$
\begin{aligned}
A &= LU \\
&= \begin{bmatrix} 1 & & & 0 \\ l_{21} & 1 & & \\ \vdots & \vdots & \ddots & \\ l_{n1} & l_{n+1,1} & \cdots & 1 \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ & u_{22} & \cdots & u_{1,n+1} \\ & & \ddots & \vdots \\ & & & u_{nn} \end{bmatrix}
\end{aligned}
$$

LU-decomposition corresponds to solving for n^2 unknowns in l_{jk}'s and u_{jk}'s for n^2 equations $a_{jk} = \sum_{m=1}^{n} l_{jm}u_{mk}$, $j,k = 1 = n, \ldots,$

Based on this observation the following direct triangular decomposition algorithms are developed:

A Gaussian elimination scheme computes a sequence of lower and upper triangular matrices. Instead of computing the sequence of lower and upper triangular matrices, one can compute the components of the lower and upper triangular matrices directly. We introduce the following two algorithms: Doolittle's and Crout's algorithms. These algorithms are essentially identical to Gaussian elimination except the change of ordering in computation. Indeed, Gaussian elimination computes the inner-products of intermediate results, while direct triangular decompositions computes those without storing intermediate results, and therefore the latter schemes are slightly more precise in keeping double precision calculations consistently. In this sense, as long as no pivoting is required, the use of direct triangular decompositions have an advantages, and in actual hand calculators the direct decomposition algorithms have been implemented.

ALGORITHM 2.3. *Doolittle's algorithm for LU-decomposition*

$PA = LU$ (ASSUME $P = I$.)

L IS UNIT LOWER TRIANGULAR AND U IS UPPER TRIANGULAR.

$a_{jk} = \sum_{m=1}^{min\{j,k\}} l_{jm}u_{mk}$

```
1 do j=1,n
2    do k=j,n
3       u(j,k)=a(j,k)-DOT_PRODUCT(l(j,1:j-1), u(1:j-1,k))   a_jk = Σ_{m=1}^{j-1} l_jm u_mk + l_jj u_jk
4    enddo
5    do k=j+1,n
6       l(k,j)=(a(k,j)-DOT_PRODUCT(l(k,1:j-1),u(1:j-1,j)))/u(j,j)   a_jk =
Σ_{m=1}^{k-1} l_jm u_mk + l_jk u_kk
7    enddo
8 enddo
```

2.3.2 Crout's algorithm for LU-decomposition

The following proposition is trivial, but important.

Proposition 2.2. *Let A be an $m \times n$ matrix. Then, we have the following properties:*

1. *If D is an $m \times m$ diagonal matrix with entries $\{d_1, d_2, \cdots, d_m\}$, then*

$$DA = \begin{bmatrix} d_1 A(1, 1:n) \\ d_2 A(2, 1:n) \\ \vdots \\ d_m A(m, 1:n) \end{bmatrix}.$$

   That is, the left multiplication of a diagonal matrix is a row-wise scaling.

2. *If D is an $n \times n$ diagonal matrix with entries $\{d_1, d_2, \cdots, d_n\}$, then*

$$AD = \begin{bmatrix} d_1 A(1:m, 1) & d_2 A(1:m, 2) & \cdots & d_n A(1:m, n) \end{bmatrix}.$$

   That is, the right multiplication of a diagonal matrix is a column-wise scaling.

Proof. Trivial. □

If an $m \times n$ matrix A has an LU-decomposition with a unit-lower and an upper triangular matrics L and U, then one can have another LU-decomposition with a lower and a unit-upper triangular matrics L' and U'. Indeed, let D be the diagonal matrix of size $m \times m$ whose entries are $\{u_{11}, u_{22}, \cdots, u_{mm}\}$. Since LU is an LU-decomposition of A, D is nonsingular. Define $L' = LD$ and $U' = D^{-1}U$. Then it is easy to see that U' is a unit-upper triangular matrix.

This implies that one can find an $L'U'$-decompostion of A. Based on this observation, one may have the following Crout's algorithm in place of the Doolittle's algorithm.

ALGORITHM 2.4. *Crout's algorithm for LU-decomposition*

$PA = LU$ (ASSUME $P = I$.)

L IS LOWER TRIANGULAR AND U IS UNIT UPPER TRIANGULAR.

$a_{jk} = \sum_{m=1}^{min\{j,k\}} l_{jm} u_{mk}$

```
1 do j=1,n
2    do k=j,n
3       l(k,j) = a(k,j) - DOT_PRODUCT(l(k,1:j-1), u(1:j-1,j))        a_jk =
```

$\sum_{m=1}^{j-1} l_{jm} u_{mk} + l_{jj} u_{jk}$

```
4    enddo
5    do k=j+1,n
6       u(j,k) = (a(j,k) - DOT_PRODUCT(l(j,1:j-1), u(1:j-1,k)) )/l(j,j)    a_jk =
```

$\sum_{m=1}^{k-1} l_{jm} u_{mk} + l_{jk} u_{kk})$

```
7    enddo
8 enddo
```

Total number of flops

$$\begin{aligned}
&\sum_{j=1}^{n} [(j-1)(n-j+1) + j*(n-j)] \\
&\quad = \sum_{j=1}^{n} \left(nj - j^2 + j - n + j - 1 + nj - j^2 \right) \\
&\quad = \sum_{j=1}^{n} \left[(-2j^2) + 2(n+1)j - n - 1 \right] \\
&\quad = -2\frac{n(n+1)(2n+1)}{6} + 2(n+1)\frac{n(n+1)}{2} - (n+1)n \\
&\quad = n(n+1)\left(-\frac{2n}{3} - \frac{1}{3} + n + 1 - 1 \right) \\
&\quad = n(n+1)\left(\frac{n}{3} - \frac{1}{3} \right).
\end{aligned}$$

Exercise 2.9. *Write a Crout algorithm to solve $n \times n$ symmetric, prositive-definite matrix. Solve $Ax = b$, where A is an $n \times n$ tridiagonal matrix with $a_{jj} = 2$, $j = 1, \cdots n$, and $a_{j,j+1} = a_{j+1,j} = -1$, $j = 1, \cdots n-1$, with $b = e_1$, the first standard unit vector. Let $n = 100$.*

Exercise 2.10. *Implement the Doolittle's algorithm to solve*

$$-u''(x) = \exp(\sin(x)), x \in (0,1) \text{ with } u(0) = u(1) = 1 \tag{2.17}$$

using the uniform meshes $h = \frac{1}{16}, \frac{1}{32}, \frac{1}{64}$, and $\frac{1}{128}$.

F90 Program 2.12: Main program for Crout's algorithm

```
1  ! Crout_main.f90 Written by D. Sheen (http://www.nasc.snu.ac.kr) 2003. 3.
2  program crout_main
3    implicit none
4    integer:: n, j, k
5    real(8), allocatable :: a(:,:), b(:), x(:), savea(:,:), saveb(:)
6
7    write(6,*) "What is the dimension n?"
8    read(5,*) n ! n =10, 100, 1000
9    n = n-1 ! Guess why do I put this line!
10   allocate(a(1:n, 1:n), b(1:n), x(1:n), saveb(1:n), savea(1:n,1:n) )
11
12   call build_a(a, n)
13   call build_b(b, n)
14
```

```
15    savea = a  ! Save the matrix a to test if the soler is correct
16    saveb = b  ! Save the vector b to test if the soler is correct
17
18    call crout(a, n) ! A = LU, L:lower, U:unit-upper triangular
19    call forward_elim_lower(a,b,n) ! Solve L y = b by forward elimination
20    call back_subs_unit_upper(a,b,n) ! Solve U x = y by back substitution
21    call check_axb(savea, b, saveb, n) ! Check if the solver is correct
22
23    write(6,*) "The solution x is as follows:"
24    write(6,91) b
25    write(66,91) b
26  91 format(5g12.3)
27
28  end program crout_main
```

F90 Program 2.13: Crout's algorithm

```
1   ! Crout Algorithm written by D. Sheen http://www.nasc.snu.ac.kr
2   subroutine crout(a,n)
3     use prec
4     implicit none
5
6     integer(mpi):: n,j,k
7     real(mp):: a(1:n,1:n)
8
9     do j=1,n
10       do k=j,n
11          a(k,j)=(a(k,j)-dot_product(a(k,1:j-1),a(1:j-1,j)))
12       end do
13       do k=j+1,n
14          a(j,k)=(a(j,k)-dot_product(a(j,1:j-1),a(1:j-1,k)))/a(j,j)
15       end do
16    end do
17  end subroutine crout
```

F90 Program 2.14: forward elimination for lower triangula matrices

```
1   subroutine forward_elim(a,b,n)
2     implicit none
3     integer:: n,j
4     real(8), intent(in):: a(1:n,1:n)
5     real(8), intent(inout):: b(1:n)
6
7     do j = 1, n
8        b(j) = (b(j) - dot_product(a(j,1:j-1), b(1:j-1)))/a(j,j)
9     end do
10  end subroutine forward_elim
```

F90 Program 2.15: Back substitution algorithm for unit upper triangula matrices

```
1   subroutine back_subs_unit_upper(a,b,n)
2     use prec
3     implicit none
4     integer(mpi):: n,j
5     real(mp), intent(inout):: a(1:n,1:n), b(1:n)
```

```
6
7     do j = n, 1, -1
8        b(j) = b(j) - dot_product(a(j, j+1:n), b(j+1: n))
9     end do
10  end subroutine back_subs_unit_upper
```

2.4 Computing A^{-1}

In many cases, one needs to find the inverse A^{-1} of a square matrix A, instead of solving a linear system. In data fitting problem, the component A^{-1}_{jk} provides a certain information on the dependence of input to output data. In these cases, an actual computation of A^{-1} is desirable. There are also many situtations to solve linear systems with fixed matrix but variable right hand sides, for instance, in evolutionary partial differential equations such as heat propagation, and many optimization problems. For this, one can compute the inverse A^{-1} as follows: For $j = 1, \cdots, n$, let e^j be the j-th standard unit vector. Then compute x^j such that

$$Ax^j = e^j, j = 1, \cdots, n.$$

Then the matrix X with x^j being the j-th column vector, $j = 1, \cdots, n$ is the inverse A^{-1}. Indeed,

$$AX = \left[Ax^1 Ax^2 \cdots Ax^n\right] = \left[e^1 e^2 \cdots e^n\right] = I$$

Exercise 2.11. *Use the above scheme to find the inverse of A where A is an $n \times n$ tridiagonal matrix with $a_{jj} = 2$, $j = 1, \cdots n$, and $a_{j,j+1} = a_{j+1,j} = -1$, $j = 1, \cdots n-1$. Let $n = 10$.*

Suppose that one needs to solve m linear systems
$Ax^{(j)} = b^{(j)}, j = 1, \cdots, m$ flops : $\frac{n^3}{3} - \frac{n}{3} + mn^2$

- If $m \gg n$, A^{-1} would be helpful.
- In data fitting problem, some component A^{-1}_{jk} gives certain information on the dependence of input to output data. In these cases, an actual computation of A^{-1} is desirable.
  $PA = LU \Rightarrow A^{-1}$
  Assume that $P = I$. $I = AA^{-1}$
  Then n column vectors of A^{-1} x^j satisfy $Ax^j = e_j$, $j = 1, \cdots, n$

This amounts to solving a linear system n times
$\frac{n^3}{3} - \frac{n}{3} + n^3 \cong \frac{4}{3}n^3$ flops.

2.5 Gauss-Jordan Algorithm

In order to find the inverse A^{-1} of A, let us consider the following linear system:

$$Ax = y, \tag{2.18}$$

for general n-dimensional vectors x and y. If x can be expressed in terms of y in the following form

$$By = x, \tag{2.19}$$

the matrix B should be the inverse of A since

$$y = Ax = A(By) = (AB)y,$$

for general y and

$$x = By = B(Ax) = (BA)x,$$

for general x.

In order to the above procedure, we begin with writing (2.18) in the following form:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots a_{1n}x_n &= y_1, \\ &\vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots a_{nn}x_n &= y_n. \end{aligned}$$

The first step consists two procedures: (1) partial pivoting to find the suitable index s_1 such that $|a_{s_1}| = \max_{k\geq 1} |a_{k1}|$ after swapping the 1st and s_1th rows. Denote by σ_1 the transposition operator between 1 and s_1 such that

$$\sigma_1(1) = s_1, \sigma_1(s_1) = 1, \sigma_1(k) = k \text{ if } k \neq 1 \text{ and } k \neq s_1.$$

Denote by $\alpha_{jk}^{(1)}$'s and y_j''s the swapped coefficients and the swapped components of y. Notice that

$$y' = [y_{\sigma_1(1)}, y_{\sigma_1(2)}, \cdots, y_{\sigma_1(n)}]^t.$$

Then, we have

$$\begin{aligned} \alpha_{11}^{(1)}x_1 + \alpha_{12}^{(1)}x_2 + \cdots \alpha_{1n}^{(1)}x_n &= y_1', \\ &\vdots \\ \alpha_{n1}^{(1)}x_1 + \alpha_{n2}^{(1)}x_2 + \cdots \alpha_{nn}^{(1)}x_n &= y_n'. \end{aligned}$$

Then (2) solve the resulting first equation for x_1 and substitute x_1's in all the other equations with the x_1 represented in the first equation. Denote the resulting system as follows:

$$\begin{aligned} a_{11}'y_1' + a_{12}'x_2 + \cdots a_{1n}'x_n &= x_1, \\ a_{21}'y_1' + a_{22}'x_2 + \cdots a_{2n}'x_n &= y_2', \\ a_{31}'y_1' + a_{32}'x_2 + \cdots a_{3n}'x_n &= y_3', \\ &\vdots \\ a_{n1}'y_1' + a_{n2}'x_2 + \cdots a_{nn}'x_n &= y_n'. \end{aligned}$$

Denote by $m_{jk} = \frac{\alpha_{jk}^{(j)}}{\alpha_{jj}^{(j)}}$ for $j, k = 1, \cdots, n$. Then, the coefficients a_{jk}''s are given by

$$a_{11}' = \frac{1}{\alpha_{11}^{(1)}},\ a_{1k}' = -m_{1k}, \qquad k = 2, \cdots, n, \tag{2.20a}$$

$$a_{k1}' = m_{k1}, \qquad k = 2, \cdots, n. \tag{2.20b}$$

$$a_{k\ell}' = \alpha_{k\ell}^{(1)} - m_{k1}\alpha_{1\ell}^{(1)}, \qquad k, \ell = 2, \cdots, n. \tag{2.20c}$$

The next step begins again with partial pivot in the second column. Denote by σ_j the Then solve the resulting second equation for x_2 and substitute x_2 in all equations with the x_2 to be represented in the second equation. Write the resulting equations in the form

$$\begin{aligned} a_{11}''y_1'' + a_{12}''y_2'' + a_{13}''x_3 + \cdots a_{1n}''x_n &= x_1, \\ a_{21}''y_1'' + a_{22}''y_2'' + a_{23}''x_3 + \cdots a_{2n}''x_n &= x_2, \\ a_{31}''y_1'' + a_{32}''y_2'' + a_{33}''x_3 + \cdots a_{3n}''x_n &= y_3'', \\ &\vdots \\ a_{n1}''y_1'' + a_{n2}''y_2'' + a_{n3}''x_3 + \cdots a_{nn}''x_n &= y_n''. \end{aligned}$$

Repeat these n steps until we obtain the following system:

$$\begin{aligned} a_{11}^{(n)}y_1^{(n)} + a_{12}^{(n)}y_2^{(n)} + a_{13}^{(n)}y_3^{(n)} + \cdots a_{1n}^{(n)}y_n^{(n)} &= x_1, \\ a_{21}^{(n)}y_1^{(n)} + a_{22}^{(n)}y_2^{(n)} + a_{23}^{(n)}y_3^{(n)} + \cdots a_{2n}^{(n)}y_n^{(n)} &= x_2, \\ a_{31}^{(n)}y_1^{(n)} + a_{32}^{(n)}y_2^{(n)} + a_{33}^{(n)}y_3^{(n)} + \cdots a_{3n}^{(n)}y_n^{(n)} &= x_3, \\ &\vdots \\ a_{n1}^{(n)}y_1^{(n)} + a_{n2}^{(n)}y_2^{(n)} + a_{n3}^{(n)}y_3^{(n)} + \cdots a_{nn}^{(n)}y_n^{(n)} &= x_n. \end{aligned}$$

In the jth step, set the transposition operator between j and s_j such that

$$\sigma_j(j) = s_j, \sigma_j(s_j) = j, \sigma_j(k) = k \text{ if } k \neq j \text{ and } k \neq s_j.$$

The partial pivoting at the jth step yields $y^{(j)}$, where

$$y^{(j)} = [y_{\sigma_1\cdot\sigma_j(1)}, y_{\sigma_1\cdot\sigma_j(2)}, \cdots, y_{\sigma_1\cdot\sigma_j(n)}]^t.$$

Observe that, in the jth step, after partial pivoting, the coefficients are given by

$$a_{jj}^{(j)} = \frac{1}{\alpha_{jj}^{(j)}}; \quad a_{jk}^{(j)} = -m_{jk}, \qquad k \neq j, \tag{2.21a}$$

$$a_{kj}^{(j)} = m_{kj}, \qquad k \neq j, \tag{2.21b}$$

$$a_{k\ell}^{(j)} = \alpha_{k\ell}^{(j)} - m_{kj}\alpha_{j\ell}^{(j)}, \qquad k \neq j, \ell \neq j. \tag{2.21c}$$

Set

$$m_j = \begin{bmatrix} m_{1j} & \cdots & m_{j-1,j} & 0 & m_{j-1,j} & \cdots & m_{nj} \end{bmatrix}^t \text{ and } m_j e_j^t.$$

Then,

$$\begin{aligned} (m_j e_j^t)A &= m_j(e_j^t A) = m_j \begin{bmatrix} a_{j1} & \cdots a_{j,j-1} & a_{jj} & a_{j,j+1} \cdots a_{jn} \end{bmatrix} \\ &= \begin{bmatrix} m_{1j}a_{j1} & \cdots & m_{1j}a_{j,j-1} & m_{1j}a_{j,j} & m_{1j}a_{j,j+1} & \cdots & m_{1j}a_{jn} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ m_{j-1,j}a_{j1} & \cdots & m_{j-1,j}a_{j,j-1} & m_{j-1,j}a_{j,j} & m_{j-1,j}a_{j,j+1} & \cdots & m_{j-1,j}a_{jn} \\ 0 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ m_{j+1,j}a_{j1} & \cdots & m_{j+1,j}a_{j,j-1} & m_{j+1,j}a_{j,j} & m_{j+1,j}a_{j,j+1} & \cdots & m_{j+1,j}a_{jn} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ m_{nj}a_{j1} & \cdots & m_{nj}a_{j,j-1} & m_{nj}a_{j,j} & m_{nj}a_{j,j+1} & \cdots & m_{nj}a_{jn} \end{bmatrix} \end{aligned}$$

Notice that (2.21) is equivalent to

$$a_{jj}^{(j)} = \frac{1}{\alpha_{jj}^{(j)}}; \quad a_{jk}^{(j)} = -m_{jk}, \qquad k \neq j, \tag{2.22a}$$

$$a_{kj}^{(j)} = m_{kj}, \qquad k \neq j, \tag{2.22b}$$

$$a_{k\ell}^{(j)} = \alpha_{k\ell}^{(j)} - m_{kj}\alpha_{j\ell}^{(j)}, \qquad k \neq j, \ell \neq j. \tag{2.22c}$$

Writing P_j the permutation matrix corresponding to σ_j, finally we arrive at

$$A^{(n)}y^{(n)} = x,$$

where

$$y^{(n)} = P_n \cdots P_1 y^{(0)} = Py^{(0)} = Py, \quad \text{with } P = P_n \cdots P_1 y^{(0)} = P_n \cdots P_1.$$

since

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \overset{P_n}{\longrightarrow} \begin{bmatrix} y_{\sigma_n(1)} \\ y_{\sigma_n(2)} \\ \vdots \\ y_{\sigma_n(n)} \end{bmatrix} \overset{P_{n-1}}{\longrightarrow} \begin{bmatrix} y_{\sigma_{n-1}\sigma_n(1)} \\ y_{\sigma_{n-1}\sigma_n(2)} \\ \vdots \\ y_{\sigma_{n-1}\sigma_n(n)} \end{bmatrix} \overset{P_{n-2}}{\longrightarrow} \cdots \overset{P_1}{\longrightarrow} \begin{bmatrix} y_{\sigma_1\cdots\sigma_{n-1}\sigma_n(1)} \\ y_{\sigma_1\cdots\sigma_{n-1}\sigma_n(2)} \\ \vdots \\ y_{\sigma_1\cdots\sigma_{n-1}\sigma_n(n)} \end{bmatrix}.$$

Hence, we have

$$A^{(n)}P = A^{-1}. \tag{2.23}$$

In other words, A^{-1} is the final matrix $a_{jk}^{(n)}$ in the Gauss-Jordan* algorithm multiplied to the right by P, the matrix of partial pivoting.

Exercise 2.12. *Prove* (2.23).

Let us overwrite the coefficient matrix $a(1:n, 1:n)$ in actual programming. Then, (2.22a) can be written as follows:

$$m = a_{jj}, \quad a_{jj} = 1/a_{jj}; \quad a_{jk} = -a_{jk}a_{jj}, k \neq j.$$

while (2.22b) can be written as follows:

$$a_{kj} = a_{kj}a_{jj}, \quad k \neq j.$$

*Wilhelm Jordan (1 March 1842, Ellwangen, Wrttemberg 17 April 1899, Hanover) was a German geodesist who did surveys in Germany and Africa and founded the German geodesy journal. Notice that there is an eminent mathematician Camille Jordan (1838–1922), too. Gauss-Jordan algorithm was discovered by Jordan and Clasen, independently, in the same year 1888. See [2, 5].
Althoen, S. C.; R. McLaughlin (1987). "Gauss-Jordan Reduction: A Brief History". American Mathematical Monthly (Mathematical Association of Americ) 94 (2): 130-142.

Then (2.22c) for $k \neq j$ and $\ell \neq j$ should read as

$$a_{k\ell} = a_{k\ell} + a_{kj} m a_{j\ell}. \tag{2.24}$$

Exercise 2.13. *Show that* (2.24) *for* $k \neq j$ *and* $\ell \neq j$.

Thus, considering partial pivoting, we arrive at the following algorithm.

ALGORITHM 2.5. *Gauss-Jordan Algorithm (overwiting a(:,:))*

```
1  do j = 1,n
2     jsave=MAXLOC(ABS(a(j:n,j),1))
3     js = jsave(1) + j-1
4     swap a(j,:) and a(js,:)
5     swap b(j) and b(js)
6     m=a(j,j); a(j,j) = 1/a(j,j)
7     do while (k /=j .and. ℓ /= j)
8        a(j,k)=-a(j,j)*a(j,k)
9        a(k,j)=a(k,j)*a(j,j)
10       a(k,ℓ)=a(k,ℓ) + a(k,j)*m*a(j,ℓ)
11    enddo
12 enddo
```

The following Fortran program is based on the above algorithm.

F90 Program 2.16: Gauss-Jordan algorithm with partial pivot

```
1  ! Gauss-Jordan algorithm with partial pivot: this program computes A^{-1}.
2  ! Written by D. Sheen 2003. 3.
3  include "prec.f90"
4
5  program gauss_jordan
6    use prec
7    implicit none
8
9    logical:: pivot
10   integer(mpi):: n, j, ksave, k, l
11   integer(mpi), allocatable :: perm(:)
12   real(mp), allocatable :: a(:,:), diag(:,:), savea(:,:), tmp(:)
13   real(mp) :: m, mat_norm, p, delta
14
15   write(6,*) "What is the dimension n?"
16   read(5,*) n ! n =200
17   allocate(perm(1:n))
```

```
18    allocate(a(1:n, 1:n), diag(1:n, 1:n), savea(1:n,1:n), tmp(1:n))
19
20    write(6,*) "Do you want to use partial pivotting? Please answer by .true. or .false."
21    read(5,*) pivot
22    a = 0._mp
23    do j = 1, n
24       a(j,j) = 4._mp
25    enddo
26    do j = 1, n-1
27       a(j,j+1) = -1._mp
28       a(j+1,j) = -1._mp
29       a(min(j+3,n),j) = -5._mp
30       a(j, min(j+2,n)) = -7._mp
31    enddo
32
33    savea = a  ! for test
34
35    do j =1, n
36       if(pivot) then
37          ksave = maxloc(abs(a(j:n,j)),1)
38          k = ksave + j -1; perm(j) = k
39          tmp(1:n) = a(j, 1:n)
40          a(j, 1:n) = a(k, 1:n)
41          a(k, 1:n) = tmp(1:n)
42       end if
43
44       m=a(j,j);  a(j,j) = 1._mp/a(j,j)
45       a(j,1:j-1) = - a(j,j)*a(j,1:j-1);    a(j,j+1:n) = - a(j,j)*a(j,j+1:n)
46       a(1:j-1,j) = a(1:j-1,j)*a(j,j);      a(j+1:n,j) = a(j+1:n,j)*a(j,j)
47       do k = 1, n
48          if (k /= j) then
49             do l = 1, n
50                if (l /=j) a(k,l) = a(k,l) + a(k,j)*m*a(j,l)
51             end do
52          end if
53       end do
54    end do
55
56    if(pivot) then ! (AP)A_save = I, P=P_nP_{n-1}\cdots P_1
57 !     print*, "perm =", perm
58       do j = n-1, 1, -1
59          if(j /= perm(j) ) then ! (A_save)^{-1} = AP = column swapping
60             tmp(1:n) = a(1:n,j)
61             a(1:n,j) = a(1:n,perm(j))
62             a(1:n, perm(j)) = tmp(1:n)
63          end if
64       end do
65    end if
66
67    diag = matmul(a,savea) ! to check if a is the inverse of savea
68
69    do j =1, n
70       do k = 1, n
71          m = dot_product(diag(j,1:n), diag(1:n,k))
72          if (abs(m - delta(j,k)) > 1.e-7_mp) then
73             print*, j,k, "th", m
```

```
74              stop "Inverse is not obtained"
75            end if
76         end do
77      end do
78      print*, "Inverse of a is successfully obtained: "
79      do j = 1, n
80         write(6,96) a(1:n,j)
81      end do
82   96 format(5f12.3)
83
84      p = 1._mp
85      write(6,91) int(p), mat_norm(a,n,p)*mat_norm(savea,n,p)
86   91 format("Condition number of U in ",i4,"-norm =",f20.10)
87
88   end program gauss_jordan
89
90   function delta(j,k)
91      use prec
92      implicit none
93      integer(mpi):: j,k
94      real(mp):: delta
95      if (j==k) then
96         delta = 1._mp
97      else
98         delta = 0._mp
99      end if
100  end function delta
101
102  function mat_norm(a,n,p)
103     use prec
104     implicit none
105     real(mp):: a(1:n,1:n)
106     real(mp):: mat_norm, col, p
107     integer(mpi):: j, k, n
108
109     mat_norm = 0._mp
110     if (p==1._mp) then
111        do k = 1, n
112           col = 0._mp
113           do j = 1, n
114              col = col + abs(a(j,k))
115           end do
116           mat_norm = max(mat_norm, col)
117        end do
118     end if
119  end function mat_norm
```

Exercise 2.14. *Write a Gauss-Jordan program to find the inverse of A where A is an $n \times n$ tridiagonal matrix with $a_{jj} = 2$, $j = 1, \cdots n$, and $a_{j,j+1} = a_{j+1,j} = -1$, $j = 1, \cdots n-1$. Let $n = 10$.*

2.5.1 An alternative method to find A^{-1}

To find the inverse A^{-1} of A, assume that we have the LU-decomposition $PA = LU$. let $X = A^{-1}$, then writing $X = [x_1, \ldots, x_n] \in \mathbb{R}^{n\times n}$

$$\begin{aligned} Ax_j &= e_j \qquad j = 1, \ldots, n \qquad (2.25)\\ PAx_j &= Pe_j \qquad j = 1, \ldots, n \qquad (2.26)\end{aligned}$$

Setting $Ux_j = y_j$ for $j = 1, \ldots, n$, $Ly_i = Pe_j$ by forward elimination and $Ux_j = y_j$ have $\frac{n^3}{3} + n\left(\frac{n(n-1)}{2} + \frac{n(n+1)}{2}\right) \simeq \frac{4}{3}n^3$ flops.

2.6 Existence of an LU-decomposition

Definition 2.3. *Let $A = (a_{ij})$ be an $m \times n$ matrix. Consider the subindices*

$$(i_p)_{p=1,\cdots,r} \qquad \text{and } (j_q)_{q=1,\cdots,s}$$

satisfying

$$1 \le i_1 < \cdots < i_r \le m, \quad 1 \le j_1 < \cdots < j_s \le n.$$

Then $B = (b_{pq}) = (a_{i_p j_q})$ is called a submatrix of A. In particular if $r = s$ and $i_p = j_p$ for all $p = 1, \cdots, r$, B is called a principal submatrix of A. Moreover, if in addition $i_p = j_p = p$ for all $p = 1, \cdots, r$, B is called a leading principal submatrix of A.

Theorem 2.2 (LU-decomposition)**.** *Suppose that all the leading principal submatrices of A are nonsingular. Then there exist a unit lower triangular matrix L and an upper triangular matrix U such that $A = LU$.*

Proof. In the Gaussian elimination procedure, for $1 \le k \le n-1$, we have

$$M_k A^{(k-1)} = M_k M_{k-1} \cdots M_1 A^{(0)},$$

where $A^{(0)} = A$ and $M_k = I - m^{(k)} e_k^t$ is a Gaussian transformation matrix. Denote

$$A^{(k)} = \begin{bmatrix} A_{11}^{(k)} & A_{12}^{(k)} \\ 0 & A_{22}^{(k)} \end{bmatrix},$$

where $A_{11}^{(k)}$ is the $k \times k$ upper triangular matrix and $A_{12}^{(k)}$ and $A_{22}^{(k)}$ denote the remaining $k \times (n-k)$ and $(n-k) \times (n-k)$ block submatrices of $A^{(k)}$. Since

$$\begin{aligned} A &= A^{(0)} = (M_k M_{k-1} \cdots M_1)^{-1} \begin{bmatrix} A_{11}^{(k)} & A_{12}^{(k)} \\ 0 & A_{22}^{(k)} \end{bmatrix} \\ &= \begin{bmatrix} L_{11}^{(k)} & 0 \\ L_{21}^{(k)} & I_{n-k} \end{bmatrix} \begin{bmatrix} A_{11}^{(k)} & A_{12}^{(k)} \\ 0 & A_{22}^{(k)} \end{bmatrix} \\ &= \begin{bmatrix} L_{11}^{(k)} A_{11}^{(k)} & L_{11}^{(k)} A_{12}^{(k)} \\ L_{21}^{(k)} A_{11}^{(k)} & L_{21}^{(k)} A_{12}^{(k)} + A_{22}^{(k)} \end{bmatrix}, \end{aligned}$$

we observe that the leading $k \times k$ principal submatrix of A, denoted by A_k, is equal to $L_{11}^{(k)} A_{11}^{(k)}$. Utilizing the fact that $L_{11}^{(k)}$ is a unit lower triangular matrix and that $A_{11}^{(k)}$ is an upper triangular matrix, one sees that

$$\det(A_k) = \det(L_{11}^{(k)}) \det(A_{11}^{(k)}) = \det(A_{11}^{(k)}) = a_{11}^{(k)} a_{22}^{(k)} \cdots a_{kk}^{(k)}.$$

The Guassian elimanation procedure will continue as long as $a_{11}^{(k)} a_{22}^{(k)} \cdots a_{kk}^{(k)} \neq 0$. until $k = n-1$. □

2.6.1 LDM^t-decomposition

Theorem 2.3 ([3]LDM^t-decomposition)**.** *Suppose that all the leading principal submatrices of A are nonsingular. Then there exist L, M (unit lower triangular matrices) that $A = LDM^t$, D diagonal matrix. (Assume that $P = I$)*

Proof. By the assumption, there exists an LU-decomposition of A: $A = LU$. Let $D = \mathrm{diag}\{u_{11}, \cdots, u_{nn}\} = \begin{pmatrix} u_{11} & & 0 \\ & \ddots & \\ 0 & & u_{nn} \end{pmatrix}$. Since A is nonsingular, $u_{jj} \neq 0$, $(j = 1, \cdots, n)$. Thus $D^{-1} = \mathrm{diag}\{u_{11}^{-1}, \cdots, u_{nn}^{-1}\}$. Let $M^t = D^{-1}U$. Then M^t is a unit upper triangular and M is a unit lower triangular. Hence $A = LU = LDD^{-1}U = LDM^t$.

□

ALGORITHM 2.6. $A = LDM^t \quad D = diag\{d_1, \cdots, d_n\}$

```
1 do j=1,n
2    a_jj = Σ_{p=1}^{j-1} l_jp d_p m_jp + l_jj d_j m_jj  ⇒  d_j = a_jj − Σ l_jp d_p m_jp
3    do k=j+1,n                                                    FOR k > j
4       l_kj = (a_kj − Σ_{p=1}^{j-1} l_kp d_p m_jp)/d_j            a_kj = Σ_{p=1}^{j-1} l_kp d_p m_jp + l_kj d_j m_jj,  m_jj = 1)
5       m_kj = (a_jk − Σ_{p=1}^{j-1} l_jp d_p m_kp)/d_j            a_jk = Σ_{p=1}^{j-1} l_jp d_p m_kp + l_jj d_j m_kj,  l_jj = 1)
6    enddo
7 enddo
```

The above algorithm requires $\frac{2n^3}{3}$ flops. However, notice that

$$r_{jp} = d_p m_{jp}, \quad j = 1, \cdots, n-1,$$

$$w_{jp} = l_{jp} d_p, \quad j = 1, \cdots, n-1$$

are independent of the inner loo index i. Compute these before executing the inner loop, to have the following algorithm of flops $\frac{n^3}{3}$:

ALGORITHM 2.7. *$A = LDM^t$ (reduced version)*

```
1 do j=1,n
2    do p=1,j-1
3       r_p = d_p a_pj
4       w_p = a_jp d_p
5    enddo
6    d_j = a_jj − Σ_{p=1}^{j-1} a_jp r_p
7    do k=j+1,n
8       a_kj = (a_kj − Σ_{p=1}^{j-1} a_kp r_p)/d_j
9       a_jk = (a_jk − Σ_{p=1}^{j-1} w_p a_pk)/d_j
10   enddo
11 enddo
```

Exercise 2.15. *Write a program to decompose A as an LDM^t, where A is an*

$n \times n$ tridiagonal matrix with $a_{jj} = 2$, $j = 1, \cdots n$, and $a_{j,j+1} = a_{j+1,j} = -1$, $j = 1, \cdots n-1$. Let $n = 100$.

2.6.2 LDL^t-decomposition of symmetric matrix

Proposition 2.3 (LDL^t-decomposition of symmetric matrix)**.** *Suppose that all the leading principal submatrices of A are nonsingular. For a nonsingular symmetric matrix A, there exists a unit lower triangular matrix L and a diagonal matrix D such that $A = LDL^t$.*

Proof. We start with $A = LDM^t$ (L,M:unit lower triangular matrices.)

$$\Rightarrow \underbrace{M^{-1}AM^{-T}}_{symmetric} = \underbrace{M^{-1}LD}_{lower\ triangular}$$

Therefore, $M^{-1}L$ is diagonal. Moreover, it is unit lower triangular, which implies that $M^{-1}L = I$. Hence $L = M$. □

The Gaussian elimination of an $n \times n$ matrix ($A = LU$ decomposition) needs $\frac{n^3}{3} - \frac{n}{3}$ flops.

ALGORITHM 2.8. *$A = LDL^t$ (reduced version)*

```
1 do j=1,n
2    do p=1,j-1
3       r_p = d_p a_pj
4    enddo
5    d_j = a_jj − Σ_{p=1}^{j−1} a_jp r_p
6    do k=j+1,n
7       a_kj = (a_kj − Σ_{p=1}^{j−1} a_kp r_p)/d_j
8    enddo
9 enddo
```

Exercise 2.16. *Count the number of flops of the above algorithm.*

Exercise 2.17. *Implement the above algorithm to solve*

$$-u''(x) = 1, x \in (0,1); \quad u(0) = 0, u(1) = 1.$$

Definition 2.4. *An $n \times n$ real matrix A is called* positive-definite *if $x^tAx \geq c(x_1^2 + \cdots + x_n^2)$ for all $x \in R^n$, where $c > 0$ is independent of x. For complex cases, an $n \times n$ complex matrix A is postive-definite if $x^*Ax \geq c(|x_1|^2 + \cdots + |x_n|^2)$ for all $x \in \mathbb{C}^n$, $(x^* = \bar{x}^t)$, where $c > 0$ is again independent of x.*

Proposition 2.4. *Let A be a positive-definite matrix. Then, all the diagonal entries of A are positive and all the eigenvalues of A are positive.*

Proof. Suppose that A is a positive-definite matrix. For $j = 1, \cdots, n$, let $x = e^{(j)}$ be the j–th standard unit vector. Then,

$$a_{jj} = (e^{(j)})^* A e^{(j)} \geq c > 0, \text{ for som } c > 0.$$

Thus, all the diagonal entries are postive.

Next, let λ be any eigenvalue of A with a corresponding eigenvector $\xi \in \mathbb{C}^n$. Then, since $A\xi = \lambda\xi$,

$$0 < c|\xi|^2 \leq \xi^* A\xi = \xi^* \lambda \xi = \lambda |\xi|^2.$$

Hence, all eigenvalues $\lambda \geq c > 0$. □

Theorem 2.4. *If $A \in \mathbb{R}^{n \times n}$ is positive-definite, then there exist a diagonal matrix with with positive diagonal entries and unit lower triangular matrices L and M such that $A = LDM^t$.*

Proof. Since $x^tAx > 0$, for all $x \in \mathbb{R}^n \backslash \{0\}$, all principal submatrices of A are positive-definite. Then A has a decomposition $A = LDM^t$ by the previous theorem. Set $S = DM^tL^{-T} = L^{-1}AL^{-T}$. Then S is positive-definite and upper triangular with the jth diagonal element s_{jj} is equal to d_j for $j = 1, \cdots, n$. □

2.6.3 Cholesky decomposition

Carl F. Gauss (1777-1855) formulated and studied intensively least squares problems, which generate the normal equation $A^tAx = A^tb$. Here, A and b are a given $m \times n$ matrix and given m dimensional vector, while $x \in \mathbb{R}^n$ is supposed to be sought. Usually $m \geq n$. The matrix A^tA is symmetric; moreover, it is positive-definite if and only if rank $(A) = n$.

Andre-Louis Cholesky (1875–1918) When A is a symmetric, positive-definite matrix.

Theorem 2.5 (Cholesky decomposition)**.** *Let A be a symmetric, positive-definite $n \times n$ matrix. Then there exists a lower triangular matrix G such that $A = GG^t$ with positive diagonal entries of G.*

Proof. $A = LDM^t$. Since A is symmetric, $L = M$. Thus,

$$A = LDL^t = LD^{1/2}D^{1/2}L^t = LD^{1/2}(LD^{1/2})^t = GG^t.$$

□

ALGORITHM 2.9. *Cholesky decomposition*

1 do j=1,n $\qquad a_{kj} = \sum_{p=1}^{j} g_{kp}g_{jp}$ *if* $k \geq j$)

2 $\quad a_{jj}(= g_{jj}) = \sqrt{a_{jj} - \sum_{p=1}^{j-1} a_{jp}^2}$

3 $\quad$ do k=j+1,n

4 $\quad\quad a_{kj}(= g_{kj}) = (a_{kj} - \sum_{p=1}^{j-1} a_{kp}a_{jp})/a_{jj}$

5 $\quad$ enddo

6 enddo

Exercise 2.18. *Write a Cholesky decomposition program to solve $n \times n$ symmetric, prositive-definite matrix. Solve $Ax = b$, where A is an $n \times n$ tridiagonal matrix with $a_{jj} = 2$, $j = 1, \cdots n$, and $a_{j,j+1} = a_{j+1,j} = -1$, $j = 1, \cdots n-1$, with $b = e_1$, the first standard unit vector. Let $n = 100$.*

2.6.4 Banded matrices

Definition 2.5. *$A = (a_{ij})$ has an upper bandwidth b_u if a_{ij}=0 for $j > i + b_u$.*

$$\begin{bmatrix} \ddots & \cdots & \ddots & & & 0 \\ & a_{i-b_u,i-b_u} & \cdots & a_{i-b_u,i} & & \\ & & \ddots & \vdots & \ddots & \\ & & & a_{ii} & \cdots & a_{i,i+b_u} \\ * & & & & \ddots & \cdots \quad \ddots \end{bmatrix}$$

and A has a low bandwidth b_l if $a_{ij} = 0$ for $i < j + b_l$

$$\begin{bmatrix} \ddots & \cdots & \ddots & & & * \\ & a_{i,i-b_l} & \cdots & a_{ii} & & \\ & & \ddots & \vdots & \ddots & \\ & & & a_{i+b_l,i} & \cdots & \ddots \\ 0 & & & & \ddots & \cdots & \ddots \end{bmatrix}$$

A has lower and upper bandwidth b_l and b_u $a_{ij} = 0$ for $i > j + b_l$ or $j > i + b_u$.

Thus if A has lower and upper bandwidth b and b, the only possible nonzero entries of A are within the bandwidth $2b + 1$ around the diagonal.

Example 2.3 (Finite difference or finite element method)**.** *The standard finite difference method for the following boundary value problem*

$$-\frac{d^2u}{dx^2}(x) = f(x) \quad x \in (a, b), \tag{2.27a}$$

$$u(a) = u(b) = 0eq : 1d - ell \tag{2.27b}$$

is as follows. Consider a uniform mesh $x_j, j = 0, 1, \cdot, N$, with $x_j = a + jh$ and $h = \frac{b-a}{N}$. The finite difference method tries to approximate the values $u(x_j), j = 1, \cdots, N-1$, satisfying

$$-\frac{d^2u}{dx^2}(x_j) = f(x_j),\ j = 1, \cdots, N-1,$$

which is approximated by

$$\frac{-\tilde{u}(x_{j-1}) + 2\tilde{u}(x_j) - \tilde{u}(x_{j+1})}{h^2} = f(x_j),\ j = 1, \cdots, N-1$$

assuming that $\tilde{u}(x_0) = 0 = \tilde{u}(x_N)$ which reflects the boundary condition (2.27b)*. Denoting by U_j and b_j the approximate values $\tilde{u}(x_j)$ and $f(x_j)$ for $j = 1, \cdots, N-1$, we have*

$$\frac{-U_{j-1} + 2U_j - U_{j+1}}{h^2} = b_j,\ j = 1, \cdots, N-1, \tag{2.28a}$$

which form an $N-1$ equations in $N-1$ unknowns.

Let $\mathbf{U}$ and $\mathbf{b}$ denote $N-1$–dimensional vectors such that $\mathbf{U} = (U_1, \cdots, U_{N-1})^t$ and $\mathbf{b} = (h^2 f(x_1), \cdots, h^2 f(x_{N-1}))^t$. Then we are led to the following matrix value problem

$$A\mathbf{U} = \mathbf{b},$$

where A is an $(N-1)\times(N-1)$ matrix given by

$$A=\frac{1}{h^2}\begin{bmatrix} 2 & -1 & 0 & 0 & \cdots & 0 & 0 & 0 & 0\\ -1 & 2 & -1 & 0 & \cdots & 0 & 0 & 0 & 0\\ 0 & -1 & 2 & -1 & \cdots & 0 & 0 & 0 & 0\\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots & \vdots & \vdots & \vdots\\ \vdots & \vdots & \vdots & \ddots & \ddots & \ddots & \vdots & \vdots & \vdots\\ \vdots & \vdots & \vdots & \vdots & \ddots & \ddots & \ddots & \vdots & \vdots\\ 0 & 0 & 0 & 0 & \cdots & -1 & 2 & -1 & 0\\ 0 & 0 & 0 & 0 & \cdots & 0 & -1 & 2 & -1\\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & -1 & 2 \end{bmatrix}, \tag{2.29}$$

which has lower bandwidth=1, upper bandwidth=1, so named tridiagonal matrix.

Remark 2.1. *The above matrix is positive-definite. Indeed, consider*

$$\begin{aligned} x^t(h^2A)x &= \sum_i\sum_j x_i a_{ij} x_j\\ &= 2(x_1^2+\cdots+x_n^2-x_1x_2-x_2x_3-\cdots-x_{n-1}x_n)\\ &= x_1^2+(x_1-x_2)^2+(x_2-x_3)^2+\cdots+(x_{n-1}-x_n)^2+x_n^2\geq 0. \end{aligned}$$

Now assume that $x^tAx=0$. Then one has $x_1=x_2=\cdots=x_n=0$, which implies that A is positive definite.

Exercise 2.19. *If A is the above tridiagonal matrix, show that all the components of A^{-1} is positive.*

Example 2.4 (Positive-definiteness and ellipticity)**.** *The matrix A of the linear system $Ax=b$ obtained by the finite difference method of the elliptic problem*

$$-\frac{d}{dx}\left[c(x)\frac{du}{dx}(x)\right] = f(x) \quad x\in(a,b), \tag{2.30}$$

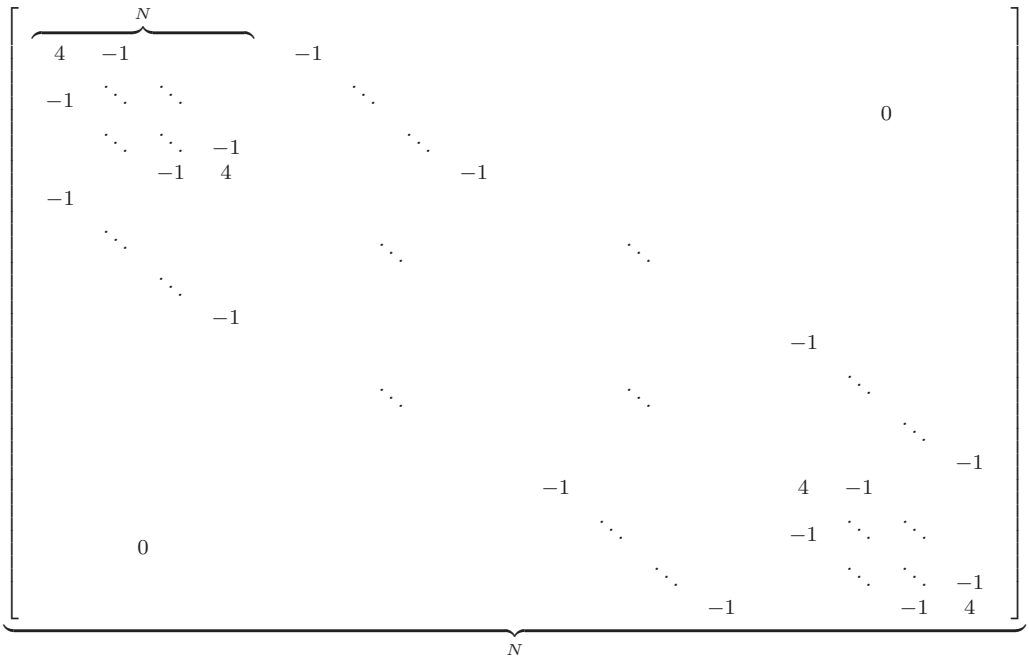
$$u(a)=u(b) = 0, \tag{2.31}$$

is positive-definite if $c(x)\geq 0$ for all $x\in(a,b)$. If $c(x)\geq 0$, the above differential equation is called "a uniformly elliptic differential equation".

Example 2.5. *[The 2D-finite difference method] Consider the two dimensional elliptic problem:*

$$-\Delta u(x) = f(x), \quad x \in (0,1)^2, \tag{2.32a}$$
$$u(x) = g(x), \quad x \in \partial(0,1)^2, \tag{2.32b}$$

$$A = \frac{1}{h^2}$$



upper bandwidth is N and lower bandwidth is N.

Exercise 2.20. *Show that the above matrix A is positive-definite.*

The following theorem implies that the band-structure of LU-decomposition does not change from that of A.

Theorem 2.6 (LU-decomposition of banded matrices)**.** *Let A has the LU-decomposition $A = LU$ and lower and upper bandwidth b_l and b_u. Then L has lower bandwidth b_l and U has upper bandwidth b_u.*

Proof. Write $A = \begin{bmatrix} \alpha & \overrightarrow{w}^t \\ \overrightarrow{v} & B \end{bmatrix}$ for $\alpha \in \mathbb{R}$, $\overrightarrow{v} \in \mathbb{R}^{n-1}$, $\overrightarrow{w} \in \mathbb{R}^{n-1}$ and B is $(n-1) \times (n-1)$ matrix.

$$\begin{aligned} (\textit{one step}) &= \begin{bmatrix} 1 & \overrightarrow{0} \\ \overrightarrow{v}/\alpha & I_{n-1} \end{bmatrix} \begin{bmatrix} \alpha & \overrightarrow{w}^t \\ \overrightarrow{0} & B - (\overrightarrow{v}\,\overrightarrow{w}^t)/\alpha \end{bmatrix} \\ &= \begin{bmatrix} 1 & \overrightarrow{0} \\ \overrightarrow{v}/\alpha & I_{n-1} \end{bmatrix} \begin{bmatrix} 1 & \overrightarrow{0} \\ \overrightarrow{0} & B - (\overrightarrow{v}\,\overrightarrow{w}^t)/\alpha \end{bmatrix} \begin{bmatrix} \alpha & \overrightarrow{w}^t \\ \overrightarrow{0} & I_{n-1} \end{bmatrix} \end{aligned}$$

Since B and $\vec{v}\,\vec{w}^t$ are $(n-1)\times(n-1)$ matrices with lower and upper bandwidth b_l and b_u, $B-(\vec{v}\,\vec{w}^t)/\alpha$ has lower and upper bandwidth b_l and b_u. By induction assumption we may assume that $B-(\vec{v}\,\vec{w}^t)/\alpha$ has LU-decomposition such that $B-(\vec{v}\,\vec{w}^t)/\alpha = L_1U_1$ with L_1 has lower bandwidth b_l and U_1 has upper bandwidth b_u.

$$A=\underbrace{\begin{bmatrix}1 & \vec{0}\\ \vec{v}/\alpha & I_{n-1}\end{bmatrix}\begin{bmatrix}1 & \vec{0}\\ \vec{0} & L_1\end{bmatrix}}_{lower\ bandwidth\ b_l}\underbrace{\begin{bmatrix}1 & \vec{0}\\ \vec{0} & U_1\end{bmatrix}\begin{bmatrix}\alpha & \vec{w}^t\\ \vec{0} & I_{n-1}\end{bmatrix}}_{upper\ bandwidth\ b_u} \tag{2.33}$$

□

ALGORITHM 2.10. *LU-decomposition without pivoting*

```
1 do j=1,n-1
2    ml = MIN(j + b_l, n)
3    a(j+1:ml, j) = a(j+1:ml, j) / a(j,j)
4    do k=j+1,ml
5       mu = MIN(j + b_u, n)
6       a(k, j+1:mu) = a(k, j+1:mu) - a(k,j)*a(j,j+1:mu)
7    enddo
8 enddo
```

ALGORITHM 2.11. *forward elimination*

```
1 do j=1,n
2    k = MAX(1,j - b_l)
3    b(j) = b(j) - DOT_PRODUCT(a(j, k:j-1), b(k:j-1))
4 enddo
```

ALGORITHM 2.12. *back substitution*

```
1 do j=n,1,-1
2    k = MIN(j + b_u, n)
3    b(j) = ( b(j) - DOT_PRODUCT(a(j, j+1:k), b(j+1:k)) ) / a(j,j)
4 enddo
```

Exercise 2.21. *Implement the above LU-decomposition and forward elimination and back substitution to solve Example 2.5 with $f(x,y) = 1$ and $g(x,y) = 1$ with mesh size $h = 1/100$.*

Exercise 2.22. *Check that LU-decomposition requires*

$$\begin{cases} nb_l b_u - \frac{1}{2}b_l b_u^2 - \frac{b_l^3}{6} + b_l n \text{ flops} & \text{if } b_l \le b_u \\ nb_l b_u - \frac{1}{2}b_l^2 b_u - \frac{b_u^3}{6} + b_l n \text{ flops} & \text{if } b_l > b_u \end{cases}$$

$$\text{forward elimination} \ : \ nb_l - \frac{b_l^2}{2} \text{ flops}$$

$$\text{back substitution} \ : \ n(b_u + 1) - \frac{b_u^2}{2} \text{ flops}$$

Thus $N^2 \times N^2$ matrix with bandwidth N, $n = N^2$ and $b_l = b_u = N$

$$O\left(N^4 - \frac{1}{2}N^3 - \frac{1}{6}N^3 + N^3 + N^3 - \frac{1}{2}N^2 + N^2(N+1) + \frac{1}{2}N^2\right) = O(N^4)$$

Due to Theorem 2.5 and Theorem 2.6, Algorithm 2.5 can be optimized as follows:

ALGORITHM 2.13. *Cholesky decomposition of symmetric, positive-definite, bandwidth*

```
1 do j=1,n
2    mj = MAX(j − b_w, 1)
3    a(j, j) = SQRT(a(j, j) - DOT_PRODUCT(a(j, mj:j-1), a(j, mj:j-1)) )
4    do k=j+1,MIN(n, j + b_w)
5       mk = MAX(k − b_w, 1)
6       a(k, j) = ( a(k, j) - DOT_PRODUCT(a(k, mk:j-1), a(j, mk:j-1)) ) /
a(j, j)
7    enddo
8 enddo
```

$$\text{do} \quad k = 1, n \tag{2.34a}$$
$$m = \max(1, k - b_w) \tag{2.34b}$$
$$a_{kk} = \sum_{l=m}^{k} g_{kl} g_{kl} \tag{2.34c}$$
$$a_{jk} = \sum_{l=m}^{min(j,k)} g_{jl} g_{kl} \tag{2.34d}$$
$$\text{end do} \tag{2.34e}$$

Exercise 2.23. *Implement the banded Cholesky decomposition solver and apply it to solve the 2D Poisson problem Example 2.5 with $f(x, y) = 1$ and $g(x, y) = 1$ with mesh size $h = 1/100$.*

2.6.5 A practical example of tridiagonal matrix

Consider the tridiagonal matrix

$$A = \begin{bmatrix} a_{11} & a_{12} & & & & & \\ a_{21} & a_{22} & a_{23} & & & & \\ & a_{32} & a_{33} & a_{34} & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & a_{n-1,n-2} & a_{n-1,n-1} & a_{n-1,n} \\ & & & & a_{n,n-1} & a_{n,n} \end{bmatrix}$$

which has lower bandwidth=1 and upper bandwidth=1. We assume that the matrix has an LU-decomposition.

$$L = \begin{bmatrix} 1 & 0 & & 0 & 0 & 0 \\ l_{21} & 1 & 0 & & & \\ & l_{32} & 1 & 0 & & \\ & & \ddots & \ddots & \ddots & \\ & & & l_{n-1,n-2} & 1 & 0 \\ & & & & l_{n,n-1} & 1 \end{bmatrix}$$

$$U = \begin{bmatrix} u_{11} & u_{12} & & & & \\ 0 & u_{22} & u_{23} & & & \\ & 0 & u_{33} & u_{34} & & \\ & & \ddots & \ddots & \ddots & \\ & & & 0 & u_{n-1,n-1} & u_{n-1,n} \\ & & & & 0 & u_{n,n} \end{bmatrix}$$

Comparing the components of $A = LU$ column by column, one gets for each $j = 1, \cdots, n$

$$\begin{aligned} a_{j-1,j} &= u_{j-1,j}, && (2.35a) \\ a_{j,j} &= u_{jj} + l_{j,j-1}u_{j-1,j}, && (2.35b) \\ a_{j+1,j} &= l_{j+1,j}u_{j,j}, && (2.35c) \end{aligned}$$

where $a_{1,0} = a_{0,1} = 0$ are assumed and similarly $l_{1,0}$ and $u_{0,1}$ are defined. Thus, we have the algorithm

$$\begin{aligned} u_{j-1,j} &= a_{j-1,j} && (2.36a) \\ u_{j,j} &= a_{jj} - l_{j,j-1}u_{j-1,j} && (2.36b) \\ l_{j+1,j} &= a_{j+1,j}/u_{j,j}. && (2.36c) \end{aligned}$$

Thus, we have the algorithm

ALGORITHM 2.14.

```
1 do j=1,n
2    u(j-1,j) = a(j-1,j)
3    u(j,j) = a(j,j) - l(j,j-1)* u(j-1,j)
4    l(j+1,j) = a(j+1,j)/ u(j,j)
5 enddo
```

Now, we observe that the values of $a(j-1,j), a(j,j), a(j+1,j)$ used at the step j will not be called again in the next steps. Therefore, one can overwrite $a(j-1,j), a(j,j), a(j+1,j)$ with the corresponding values of $u(j-1,j), u(j,j), l(j+1,j)$. This leads to the overwriting algorithm:

ALGORITHM 2.15.

```
1 do j=1,n
```

```
2    a(j-1,j) = a(j-1,j) ! this statement is not needed now
3    a(j,j) = a(j,j) - a(j,j-1)* a(j-1,j)
4    a(j+1,j) = a(j+1,j)/ a(j,j)
5 enddo
```

Now in order to minimize allocating memories, we introduce $ad(1:n), al(0:n-1), au(0:n-1)$ to replace the diagonal, lower and upper diagonal entries. Using this, the above algorithm is rewritten as follows:
ALGORITHM 2.16.

```
1 do j=1,n
2    ad(j) = ad(j) - al(j-1)* au(j-1)
3    al(j) = al(j)/ ad(j)
4 enddo
```

Exercise 2.24. *Using the above algorithm, write a compact program to solve Exercise 2.10.*

Notice that we utilized the first and second indices of upper and lower diagonal entries of $a(j-1,j)$ and $a(j+1,j)$ to match $au(j-1)$ and $al(j)$.

2.6.6 Thomas algorithm: tridiagonal matrix solver

The original Thomas[†] algorithm is essentially the LU-decomposition with the unit lower and upper triangular matrices L and U.

F90 Program 2.17: Thomas algorithm

```
1 ! Modified as of Apr. 21, 2011 to solve -u" = f; u(0)=(1)=0
2 ! LU decomposition of a tridaiangal matrix overwriting ad, al, au
3 ! D. Sheen (http://www.nasc.snu.ac.kr)
4 include "prec.90"
5 program tridiag_thomas
6   use prec
7   implicit none
8
9   integer(mpi) :: n, nx, j
```

[†]Llewellyn Hilleth Thomas (21 October 1903 20 April 1992) was a British physicist and applied mathematician. He is best known for his contributions to atomic physics (http://en.wikipedia.org/wiki/Llewellyn_Thomas), especially in connection with the Thomas-Fermi electron gas model.
Thomas, L.H., Elliptic problems in linear difference equations over a network, Watson Sci. Comput. Lab. Rept., Columbia University, New York, 1949.

```
  real(mp), allocatable :: ad(:), au(:), al(:), b(:), saveb(:)
!  complex(mpc), allocatable :: ad(:), au(:), al(:), b(:), saveb(:)

  write(6,*) "What is the number of meshes?   nx ="
  read(5,*) nx ! nx =200
  n = nx-1
  allocate(ad(1:n), au(1:n-1), al(2:n), b(1:n), saveb(1:n) )

  !  ad(1) au(1) 0 .........
  !  al(2) ad(2) au(2) .....
  !  .........
  !  .........  al(n) ad(n)
  ad = 2._mp
  al = -1._mp
  au = -1._mp

  b = 0._mp
  b(1) = 1._mp
  saveb(:) = b(:)

  call thomas(ad,al,au,b,n)

  do j = 1, n
     if ( abs(saveb(j) - (2._mp*b(j) - b(j-1) - b(j+1)  )    ) > 5.e-13_mp ) then
        print*, "The computed solution seems to be wrong at j = ", j
        stop "Something wrong"
     end if
  end do

!     if ( abs(saveb(j) - dot_product(savea(j,1:n), x(1:n))) > 5.d-13 ) then
  write(6,*) "The solution x is as follows:"
  write(66,91) b
  write(6,91) b
91 format(5g12.3)

end program tridiag_thomas
```

Save F90 Prog. 2.17 with the file name "tridiag_thomas.f90." Also, save the following program F90 Prog. 2.18 with the file name "thomas.f90."

F90 Program 2.18: Gaussian elimination without pivoting

```
! LU decomposition of a tridaiangal matrix overwriting ad, al, au
! D. Sheen (http://www.nasc.snu.ac.kr) identical to the Gaussian eliminaton
subroutine thomas(ad,al,au,b,n) ! 5n-4 flops algorithm (Thomas algorithm)
  implicit none
  integer :: n, j
  real(8) :: m
  real(8):: ad(1:n), au(1:n-1), al(2:n), b(1:n)

  do j = 2, n ! LU-decomposition
     m = al(j)/ad(j-1);  al(j) = m
     ad(j) = ad(j) -  m *au(j-1)
     b(j) = b(j) - m * b(j-1)
  end do

```

```
15    b(n) = b(n)/ad(n)    !back substitution
16    do j = n-1, 1, -1
17       b(j) = ( b(j) - au(j)*b(j+1) ) / ad(j)
18    end do
19
20  end subroutine thomas
```

Then compile both programs by typing

```
f90 tridiga_thomas.f90 thomas.f90
```

Try to run the program with various inputs.

2.6.7 Gaussian elimination of a banded matrix with partial pivoting.

Recall that

$$P = P_n P_{n-1} \cdots P_2 P_1,$$

where P_j is the permutation matrix at jth step, which swaps j-th row with s_j-th row. Let $e_{p_j}, j = 1, \cdots, n$, denote the n-dimensional unit vector with 1 at the $p_j$the entry. Then, we have the following property.

Exercise 2.25. *There exist $p_j, j = 1, \cdots, n$, such that $\{1, \cdots, n\} = \{p_1, \cdots, p_n\}$ and the permutation matrix P is written as*

$$P = \begin{bmatrix} e_{p_1}^t \\ e_{p_2}^t \\ \vdots \\ e_{p_n}^t \end{bmatrix}, \tag{2.37a}$$

$$P^t = [e_{p_1}, \cdots, e_{p_{n-1}}, e_{p_n}]. \tag{2.37b}$$

Exercise 2.26. *Show the following identities.*

1. *$P_j = P_j^t$ and $P_j^2 = I$.*
2. *$P^t = P^{-1}$.*

We have the following LU-decomposition theorem for banded matrics.

Theorem 2.7. *Suppose that A is an invertible $n \times n$ matrix with lower and upper bandwidth b_l and b_u. For $j = 1, 2, \ldots, n-1$, set $G_j = I - g^{(j)} e_j^t$, and $P_1, P_2, \ldots, P_{n-1}$ to be the permutation matrices such that*

$$U = G_{n-1} P_{n-1} G_{n-2} P_{n-1} \cdots G_1 P_1 A$$

is upper triangular. Then in the $PA = LU$-decomposition, U has upper band $b_l + b_u$ and $g_j{}^{(k)} = 0$ if $j \le k$ or $j > k + b_l$.

Proof. From (2.37), there exist disctinct $p_1, \cdots, p_n$, $1 \le p_j \le n$, such that

$$P^t = [e_{p_1}, \cdots, e_{p_{n-1}}, e_{p_n}].$$

We first claim that

$$p_j - j \le b_l. \tag{2.38}$$

To show this, assume the contrary $p_j - j > b_l$, or $p_j - b_l - 1 \ge j$. We will deduce a contradiction from this. We look at the leading $j \times j$ principal submatrix of PA. Under the assumption, we see that the first $p_j - b_l - 1$ entries of jth row of PA vanish. Indeed, let k be any integer between 1 and $p_j - b_l - 1$.

$$(PA)_{jk} = \sum_{\ell=1}^{n} p_{j\ell} a_{\ell k} = p_{j,p_j} a_{p_j,k} = a_{p_j,k}.$$

since only p_{j,p_j} is nonzero among the jth row of P. But, the fact that $p_j \ge k + b_l + 1$ and A has lower bandwidth b_l implies that $a_{p_j,k} = 0$. And thus,

$$0 = (PA)_{jk} = (LU)_{jk},\ k = 1, \cdots, p_j - b_l - 1.$$

The assumption $p_j - b_l - 1 \ge j$ implies that the jth column of U has zero entries for $k = 1, \cdots, p_j - b_l - 1 \ge j$, including the diagonal entry of U in that column. Hence U is singular, and thus A is singular, too. This is a contradiction, from which we conclude that $p_j - j \le b_l$ for $j = 1, \ldots, n$.

This means that the jth row of PA can be replaced with rows at most b_lth lower than jth row. Thus PA has upper bandwidth $b_l + b_u$. Indeed, $(PA)_{jk} = a_{p_j,k} = 0$ for $k > j + b_l + b_u$ as $k - p_j > j + b_l + b_u - p_j \ge b_u$, since $p_j - j \le b_l$.

By the previous Theorem 2.6 (without partial pivoting), L has lower bandwidth b_l and U has upper bandwidth $b_l + b_u$.

The fact that $g_j{}^{(k)} = 0$ for $j \le k$ or $j > k + b_l$ follows from the observation that only elements of kth column below the diagonal and not lower than $k + b_l$th need to be zero by G_k. That is, the entries $(k+1, k), \ldots, (k + b_l, k)$ of $G_{k-1}P_{k-1} \cdots G_1 P_1 A$ need to be zero by G_k. □

Recall that $L = P(G_{n-1}P_{n-1}\cdots G_1P_1A)^{-1}$ has all the diagonal entries being equal to 1.

Remark 2.2. *If a matrix from LU-decomposition is nonsingular resulting, all its principal submatrices are nonsingular.*

ALGORITHM 2.17.

```
1 do j=1,n-1
2    k=MAXLOC(ABS(a(j : j + b_l, j)) + j-1)
3    SWAP(a(j, j : j + b_l + b_u), a(k, j : j + b_l + b_u))
4    do k=j+1, MIN(j + b_l, n)
5       m = a(k,j)/a(j,j)
6       a(k, j + 1 : j + b_l + b_u) = a(k, j + 1 : j + b_l + b_u)  - m*a(j, j + 1 : j + b_l + b_u)
7    enddo
8 enddo
```

Exercise 2.27. *Solve the following problem*

$$-\Delta u(x,y) = f(x,y), \quad x \in (0,1)^2, \tag{2.39a}$$
$$u(x,y) = 0, \quad (x,y) \in \partial(0,1)^2, \tag{2.39b}$$

by the finite difference method with the mesh size $h = 1/100$ *and* $f(x,y) = 2\pi^2 \sin(x)\sin(y)$ *using (1) the method of LU-decomposition of banded matrix with partial pivoting and (2) the Cholesky method of banded matrix.*

Chapter 3

Matrix norms and condition number

3.1 Norm

Definition 3.1. *Let $F = \mathbb{R}$ or $F = \mathbb{C}$. A norm $\|\cdot\|$ on a vector space X over F is a nonnegative real-valued function, $\|\cdot\| : X \to \mathbb{R}^+$ is such that*

1. $\|x\| \geq 0$ *for* $\forall x \in X$,
2. $\|\alpha x\| = |\alpha|\|x\|$ *for* $\forall \alpha \in F, \qquad \forall x \in X$,
3. $\|x + y\| \leq \|x\| + \|y\|$ *for* $\forall x, y \in X$,
4. $\|x\| = 0$ *iff* $x = 0$.

If 1, 2, 3 are satisfied, $\|\cdot\|$ is called a seminorm.

We will implicitly assume that all the vector spaces are over the field $F = \mathbb{R}$ or $F = \mathbb{C}$.

Example 3.1 (Hölder norm or p-norm)**.** *

$$\begin{aligned} \|x\|_p &:= \left(\sum_{j=1}^{N} |x_j|^p \right)^{\frac{1}{p}} \qquad 1 \leq p < \infty, \\ \|x\|_\infty &:= \max_{1 \leq j \leq N} |x_j|, \end{aligned}$$

Hölder inequality ($p \geq 1$, $q \geq 1$, $\frac{1}{p} + \frac{1}{q} = 1$) †

$$|x^* y| = |\overline{x}^T y| \left(= \sum_{j=1}^{N} \bar{x}_j y_j \right) \leq \|x\|_p \|y\|_q \qquad \forall x, y \in \mathbb{C}^N$$

*$\|x\|_\infty$ is called *∞-norm* or *sup-norm*

†if $p = q = 2$, it is called *Cauchy-Schwarz inequality*

Example 3.2. *Unit circles in* $\mathbb{R}^2$ *with respect to p-norm*

$$\{x \in \mathbb{R}^2; \quad \|x\|_p = 1\}.$$

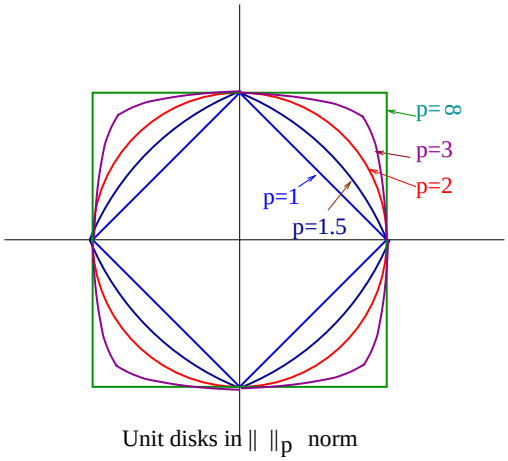


Figure 3.1.1: Unit disks in $\|\cdot\|_p$ norm in $\mathbb{R}^2$ for $1 \le p \le \infty$.

Definition 3.2. *A linear transformation* Q *is called unitary (or orthogonal if* A *is a transformation in real vector space) if*

$$\bar{Q}^t Q = Q^* Q = I = QQ^* = Q\bar{Q}^t.$$

Remark 3.1. *If* Q *is unitary, then*

$$\|Qx\|_2 = \|x\|_2 \qquad \forall x \in X.$$

Indeed,

$$\|Qx\|_2{}^2 = (Qx)^*(Qx) = x^*Q^*Qx = x^*Ix = \|x\|_2.$$

If A is a real, $n \times n$ matrix, A is called orthogonal if

$$A^T A = AA^T = I. \tag{3.1}$$

If A is a complex, $n \times n$ matrix, A is called unitary if

$$A^* A = AA^* = I \tag{3.2}$$

Thus if A is a complex, $n \times n$ matrix and A satisfies only (3.1) instead of (3.2), it is called complex orthogonal.

Notice that a complex orthogonal matrix is not unitary. Recall that the complex matrices are applied to a complex vector. Let us try to have an example for this case. Consider the 2×2 matrix

$$M(t) = \begin{pmatrix} \cosh(t) & i\sinh(t) \\ -i\sinh(t) & \cosh(t) \end{pmatrix}$$

Then show that $M(t)$ is orthogonal, but not unitary. Then can you say that the matrix $M(t)$ has eigenvalues bounded?

Definition 3.3. *Two norms $\|\cdot\|_\alpha$ and $\|\cdot\|_\beta$ on a vector space are called equivalent if there exist positive constants c_1 and c_2 such that*

$$c_1\|x\|_\beta \le \|x\|_\alpha \le c_2\|x\|_\beta \qquad \forall x \in X.$$

Exercise 3.1. *Let A_1 be an $(n-1)\times(n-1)$ principal submatrix of an $n \times n$ matrix A. Then show that $\|A_1\|_p \le \|A\|_p$ for any p-matrix norm $\|\cdot\|_p$.*

Exercise 3.2. *For $\forall x \in \mathbb{R}^N$, show that*

$$\begin{aligned} \|x\|_2 &\le \|x\|_1 \le \sqrt{N}\|x\|_2, \\ \|x\|_\infty &\le \|x\|_2 \le \sqrt{N}\|x\|_\infty, \\ \|x\|_\infty &\le \|x\|_1 \le N\|x\|_\infty. \end{aligned}$$

Also describe when the equalities hold.

Definition 3.4. *A function $f : X \to \mathbb{R}$ is called uniformly continuous if $\forall \varepsilon > 0 \quad \exists \delta(\varepsilon) > 0$ such that*

$$\|x - y\| < \delta \Longrightarrow |f(x) - f(y)| < \varepsilon$$

Example 3.3. *$f(x) = 1/x$ is continuous, but not uniformly continuous. Let $\varepsilon > 0$ be given, then $\forall \delta > 0$ we can find $x, y \in (0, 1]$ such that $\|x - y\| < \delta$ and $|f(x) - f(y)| > \varepsilon$*

$$\begin{aligned} \left|\frac{1}{x} - \frac{1}{y}\right| &= \left|\frac{x-y}{xy}\right| \qquad \text{let} \quad y - x = \frac{\delta}{2} \quad x, y = (0, 1] \\ &= \frac{\delta/2}{xy} > \frac{\delta}{2}\frac{1}{y} \ge \varepsilon \end{aligned}$$

Remark 3.2. *A continuous function on a compact set is uniformly continuous.*

Table 3.1: Example 3.3

Function	Range	Continuous	Uniform
$f(x) = x$	$x \in \mathbb{R}$	yes	yes
$f(x) = \frac{1}{x}$	$x \in (0, \infty)$	yes	no

Lemma 3.1. *A norm $\|\cdot\|$ on $\mathbb{C}^N$ is uniformly continuous with respect to $\|\cdot\|_\infty$.*

Proof. Let $\varepsilon > 0$ be given, then $\forall \quad x, y \in \mathbb{C}^N$,

$$\begin{aligned}\|x - y\| &= \|(x_1 - y_1)e_1 + (x_2 - y_2)e_2 + \cdots + (x_N - y_N)e_N\| \\ &\le \sum_{j=1}^{N} |x_j - y_j| \|e_j\| \\ &\le \max_j |x_j - y_j| \sum_{j=1}^{N} \|e_j\| \\ &= \|x - y\|_\infty \sum_{j=1}^{N} \|e_j\|.\end{aligned}$$

Thus if we choose $\delta = \frac{\varepsilon}{c} = \varepsilon / \sum_{j=1}^{N} \|e_j\| \quad \forall x, y,$

$$\|x - y\|_\infty < \delta \Longrightarrow \|x - y\| < \varepsilon.$$

□

Theorem 3.1. *Any two norms on $\mathbb{C}^N$ are equivalent. Indeed, any two norms on a finite-dimensional vector space are equivalent.*

Proof. We will show that any norm $\|\cdot\|$ is equivalent to $\|\cdot\|_\infty$ that is, $\exists c_1 > 0$ and $\exists c_2 > 0$ such that

$$c_1 \|x\|_\infty \le \|x\| \le c_2 \|x\|_\infty, \qquad \forall x \in \mathbb{C}^N.$$

For this, set $S = \left\{ x \in \mathbb{C}^N \middle| \|x\|_\infty = 1 \right\}$ then S is a compact set.

Since $\|\cdot\|$ is (uniformly) continuous by the previous lemma, there exist $c_1 := \min_{x \in S} \|x\|$ and $c_2 := \max_{x \in S} \|x\|$. Since the zero vector 0 is not included in S, $c_1 > 0$. Then $\forall y \neq 0$, $\left\| \frac{y}{\|y\|_\infty} \right\|_\infty = 1$ implies that $\frac{y}{\|y\|_\infty} \in S$.

Therefore,

$$c_1 \leq \left\| \frac{y}{\|y\|_\infty} \right\| \leq c_2, \forall y \in \mathbb{C}^N, y \neq 0.$$

This implies that

$$c_1 \|y\|_\infty \leq \|y\| \leq c_2 \|y\|_\infty.$$

Thus $\|\cdot\|$ and $\|\cdot\|_\infty$ are equivalent. □

3.2 Matrix norm

Let $\mathcal{M}(m,n)$ be the set of all $m \times n$ matrices over $\mathbb{C}$(or $\mathbb{R}$).

Definition 3.5. *A matrix norm $\|\cdot\| : \mathcal{M}(m,n) \longrightarrow \mathbb{R}^+$ is a nonnegative real-valued function such that*

1. $\|A\| \geq 0, \qquad \forall A \in \mathcal{M}(m,n)$
2. $\|\alpha A\| = |\alpha| \|A\|, \qquad \forall \alpha \in \mathbb{C}, \quad \forall A \in \mathcal{M}(m,n)$
3. $\|A + B\| \leq \|A\| + \|B\|, \qquad \forall A, B \in \mathcal{M}(m,n)$
4. $\|A\| = 0$ *if and only if* $A = 0$.

Example 3.4. *Let $\|A\|_\Delta$ be defined by*

$$\|A\|_\Delta = \max_{j,k} |a_{jk}|.$$

Then $\|\cdot\|_\Delta$ defines a norm.

But if $A = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$, $A^2 = \begin{bmatrix} 2 & 2 \\ 2 & 2 \end{bmatrix}$, then $\|A^2\|_\Delta = 2, \quad \|A\|_\Delta = 1.$

$$\|A^2\|_\Delta \not\leq \|A\|_\Delta \|A\|_\Delta.$$

Definition 3.6. *A matrix norm on $\mathcal{M}(n,n)$ is* submultiplicative *if*

$$\|AB\| \leq \|A\| \|B\|, \qquad \forall A, B \in \mathcal{M}(n,n).$$

Definition 3.7. *A matrix norm $\|\cdot\|$ on $\mathcal{M}(m,n)$ is said to be* consistent *with respect to the vector norms $\|\cdot\|_\alpha$ on $\mathbb{C}^n$ and $\|\cdot\|_\beta$ on $\mathbb{C}^m$ if $\|Ax\|_\beta \leq \|A\| \|x\|_\alpha, \quad \forall x \in \mathbb{C}^n, \quad A \in \mathcal{M}(m,n)$.*

Definition 3.8. *A matrix norm* $\|\cdot\|$ subordinate *to the vector norms* $\|\cdot\|_\alpha$ *on* $\mathbb{C}^n$ *and* $\|\cdot\|_\beta$ *on* $\mathbb{C}^m$ *is defined by*

$$\|A\|_{\alpha,\beta} = \sup_{x\neq 0, x\in\mathbb{C}^n} \frac{\|Ax\|_\beta}{\|x\|_\alpha} = \sup_{\|x\|_\alpha=1, x\in\mathbb{C}^n} \|Ax\|_\beta.$$

Note that a subordinate norm is consistent with respect to the associated norms.

Example 3.5 (Matrix norm)**.** *1. Frobenius norm(F-norm, Schur norm.)*

$$\|A\|_F = \left(\sum_{j=1}^{m}\sum_{k=1}^{n} |a_{jk}|^2\right)^{1/2}.$$

2. p-norms. ($p \geq 1$.)

$$\|A\|_p = \sup_{x\neq 0} \frac{\|Ax\|_p}{\|x\|_p}.$$

(subordinate to $\|\cdot\|_p$ *norms).*

Theorem 3.2. *If* A *is an* $m \times n$ *matrix, then*

$$\max_{j,k} |a_{jk}| \leq \|A\|_2 \leq \sqrt{mn}\, \max_{j,k} |a_{jk}|. \tag{3.3}$$

Proof. Let $|a_{j_0k_0}| = \max_{j,k} |a_{jk}|$. Then, recalling $\|A\|_2 = \max_{\|\mathbf{x}\|_2=1} \|A\mathbf{x}\|_2$, we have

$$|a_{j_0k_0}| \leq \sqrt{\sum_{j=1}^{m} a_{jk_0}^2} = \|A\mathbf{e}^{(k_0)}\|_2 \leq \|A\|_2,$$

$\mathbf{e}^{(k_0)}$ being the k_0th standard unit vector in $\mathbb{R}^n$.

Conversely, let B be the $m \times n$ matrix such that $b_{jk} = 1$ for all j, k. Certainly $\|A\| \leq |a_{j_0k_0}|\|B\|$ for any matrix norm $\|\cdot\|$. By Cauchy-Schwarz inequality, one has

$$\|B\|_2 = \max_{\mathbf{x}\neq\mathbf{0}} \frac{\|Bx\|_2}{\|\mathbf{x}\|_2} = \max_{\mathbf{x}\neq\mathbf{0}} \frac{\sqrt{m(\sum_{k=1}^n x_k)^2}}{\|\mathbf{x}\|_2} \leq \max_{\mathbf{x}\neq\mathbf{0}} \frac{\sqrt{mn\sum_{k=1}^n x_k^2}}{\|\mathbf{x}\|_2} = \sqrt{mn}.$$

This proves the thorem. □

Exercise 3.3. *Let $A = (a_{jk})$ be an $m \times n$ matrix over $\mathbb{C}$.*

1. *The 1-norm = the column norm: Show that*

$$\|A\|_1 = \max_{k=1,\cdots,n} \sum_{j=1}^{m} |a_{jk}|.$$

   That is, the 1-norm is the maximum of 1-norms of all columns of the matrix A.

2. *The ∞-norm (=the maximum norm = the sup norm) = the row norm: Show that*

$$\|A\|_\infty = \max_{j=1,\cdots,m} \sum_{k=1}^{n} |a_{jk}|.$$

   That is, the ∞-norm is the maximum of 1-norms of all rows of the matrix A.

3. *Show that $\|AB\|_p \le \|A\|_p \|B\|_p$ for all $1 \le p \le \infty$. Of course, B is an $n \times m$ matrix.*

4. *Suppose $A = \mathrm{diag}(\mu_1, \mu_2, \cdots, \mu_k)$, $k = \min\{m, n\}$. Show that*

$$\|A\|_p = \max_{j=1,\cdots,k} |\mu_j|.$$

5. *If $u \in \mathbb{C}^m$, and $v \in \mathbb{C}^n$, show that $\|uv^*\|_2 = \|u\|_2 \|v\|_2$ and $\|uv^*\|_\infty = \|u\|_\infty \|v\|_1$. What can be said of $\|uv^*\|_1$?*

6. *Let $y \in \mathbb{C}^m$ and $0 \ne x \in \mathbb{C}^n$. Show that $X = (y - Ax)x^*/(x^*x)$ is the solution with the smallest 2-norm satisfying $(A + X)x = y$.*

Proposition 3.1. *1. The subordinate matrix norm is the smallest* consistent *norm with respect to the vector norms $\|\cdot\|_\alpha$ and $\|\cdot\|_\beta$.*

2. *Subordinate matrix norms are submultiplicative.*

3. *Let $m \times m$ matrix Q and $n \times n$ matrix Z be unitary transformations. Then for any $m \times n$ matrix A, we have*

$$\begin{aligned} \|QAZ\|_2 &= \|A\|_2, \\ \|QAZ\|_F &= \|A\|_F. \end{aligned}$$

Proof of Proposition 3.1. The first assertion is proved already.

Let A and B be $m \times n$ and $n \times p$ matrices. Then

$$\begin{aligned} \|AB\|_{\alpha,\beta} &= \sup_{0\neq x\in\mathbb{C}^p} \frac{\|ABx\|_\beta}{\|x\|_\alpha} = \sup_{0\neq x\in\mathbb{C}^p, 0\neq Bx\in\mathbb{C}^m} \frac{\|ABx\|_\beta}{\|Bx\|_\gamma}\frac{\|Bx\|_\gamma}{\|x\|_\alpha} \\ &\leq \sup_{0\neq x\in\mathbb{C}^p} \frac{\|Bx\|_\gamma}{\|x\|_\alpha} \sup_{0\neq y\in\mathbb{C}^m} \frac{\|Ay\|_\beta}{\|y\|_\gamma} = \|B\|_{\alpha,\gamma}\|A\|_{\gamma,\beta}. \end{aligned}$$

This proves the second proposition.

For the third proposition (3.4), recall that unitary matrices preserve 2-norms by the definition. Therefore,

$$\begin{aligned} \|QAZ\|_2 &= \sup_{x\neq 0, x\in\mathbb{C}^n} \frac{\|QAZx\|_2}{\|x\|_2} \\ &= \sup_{x\neq 0, x\in\mathbb{C}^n} \frac{\{(QAZx)^*(QAZx)\}^{1/2}}{\|x\|_2} \\ &= \sup_{x\neq 0, x\in\mathbb{C}^n} \frac{(x^*Z^*A^*Q^*QAZx)^{1/2}}{\|x\|_2} \\ &= \sup_{Zx\neq 0, Zx\in\mathbb{C}^n} \frac{\|A(Zx)\|_2}{\|Zx\|_2} \\ &= \|A\|_2. \end{aligned}$$

Suppose $Z = I$. To show $\|QA\|_F = \|A\|_F$:

$$\begin{aligned} \|QA\|_F^2 &= \sum_{j,k} \left| \sum_{l=1}^n q_{jl}a_{lk} \right|^2 \\ &= \sum_{k=1}^n \left(\sum_{j=1}^m |\sum_{l=1}^m q_{jl}a_{lk}|^2 \right) \\ &= \sum_{k=1}^n \sum_{j=1}^m |a_{jk}|^2 \qquad \text{since } Q \text{ is orthogonal} \\ &= \|A\|_F^2. \end{aligned}$$

Next suppose $Q = I$ and we will show $\|AZ\|_F = \|A\|_F$. For this, from the above it follows that $\|Z^*A^*\|_F = \|A\|_F$. Therefore,

$$\|QAZ\|_F^2 = \|AZ\|_F^2 = \|A\|_F^2.$$

□

Exercise 3.4. *Prove that the p-matrix norm of a matrix is always greater than or equal to that of its principal submatrix.*

From now on, we shall assume that matrix norms are consistent and sub-multiplicative.

3.3 Condition Number

3.3.1 The concept of a condition number $\kappa(A)$

Condition numbers important in the numerical analysis of a linear system. They provide a measure of ill-posedness of the system, which means how much errors in solutions depend on the relative changes in the data and matrices.

Want to know about the relative error $\frac{\|\Delta x\|}{\|x\|}$ in terms of $\frac{\|\Delta b\|}{\|b\|}$.

First, consider $Ax = b$, $A(x + \Delta x) = b + \Delta b$. $b + \Delta b$ is a perturbed data and $x + \Delta x$ is its associated solution. We want to see $\frac{\|\Delta x\|}{\|x\|} \sim \frac{\|\Delta b\|}{\|b\|}$. We have $A\Delta x = \Delta b$, so that $\Delta x = A^{-1}\Delta b$. Since $\|b\| \leq \|A\|\|x\|$, we have

$$\frac{\|\Delta x\|}{\|x\|} \leq \frac{\|A\|\left\|A^{-1}\Delta b\right\|}{\|b\|} \leq \frac{\|A\|\left\|A^{-1}\right\|\|\Delta b\|}{\|b\|} = \kappa(A)\frac{\|\Delta b\|}{\|b\|},$$

where

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

Thus the relative change in x is bounded by the relative change in b multiplied by $\kappa(a)$.

Definition 3.9. *For an invertible $A \in \mathcal{M}(n, n)$, the condition number $\kappa(A)$ is defined by $\kappa(A) = \|A\|\,\|A^{-1}\|$. Note that the condition number depends on the choice of norm $\|\cdot\|$.*

Remark 3.3. $\kappa(A) \geq 1$ *since* $1 = \|I\| = \|AA^{-1}\| \leq \|A\|\|A^{-1}\| = \kappa(A)$.

Summary

1 $\kappa(A)$ measures the sensitivity of the relative error in the solution change to the change in the RHS b
$\frac{\|\Delta x\|}{\|x\|} \leq \kappa(A)\frac{\|\Delta b\|}{\|b\|}$.

2 $\|\Delta x\| \leq \|A^{-1}\|\|\Delta b\|$
$\tilde{x}$: approximate solution to $Ax = b, \quad \tilde{x} = x + \Delta x$.
With the residual $r(\tilde{x}) := b - A\tilde{x} \quad (r(x) = 0)$, we have
$\|\Delta x\| \leq \|A^{-1}\|\|r(\tilde{x})\|$, since $r(\tilde{x}) = b - A\tilde{x} = b - A(x + \Delta x) = -A\Delta x$.

Lemma 3.2. *If $F \in \mathcal{M}(n, n)$ with $\|F\| < 1$, then $I + F$ is invertible and $\left\|(I + F)^{-1}\right\| < 1/(1 - \|F\|)$*

Proof. $\forall x \in \mathbb{C}^n, \quad x \neq 0$, $\|(I + F)x\| = \|x + Fx\| \geq \|x\| - \|Fx\| \geq \|x\| - \|F\|\|x\| = \|x\|(1 - \|F\|) > 0$.
Thus $I + F$ is invertible. Moreover,

$$\begin{aligned} 1 &= \|I\| = \left\|(I + F)^{-1}(I + F)\right\| \\ &= \left\|(I + F)^{-1} + (I + F)^{-1}F\right\| \\ &\geq \left\|(I + F)^{-1}\right\| - \left\|(I + F)^{-1}F\right\| \\ &\geq \left\|(I + F)^{-1}\right\| - \left\|(I + F)^{-1}\right\|\|F\| \\ &= \left\|(I + F)^{-1}\right\|(1 - \|F\|) \end{aligned}$$

Consequently,

$$\left\|(I + F)^{-1}\right\| \leq \frac{1}{1 - \|F\|}.$$

□

Theorem 3.3. *Let $A \in \mathcal{M}(n,n)$ invertible $B = A(I+F)$ with $\|F\| < 1$. Consider $Ax = b$, $B(x+\Delta x) = b$. Then,*

$$\frac{\|\Delta x\|}{\|x\|} \leq \frac{\|F\|}{1-\|F\|}.$$

Moreover,

$$\frac{\|\Delta x\|}{\|x\|} \leq \frac{\kappa(A)\frac{\|B-A\|}{\|A\|}}{1-\kappa(A)\frac{\|B-A\|}{\|A\|}}.$$

Proof. $B^{-1} = (I+F)^{-1}A^{-1}$ exists by lemma 3.2. Since $x = A^{-1}b$, $\Delta x = B^{-1}b - x = (B^{-1}-A^{-1})b = B^{-1}(A-B)A^{-1}b$. We have

$$\begin{aligned}
\frac{\|\Delta x\|}{\|x\|} &= \frac{\left\|B^{-1}(A-B)A^{-1}b\right\|}{\|A^{-1}b\|} \\
&\leq \frac{\left\|B^{-1}(A-B)\right\|\left\|A^{-1}b\right\|}{\|A^{-1}b\|} \\
&= \left\|(I+F)^{-1}A^{-1}(A-B)\right\| \\
&= \left\|(I+F)^{-1}A^{-1}(-AF)\right\| \\
&\leq \left\|(I+F)^{-1}\right\|\|F\| \\
&\leq \frac{1}{1-\|F\|}\|F\|.
\end{aligned}$$

Since $\left\|A^{-1}(B-A)\right\| = \|F\|$,

$$\frac{\|\Delta x\|}{\|x\|} \leq \frac{\left\|A^{-1}(B-A)\right\|}{1-\|A^{-1}(B-A)\|} \leq \frac{\left\|A^{-1}\right\|\|B-A\|}{1-\|A^{-1}\|\|B-A\|} = \frac{\kappa(A)\frac{\|B-A\|}{\|A\|}}{1-\kappa(A)\frac{\|B-A\|}{\|A\|}}.$$

□

Now assuming that B^{-1} exists in the previous theorem, set $C = (I+F)^{-1} = B^{-1}A$, $F = A^{-1}B - I$. If $\|F\| << 1$, B^{-1} can be regarded as an approximate inverse of A. $\left\|B^{-1}A\right\| \leq \frac{1}{1-\|F\|}$ by lemma 3.2.

Interchanging A and B, $\left\|A^{-1}B\right\| \leq \frac{1}{1-\|B^{-1}A-I\|}$. Since $A^{-1} = A^{-1}(BB^{-1}) = (A^{-1}B)B^{-1}$,

$$\left\|A^{-1}\right\| \leq \left\|A^{-1}B\right\|\left\|B^{-1}\right\| \leq \frac{\left\|B^{-1}\right\|}{1-\|B^{-1}A-I\|}.$$

For $Ax = b$, assume that $\tilde{x}$ is an approximation to x, and B^{-1} is available. Writing $r(\tilde{x}) := b - A\tilde{x} = Ax - A\tilde{x} = A(x - \tilde{x})$,

$$\|\Delta x\| = \|x - \tilde{x}\| \le \left\|A^{-1}\right\| \|r(\tilde{x})\| \le \frac{\left\|B^{-1}\right\|}{1 - \|B^{-1}A - I\|} \|r(\tilde{x})\|.$$

Theorem 3.4. *Suppose that $Ax = b$ and $(A + \Delta A)(x + \Delta x) = (b + \Delta b)$, If $\|\Delta A\| < \frac{1}{\|A^{-1}\|}$, then*

$$\frac{\|\Delta x\|}{\|x\|} \le \frac{\kappa(A)}{1 - k(A)\frac{\|\Delta A\|}{\|A\|}} \left\{ \frac{\|\Delta A\|}{\|A\|} + \frac{\|\Delta b\|}{\|b\|} \right\}$$

Proof. $\left\|A^{-1}\Delta A\right\| \le \left\|A^{-1}\right\| \|\Delta A\| < 1$. By lemma 3.2, $I + A^{-1}\Delta A$ is invertible and $\|(I + A^{-1}\Delta A)^{-1}\| \le \frac{1}{1 - \|A^{-1}\Delta A\|} \le \frac{1}{1 - \|A^{-1}\| \|\Delta A\|}$.

$$\begin{aligned} A^{-1}(b + \Delta b) &= A^{-1}(A + \Delta A)(x + \Delta x) \\ &= (I + A^{-1}\Delta A)(x + \Delta x) \\ &= (I + A^{-1}\Delta A)\Delta x + x + A^{-1}\Delta A x \\ &= (I + A^{-1}\Delta A)\Delta x + A^{-1}(b + \Delta A x). \end{aligned}$$

then by canceling $A^{-1}b$ from the two end sides,

$$\begin{aligned} \frac{\|\Delta x\|}{\|x\|} &\le \frac{\left\|(I + A^{-1}\Delta A)^{-1}\left\{A^{-1}(\Delta b - \Delta A x)\right\}\right\|}{\|x\|} \\ &\le \frac{\left\|(I + A^{-1}\Delta A)^{-1}\right\| \left\|A^{-1}\right\| \|\Delta b - \Delta A x\|}{\|x\|} \\ &\le \frac{\left\|A^{-1}\right\|}{1 - \|A^{-1}\| \|\Delta A\|} \left\{ \frac{\|\Delta b\|}{\|x\|} + \|\Delta A\| \right\} \\ &= \frac{\kappa(A)}{1 - \kappa(A)\frac{\|\Delta A\|}{\|A\|}} \left\{ \frac{\|\Delta b\|}{\|A\| \|x\|} + \frac{\|\Delta A\|}{\|A\|} \right\} \\ &\le \frac{\kappa(A)}{1 - \kappa(A)\frac{\|\Delta A\|}{\|A\|}} \left\{ \frac{\|\Delta A\|}{\|A\|} + \frac{\|\Delta b\|}{\|b\|} \right\}. \end{aligned}$$

□

The following inequality is useful in estimation.

Lemma 3.3. *For any $\varepsilon > 0$ the Cauchy inequalities hold:*

$$xy \le \varepsilon x^2 + \frac{y^2}{4\varepsilon}, \quad \textit{and} \quad 2xy \le \varepsilon x^2 + \frac{y^2}{\varepsilon} \quad \forall x, y \in \mathbb{R}. \tag{3.4}$$

Let A be the $n \times n$ tridiagonal matrix as given in (2.29):

$$A = \begin{bmatrix} d & u & & & & \\ l & d & u & & & \\ & l & d & u & & \\ & & \ddots & \ddots & \ddots & \\ & & & l & d & u \\ & & & & l & d \end{bmatrix}$$

Chapter 4

Eigenvalues and eigenvectors

In this section all matrices A are assumed to be square.

Right eigenvector x and eigenvalue λ: $Ax = \lambda x, \quad x \neq 0.$

Left eigenvector y and eigenvalue λ: $y^*A = \lambda y^*, \quad y \neq 0.$

Rayleigh quotient:

$$\lambda = \frac{x^*Ax}{x^*x},$$

where λ is the associated eigenvalue of an eigenvector x.

Charateristic polynomial:

$$\begin{aligned} P_A(\lambda) &= det(A - \lambda I) = (\lambda_1 - \lambda)\cdots(\lambda_n - \lambda) \\ &= (-1)^n\lambda^n + (-1)^{n-1}(\lambda_1 + \cdots + \lambda_n)\lambda^{n-1} + \cdots + \lambda_1\lambda_2\cdots\lambda_n \\ &= \sum_{k=0}^{n} a_k\lambda^k. \end{aligned}$$

where λ_j is the eigenvalue of A. Notice that

(i) [Determinant of A:]

$$\mathrm{Det}(A) = \Pi_{j=1}^{n}\lambda_j = P_A(0).$$

(ii) [Trace of A:]

$$\mathrm{Tr}(A) = \sum_{j=1}^{n}\lambda_j.$$

(iii) [Cayley-Hamilton Theorem:]

$$P_A(A) = 0.$$

(iv) [Corollary of Cayley-Hamilton Theorem:] Suppose A is invertible. Then the inverse of A is given by

$$A^{-1} = -\frac{1}{a_0}\sum_{k=1}^{n} a_k A^{k-1}.$$

Proof. Since A is invertible, $a_0 = \mathrm{Det}(A) \neq 0$. Hence, it follows from the formula $P_A(A) = 0$ that

$$a_0 I = -A(\sum_{k=1}^{n} a_k A^{k-1}),$$

which gives the desired representation of A^{-1}. □

Eigenspaces of A: For an eigenvalue λ of A, the following space is called the eigenspace of A associated with λ:

$$E_\lambda(A) = \{x \in \mathbb{R}^n : Ax = \lambda x\}.$$

Spectrum of A: $\sigma(A)$ is the set of all eigenvalues of A.

Spectral radius:

$$\rho(A) = \max_{\lambda \in \sigma(A)} (|\lambda|).$$

Notice that $\sigma(A^*) = \overline{\sigma(A)}$. Indeed, for any $\lambda \in \sigma(A^*)$,

$0 = \det(A^* - \lambda I) = \det[(A - \bar{\lambda} I)^*]$ which is equivalent to $\det(A - \bar{\lambda} I) = 0$.

Theorem 4.1 (Cayley-Hamilton Theorem)**.** $P_A(A) = 0$, *where* $P_A(\lambda)$ *is the characteristic polynomial of* A.

Let

$$P_A(\lambda) = (\lambda_1 - \lambda)^{m_1} \cdots (\lambda_k - \lambda)^{m_k}.$$

with $\sum_{j=1}^{k} m_j = n$. m_j is called the algebraic multiplicity of λ_j. $\det(A - \lambda I) = 0$, $\lambda_1, \ldots, \lambda_k$ are eigenvalues.

The dimension of eigenspace $E_{\lambda_j}(A)$ corresponding to λ_j is called the "geometric multiplicity of λ_j".

Then we have

Proposition 4.1.

The geometric multiplicity of $\lambda_j \leq$ the algebraic multiplicity of $\lambda_j = m_j$(4.1)

Exercise 4.1. *If A is a real symmetric $n \times n$ matrix or a complex Hermitian $n \times n$ matrix, the spectrum $\sigma(A) \subset \mathbb{C}$ is included in the real axis. That is, all eigenvalues are real.*

Exercise 4.2. *If A is a positive-definite $n \times n$ matrix, the spectrum $\sigma(A) \subset \mathbb{C}$ is included in right half real line in the $\mathbb{C}$. That is, all eigenvalues are positive real.*

Exercise 4.3. *Show that A is a positive-definite $n \times n$ matrix if and only if $x^*Ax > 0$ for all $x \neq 0$, $x \in \mathbb{C}^n$.*

Exercise 4.4. *Show if A is an orthogonal matrix, the spectrum $\sigma(A) \subset \mathbb{C}$ is included in the unit circle in the $\mathbb{C}$.*

Theorem 4.2 (Shur Decomposition Theorem)**.** *All $n \times n$ matrix A is unitarily equivalent to an upper triangular matrix such that there is a unitary matrix U such that*

$$U^*AU = \begin{pmatrix} \lambda_1 & * & \cdots & * \\ & \lambda_2 & \ddots & \vdots \\ & & \ddots & * \\ 0 & & & \lambda_n \end{pmatrix} \tag{4.2}$$

where $\lambda_j, j = 1 \cdots, n$ are eigenvalues of A.

Definition 4.1. *An $n \times n$ matrix A over F is said to be diagonalizable, if there exist an $n \times n$ diagonal matrix $\Lambda = \text{diag}\{\lambda_1, \cdots, \lambda_n\}$ over F and an $n \times n$ matrix X over F such that*

$$X^{-1}AX = \Lambda.$$

In this case, $AX = X\Lambda$. Denoting by $x^1, \cdots, x^n$ be the column vectors of X, we have

$$Ax^j = \lambda_j x^j, j = 1, \cdots, n.$$

That is, for $j = 1, \cdots, n$, the j-th column of X is an eigenvector of A corresponding to the eigenvalue λ_j which is the j-th diagonal entry of Λ.

Theorem 4.3. *An $n \times n$ matrix A is diagonalizable if and only if A has n linearly independent eigenvectors.*

Proof. A is diagonalizable if and only if there exist an invertible matrix X and a diagonal matrix Λ with $X^{-1}AX = \Lambda$. This is equivalent to the n column vectors of X, which are eigenvectors of A, are linearly independent. □

Theorem 4.4. *An $n \times n$ matrix A is diagonalizable if and only if for all eigenvalues λ of A, the geometric multiplicity of λ = the algebraic multiplicity of λ.*

Proof. Notice that for all eigenvalues λ of A, the geometric multiplicity of $\lambda =$ the algebraic multiplicity of λ if and only if there exist n linearly independent eigenvecots. Then the theorem follows from the above theorem. □

Theorem 4.5. *Hermitian matrices are unitarily diagonalizable. That is, for any $n \times n$ Hermitian matrix A, there is a unitary matrix U such that*

$$U^*AU = \text{diag}[\lambda_1, \lambda_2, \cdots, \lambda_n], \tag{4.3}$$

where $\lambda_j, j = 1 \cdots, n$ are real eigenvalues of A. Moreover, the jth column of U is an eigenvector associated with λ_j.

Proof. Due to Theorem 4.2, there is a unitary matrix U such that (4.2) holds. Since A is Hermitian, taking the complex conjugates of the both sides in (4.2) shows that the left side in (4.2) is symmetric with real diagonal. Thus the right side in (4.2) is symmetric with real diagonal. This proves the representation (4.3). Moreover, for $j = 1, \cdots, n$. let $\xi^{(j)}$ denote an eigenvector with $\|\xi^{(j)}\|_2 = 1$ associated with λ_j. Then set $U = [\xi^{(1)}\ \xi^{(2)}\ \cdots\ \xi^{(n)}]$. It then follows that

$$AU = [A\xi^{(1)}\ A\xi^{(2)}\ \cdots\ A\xi^{(n)}] = [\lambda_1\xi^{(1)}\ \lambda_2\xi^{(2)}\ \cdots\ \lambda_n\xi^{(n)}] = U\text{diag}[\lambda_1, \lambda_2, \cdots, \lambda_n],$$

Since U is unitary, one gets

$$U^*AU = \text{diag}[\lambda_1, \lambda_2, \cdots, \lambda_n],$$

□

Definition 4.2. *A matrix A is called normal if*

$$A^*A = AA^* (\textit{not necessarily} = I). \tag{4.4}$$

More generally, we have

Theorem 4.6. *Normal matrices are unitarily diagonalizable. That is, for any $n \times n$ normal matrix A, there is a unitary matrix U such that*

$$U^* A U = \mathrm{diag}(\lambda_1, \lambda_2, \cdots, \lambda_n),$$

where $\lambda_j, j = 1 \cdots, n$ are (not necessarily real) eigenvalues of A. Moreover, the jth column of U is an eigenvector associated with λ_j.

Proof. □

Proposition 4.2. *1. A square matrix is invertible if and only if its column vectors are linearly independent. Hence, if A is diagonalizable such that $X^{-1}AX = \Lambda$, then the eigenvectors $x^j, j = 1, \cdots, n$, are linearly independent.*

2. Involutive matrices are diagonalizable. That is, if $A^2 = I$, then A is diagonalizable.

3. If A has an eigenvalue λ whose algebraic multiplicity is strictly bigger than the geometric multiplicitly, then A is not diagonalizable.

4. Real symmetric matrices are diagonalizable by orthogonal matrices.

5. If A has an eigenvalue which is not included in F, then A is not diagonalizable over F.

Theorem 4.7. *For any consistent matrix norm $\|\cdot\|$,*

$$\rho(A) \le \|A\|, \qquad \forall A \in \mathcal{M}(n,n) \tag{4.5}$$

Proof. Let $Ax = \lambda x, \quad x \neq 0$. Then $|\lambda|\|x\| = \|\lambda x\| = \|Ax\| \le \|A\|\|x\|$. Therefore, $|\lambda| \le \|A\|$ and $\rho(A) \le \|A\|$. □

Lemma 4.1. *Let S be a nonsingular matrix. Then $x \mapsto \|Sx\|_p$ defines a new vector norm $\|\cdot\|_{(p)}$, and induces a new matrix norm*

$$\|A\|_{(p)} = \left\|SAS^{-1}\right\|_p$$

Proof. Since $x \neq 0$ implies $Sx \neq 0$, obviously $x \mapsto \|Sx\|$ defines a new norm. Next,

$$\|A\|_{(p)} = \max_{x\neq 0} \frac{\|Ax\|_{(p)}}{\|x\|_{(p)}} = \max_{x\neq 0} \frac{\|SAx\|_p}{\|Sx\|_p} = \max_{y\neq 0} \frac{\left\|SAS^{-1}y\right\|_p}{\|y\|_p} = \left\|SAS^{-1}\right\|_p$$

This proves the assertion. □

Theorem 4.8. *Let $A \in \mathcal{M}(n,n)$ and $\varepsilon > 0$ be given. Then there exists a consistent matrix norm, $\|\cdot\| = \|\cdot\|_{A,\varepsilon}$ such that $\rho(A) \leq \|A\|_{A,\varepsilon} \leq \rho(A) + \varepsilon$.*

Proof. Let $TAT^{-1} = J$ be the Jordan canonical form where J is a block diagonal matrix where blocks are of the form,

$$J = \begin{pmatrix} J_1 & & & \\ & J_2 & & \\ & & \ddots & \\ & & & J_m \end{pmatrix}, \quad \text{where} \quad J_k = \begin{pmatrix} \lambda_k & 1 & & 0 \\ & \ddots & \ddots & \\ & & \lambda_k & 1 \\ 0 & & & \lambda_k \end{pmatrix}.$$

Let $D_\varepsilon = \text{diag}\{1, \varepsilon, \varepsilon^2, \cdots, \varepsilon^{n-1}\}$. Consider then the linear transformation,

$$J \longrightarrow D_\varepsilon^{-1} J D_\varepsilon = \begin{pmatrix} J_1' & & & \\ & J_2' & & \\ & & \ddots & \\ & & & J_m' \end{pmatrix}, \quad \text{with} \quad J_k' = \begin{pmatrix} \lambda_k & \varepsilon & & 0 \\ & \ddots & \ddots & \\ & & \lambda_k & \varepsilon \\ 0 & & & \lambda_k \end{pmatrix}.$$

Then,

$$\left\|D_\varepsilon^{-1} J D_\varepsilon\right\|_\infty = \left\|D_e^{-1} T A T^{-1} D_\varepsilon\right\|_\infty = \rho(A) + \varepsilon$$

Since D_ε and T are nonsingular, $S = {D_\varepsilon}^{-1} T$ induces a new vector norm and matrix norm as in Lemma 4.1 We thus choose the vector norm $\|x\|_{(\infty)} = \|Sx\|_\infty$ and matrix norm $\|A\|_{A,\varepsilon} := \left\|SAS^{-1}\right\|_\infty$. Then

$$\|A\|_{A,\varepsilon} = \left\|D_\varepsilon^{-1} T A (D_\varepsilon^{-1} T)^{-1}\right\|_\infty \leq \rho(A) + \varepsilon.$$

Observe that the above inequality can be strict when all the Jordan blocks of 1×1 matrices. □

Theorem 4.9. *Let $A \in \mathcal{M}(n,n)$, and $\|\cdot\|$ be a consistent norm. Then*

$$\lim_{k\to\infty} \left\|A^k\right\|^{1/k} = \frac{1}{\rho(A)} \tag{4.6}$$

Theorem 4.10. *Let $A \in \mathcal{M}(n,n)$, Then*

$$\lim_{k\to\infty} A^k = 0 \qquad \Longleftrightarrow \qquad \rho(A) < 1 \tag{4.7}$$

Proof. Assume $\rho(A) < 1$. Then by Theorem 4.8, there exists a consistent matrix norm $\|\cdot\|$ such that $\|A\| \le \rho(A) + \varepsilon < 1$. Then $\left\|A^k\right\| \le \|A\|^k \le (\rho(A)+\varepsilon)^k < 1$. Thus $\lim_{k\to\infty}\left\|A^k\right\| = 0$. and then $\left\|\lim_{k\to\infty} A^k\right\| = 0$, therefore $\lim_{k\to\infty} A^k = 0$. Next, assume that $\lim_{k\to\infty} A^k = 0$. Let $Ax = \lambda x, \quad x \neq 0$, then $A^k x = \lambda^k x$. And thus $0 = (\lim_{k\to\infty} A^k)x = \lim_{k\to\infty}\lambda^k x \Longrightarrow |\lambda| < 1$. Consequently, $\rho(A) < 1$. □

Theorem 4.11. *In the case of Theorem 4.10, i.e. $\rho(A) < 1$.*

$$\frac{1}{1+\|A\|} \le \left\|(I-A)^{-1}\right\| \le \frac{1}{1-\|A\|} \tag{4.8}$$

Moreover,

$$\therefore (I-A)^{-1} = \sum_{k=0}^{\infty} A^k. \tag{4.9}$$

Proof. The fact that $I - A$ is invertible and the second inequality (4.8) follows from Theorem 4.7 and Theorem 3.2. Next notice that $1 = \|I\| = \left\|(I-A)(I-A)^{-1}\right\| \le \|I-A\|\left\|(I-A)^{-1}\right\| \le (1+\|A\|)\left\|(I-A)^{-1}\right\|$, which implies the first inequality in (4.8)

$$\frac{1}{1+\|A\|} \le \left\|(I-A)^{-1}\right\|. \tag{4.10}$$

In order to derive (4.9), observe that $(I-A)(I+A+A^2+\cdots A^k) = I - A^{k+1}$, and hence Theorem 4.10 shows that $(I-A)\sum_{k=0}^{\infty} A^k = I$ and that $(\sum_{k=0}^{\infty} A^k)(I-A) = I$. Therefore (4.9) follows. □

Denote

$$\lambda_{\min}(A) = \min_{\lambda\in\sigma(A)} |\lambda|, \quad \lambda_{\max}(A) = \max_{\lambda\in\sigma(A)} |\lambda|$$

Theorem 4.12. *For $A \in \mathcal{M}(n,n)$, let*

$$H = \frac{A+A^*}{2} \qquad \textit{the Hermitian part of } A \tag{4.11}$$

$$S = \frac{A-A^*}{2i} \qquad \textit{the skew-symetric part of } A \tag{4.12}$$

then for every $\lambda \in \sigma(A)$,

$$\lambda_{\min}(H) \leq |\operatorname{Re}(\lambda)| \leq \lambda_{\max}(H) \tag{4.13a}$$
$$\lambda_{\min}(S) \leq |\operatorname{Im}(\lambda)| \leq \lambda_{\max}(S) \tag{4.13b}$$

Proof. Let $Ax = \lambda x$, $\|x\|_2 = 1$. Then $\lambda = x^*Ax = x^*\frac{A+A^*}{2}x + x^*\frac{A-A^*}{2}x = x^*Hx + ix^*Sx$. Since H and S are Hermitian, they are unitarily similar to real diagonal matrices.

$$H = M\Sigma M^*, \tag{4.14a}$$
$$S = N\Sigma' N^*, \tag{4.14b}$$

for some orthogonal matrices M and N, and Σ and Σ' are real diagonal matrices with diagonals being the eigenvalues of H and S, respectively. $\lambda = x^*M\Sigma M^*x + ix^*N\Sigma' N^*x = y^*\Sigma y + iz^*\Sigma' z$. Since $y^*y = x^*MM^*x = x^*x = 1$ and $z^*z = x^*NN^*x = x^*x = 1$.

$$\operatorname{Re}(\lambda) = y^*\Sigma y \tag{4.15a}$$
$$\operatorname{Im}(\lambda) = z^*\Sigma' z. \tag{4.15b}$$

This shows Equations 4.13a and 4.13b. □

4.1 Gerschgorin's Circles Theorems

Set

$$R_j := \{z \in \mathbb{C} : |z - a_{jj}| \leq \sum_{\substack{k=1 \\ k\neq j}}^{n} |a_{jk}|\}, \tag{4.16}$$

$$C_j := \{z \in \mathbb{C} : |z - a_{jj}| \leq \sum_{k=1,\ k\neq j}^{n} |a_{kj}|\}. \tag{4.17}$$

Theorem 4.13. $\sigma(A) \subset \bigcup_{j=1}^{n} R_j$.

Proof. Let $D = \operatorname{diag}\{a_{11}, a_{22}, \ldots, a_{nn}\}$ and write $A = D + E$. For $\lambda \in \sigma(A)$, if $\lambda = a_{jj}$ for some j, then nothing to prove. Therefore, assume that $\lambda \neq a_{jj},\ \forall j$.

Next set $B_\lambda := A - \lambda I = (D - \lambda I) + E$. Since $\lambda \in \sigma(A)$, there exists $x \neq 0$ such that $B_\lambda x = 0$ or $(D-\lambda I)x + Ex = 0$. Since $x = -(D-\lambda I)^{-1}Ex$, $\|x\|_\infty \leq$

$\left\|(D-\lambda I)^{-1}E\right\|_\infty \|x\|_\infty$. Dividing by $\|x\|_\infty$ and recalling the definition of $\|\cdot\|_\infty$, one has

$$1 \le \left\|(D-\lambda)^{-1}E\right\|_\infty \overset{\text{for some } j}{=} \sum_{\substack{k=1\\k\neq j}}^{n} \left|\frac{a_{jk}}{a_{jj}-\lambda}\right|.$$

Thus,

$$|a_{jj}-\lambda| \le \sum_{\substack{k=1\\k\neq j}}^{n} |a_{jk}|,$$

which completes the theorem. □

Theorem 4.14 (1st Gerschgorin's Circles Theorem)**.**

$$\sigma(A) \subset \left(\bigcup R_j\right) \cap \left(\bigcup C_j\right) \tag{4.18}$$

Proof. The same as Theorem 4.13 by using $\|\cdot\|_1$ instead of $\|\cdot\|_\infty$. □

Theorem 4.15 (2nd Gerschgorin's Circles Theorem)**.** *Suppose that*

$$\left(\bigcup_{j=1}^{m} R_{i_j}\right) \cap \left(\bigcup_{j=m+1}^{n} R_{i_j}\right) = \phi.$$

Then $\bigcup_{j=1}^{m} R_{i_j}$ *contains exactly* m *eigenvalues of* A*, with each eigenvalue being counted according to its algebraic multiplicity. The remain* $n-m$ *eigenvalues are in* $\bigcup_{j=m+1}^{n} R_{i_j}$*. Here,* $\{i_1, i_2, \cdots, i_n\} = \{1, 2, \cdots, n\}$.

Proof. Without loss of generality, we assume that $i_j = j, j = 1, 2, \cdots, n$. Let $D = \text{diag}\{a_{11}, a_{22}, \ldots, a_{nn}\}$ and write $A = D + E$. For $t \in [0,1]$, set $A_t = D + tE$. Then $A_0 = D$, $A_1 = A$. For $t \in [0,1]$, set

$$R_j(t) := \{z \in \mathbb{C} : |z - a_{jj}| \le \sum_{\substack{k=1\\k\neq j}}^{n} |ta_{jk}|\} \tag{4.19}$$

Let $\lambda_j(t)$ denote eigenvalues of A_t. For $t = 0$, the exact eigenvalues $\lambda_j(0)$ are identical to the diagonal entries a_{jj}. There exist exactly m eigenvalues of A_0 in $\bigcup_{j=1}^{m} R_j$ and $n-m$ eigenvalues in $\bigcup_{j=m+1}^{n} R_j$ (taking into account

the multiplicities.) Notice that the radii $\sum_{\substack{k=1\\k\neq j}}^{n} |ta_{jk}|$ of $R_j(t)$ is increasing in t and thus,

$$\left(\bigcup_{j=1}^{m} R_j(t)\right) \cap \left(\bigcup_{j=m+1}^{n} R_j(t)\right) = \phi \text{ for all } t \in [0,1].$$

Since the eigenvalues of A_t are continuous functions of t, by the first Theorem of Gerschgorin's circles, there will be m eigenvalues in $\bigcup_{j=1}^{m} R_j(t)$ and the other $n-m$ eigenvalues in $\bigcup_{j=m+1}^{n} R_j(t)$ for all $t \in [0,1]$. □

4.1.1 Example of 1D elliptic problem: this subsection is completely revised

Consider the 1D elliptic problem

$$-u''(x) = f(x), x \in (0,1); \quad u(0) = u(1) = 0, \tag{4.20}$$

from which the standard finite difference scheme with mesh size $h = \frac{1}{n}$ leads to the following $(n-1) \times (n-1)$ matrix

$$A_h = \frac{1}{h^2} \begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & & \ddots & & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{pmatrix}.$$

Let us find $\sigma(A_h)$ with eigenvectors.

First, possible Gerschgorin's circles are given by

$$\begin{aligned} R_1 &= R_{n-1} = \{z \in \mathbb{C} : |z - \frac{2}{h^2}| \leq \frac{1}{h^2}\}, \\ R_2 &= \cdots = R_{n-2} = \{z \in \mathbb{C} : |z - \frac{2}{h^2}| \leq \frac{2}{h^2}\}. \end{aligned}$$

By Gerschgorin's circles theorem, all eigenvalues are included in the disc $\{z \in \mathbb{C} : |z - \frac{2}{h^2}| \leq \frac{2}{h^2}\}$. Hence the least upper bound $|\lambda|$ for $\lambda \in \sigma(A_h)$ is less than or equal to $\frac{4}{h^2}$, while the greatest lower bound $|\lambda|$ for $\lambda \in \sigma(A_h)$ is larger than or may be equal to 0.

We proceed to investigate in more details. First, the eigenvalues and their associated eigenfunctions of the second-order elliptic operator $\mathcal{L} = -\frac{d^2}{dx^2}$:

$L^2(0,1) \to L^2(0,1)$, with the domain of $\mathcal{L}$ is given by

$$H^2(0,1) \cap H_0^1(0,1) = \{v \in L^2(0,1) : \ v', v'' \in L^2(0,1); v(0) = v(1) = 0\},$$

are given by

$$\lambda = (k\pi)^2, \quad \phi(x) = \sin(k\pi x), \quad \text{for } k = 1, 2, \cdots . \tag{4.21}$$

Notice that among these continuous eigenfunctions, there are only $n-1$ number of them which can be approximated by the n-uniform mesh. They are explicitly given by

$$x^{(k)} = \left(\sin\frac{k\pi}{n}, \sin\frac{2k\pi}{n}, \cdots, \sin\frac{(n-1)k\pi}{n}\right)^t, \quad k = 1, \cdots, n-1. \tag{4.22}$$

If A_h acts on $x^{(k)}$, the j-th component of $A_h x^{(k)}$ can be calculated as follows:

$$\begin{aligned} & -\frac{1}{h^2}\sin\frac{(j-1)k\pi}{n} + \frac{2}{h^2}\sin\frac{jk\pi}{n} - \frac{1}{h^2}\sin\frac{(j+1)k\pi}{n} \\ = \ & -\frac{2}{h^2}\sin\frac{jk\pi}{n}\cos\frac{k\pi}{n} + \frac{2}{h^2}\sin\frac{jk\pi}{n} = \frac{2}{h^2}\left(1-\cos\frac{k\pi}{n}\right)\sin\frac{jk\pi}{n}, \end{aligned}$$

owing to the elementary trigonometric formula $\sin(\theta_1+\theta_2) + \sin(\theta_1-\theta_2) = 2\sin\theta_1\cos\theta_2$. We therefore observe that (4.22) are eigenvectors of A_h associated with the eigenvalue $\frac{2}{h^2}\left(1-\cos\frac{k\pi}{n}\right)$ for $k = 1, \cdots, n-1$.

Remark 4.1. *Notice that the discrete eigenvectors given by* (4.22) *are nothing but the restrictions of the first* $n-1$ *continuous eigenvectors given by* (4.21) *at the interior nodes* $x_j = \frac{j}{n}, j = 1, \cdots, n-1$. *However, the discrete eigenvalues* $\frac{2}{h^2}\left(1-\cos\frac{k\pi}{n}\right)$ *are approximated from the continuous eigenvalues* $(k\pi)^2$ *since*

$$\frac{2}{h^2}\left(1-\cos\frac{k\pi}{n}\right) = (k\pi)^2 + \mathcal{O}(h^2)$$

due to the expansion:

$$1-\cos\frac{k\pi}{n} = \frac{1}{2!}\left(\frac{k\pi}{n}\right)^2 - \frac{1}{4!}\left(\frac{k\pi}{n}\right)^4 + \frac{1}{6!}\left(\frac{k\pi}{n}\right)^6 - \cdots .$$

Notice that the eigenvectors satisfy the orthogonality relationship:

$$(x^{(k)}, x^{(l)}) = \delta_{kl}\frac{n}{2},$$

where δ_{kl} denotes the Kronecker delta. Observe that we have the spectral radius

$$\rho(A_h) = \lambda^{(n-1)} = \frac{2}{h^2}(1 - \cos\frac{(n-1)\pi}{n}),$$

from which we have

$$\begin{aligned} \|A_h\|_2 &= \max_{0\neq x\in\mathbb{R}^{n-1}} \frac{x^T A_h x}{x^T x} \\ &= \max_{x\in\mathbb{R}^{n-1},\|x\|_2=1} x^T A_h x \\ &= {x^{(n-1)}}^T A_h x^{(n-1)} = \lambda^{(n-1)}. \end{aligned}$$

In the meanwhile, since $A_h^{-1}x^{(k)} = \frac{1}{\lambda^{(k)}}x^{(k)}$,

$$\begin{aligned} \|A_h^{-1}\|_2 &= \max_{0\neq x\in\mathbb{R}^{n-1}} \frac{x^T A_h^{-1} x}{x^T x} \\ &= \max_{x\in\mathbb{R}^{n-1},\|x\|_2=1} x^T A_h^{-1} x \\ &= {x^{(1)}}^T A_h^{-1} x^{(1)} = \frac{1}{\lambda^{(1)}}. \end{aligned}$$

Thus the condition number $\kappa(A_h)$ (using $\|\cdot\|_2$ norms) can be estimated as follows: for large n,

$$\begin{aligned} \kappa(A_h) &= \|A_h\|_2 \, \|A_h^{-1}\|_2 = \frac{\frac{2}{h^2}(1 - \cos\frac{(n-1)\pi}{n})}{\frac{2}{h^2}(1 - \cos\frac{\pi}{n})} \\ &\approx \frac{4}{\frac{\pi^2}{n^2}} = \frac{4}{\pi^2}n^2 = \mathcal{O}(n^2). \end{aligned}$$

Exercise 4.5. *Show that*

$$\|A_h\|_2 = \max_{0\neq x\in\mathbb{R}^{n-1}} \frac{x^T A_h x}{x^T x}$$

if A_h is a symmetric positive-definite matrix.

Exercise 4.6. *Show that*

$$\|A_h^{-1}\|_2 = \frac{1}{\min_{0\neq x\in\mathbb{R}^{n-1}} \frac{x^T A_h x}{x^T x}}$$

if A_h is a symmetric positive-definite matrix.

4.1.2 Example of 2D elliptic boundary value problem: this subsection is completely revised

The goal of this subsection is to show that for the two-dimensional elliptic problem 2.5, the matrix A_h has the condition number $\mathcal{O}(n^2)$ if the $n \times n$ uniform meshes are adopted.

Let $[0.1]^2$ be discretized uniformly with meshes $(x_j, y_k) = (jh, kh), j, k = 0, \cdots, n$, with $h = \frac{1}{n}$ and recall that the boundary values are prescribed at the points if $j = 0, j = n, k = 0$, or $k = n$.

Assume that the prescribed boundary values are zero. Denote by u_{jk} the approximate solution at (x_j, y_k). Write the $(n-1)^2$-dimensional unkonw vector with two dimensional indices as follows:

$$\mathbf{u} = \left(u_{11}u_{21}\cdots u_{n-1,1}u_{12}u_{22}\cdots u_{n-1,2}\cdots u_{1,n-1}u_{2,n-1}\cdots u_{n-1,n-1}\right)^t.$$

For the sake of convenience, we will call the u_{jk} the (j,k)-th component. Then designate by A_h the $(n-1)^2 \times (n-1)^2$ penta-diagonal matrix which discretize the two-dimensional $-\Delta$.

Exercise 4.7. *Show that* $(\ell^2 + m^2)\pi^2, \ell, m = 1, 2, \cdots,$ *are eigenvalues corresponding to the eigenfunction* $\sin(\ell\pi x)\sin(m\pi x)$ *of* $-\Delta$.

Exercise 4.8. *Show that the* (j,k)*-th component of* $A_h\mathbf{u}$ *is equal to*

$$\frac{4u_{jk} - u_{j-1,k} - u_{j+1,k} - u_{j,k-1} - u_{j,k+1}}{h^2}.$$

Exercise 4.9. *Using the above exercise, show that for* $\ell, m = 1, 2, \cdots, (n-1)$, *the* $(n-1)^2$*-dimensional vector* $\mathbf{u}$ *whose* (j,k)*-th component is given by*

$$\sin(\ell\pi x_j)\sin(m\pi y_k)$$

is an eigenvector for A_h. *Show that the eigenvalues corresponding to these eigenvectors are* $\frac{2}{h^2}\left(2 - \cos\frac{k\pi}{n} - \cos\frac{\ell\pi}{n}\right)$.

Exercise 4.10. *Show that the condition number for* A_h *is*

$$\frac{1 - \cos\frac{(n-1)\pi}{n}}{1 - \cos\frac{\pi}{n}} \approx \frac{4}{\pi^2}n^2.$$

4.1.3 Example of 1D elliptic problem with Neumann BC

Consider the 1D elliptic problem

$$-u''(x) = f(x), x \in (0,1); \quad u'(0) = u'(1) = 0. \tag{4.23}$$

Consider again the standard finite difference scheme with mesh size $h = \frac{1}{n}$ with nodes $x_j, j = 0, \cdots, n$. Then for each $j = 1, \cdots, n-1$, we have

$$\frac{-u_h(x_{j-1}) + 2u_h(x_j) - u_h(x_{j+1})}{h^2} = f(x_j).$$

with

$$\frac{-u_h(x_{j-1}) + u_h(x_j)}{h^2} = 0, \quad j = 1, n.$$

Eliminating the variables $u_h(x_0)$ and $u_h(x_n)$ leads to the following $(n-1) \times (n-1)$ matrix

$$A_h = \frac{1}{h^2} \begin{pmatrix} 1 & -1 & & & \\ -1 & 2 & -1 & & \\ & & \ddots & & \\ & & -1 & 2 & -1 \\ & & & -1 & 1 \end{pmatrix}.$$

Notice that A_h is a symmetric positive-definite matrix and 0 is an eigenvalue of A_h, and all the other eigenvalues are positive. One can repeat to find $\sigma(A_h)$. as above.

4.1.4 Example of 1D elliptic problem with periodicity

Consider the 1D elliptic problem

$$-u''(x) = f(x), x \in (0,1); \quad u(x) = u(x+1)q \quad \forall x. \tag{4.24}$$

Consider again the standard finite difference scheme with mesh size $h = \frac{1}{n}$ with nodes $x_j, j = 0, \cdots, n$. Then for each $j = 1, \cdots, n-1$, we have

$$\frac{-u_h(x_{j-1}) + 2u_h(x_j) - u_h(x_{j+1})}{h^2} = f(x_j).$$

with

$$\frac{u_h(x_j) - u_h(x_{j+n})}{h^2} = 0, \quad j = 1, n.$$

Eliminating the variables $u_h(x_0)$ and $u_h(x_n)$ leads to the following $(n-1) \times (n-1)$ matrix

$$A_h = \frac{1}{h^2} \begin{pmatrix} 2 & -1 & & & -1 \\ -1 & 2 & -1 & & \\ & & \ddots & & \\ & & -1 & 2 & -1 \\ -1 & & & -2 & 1 \end{pmatrix}.$$

Notice that A_h is a symmetric positive-definite matrix and 0 is an eigenvalue of A_h, and all the other eigenvalues are positive. One can repeat to find $\sigma(A_h)$. as above.

4.2 The power method (to compute eigenvalues and eigenvectors.)

Let $A \in M(n,n)$ be diagonalizable such that $X^{-1}AX = \Lambda$, $X = [x_1, \ldots, x_n]$, $\Lambda = \text{diag}\{\lambda_1, \ldots, \lambda_n\}$, $|\lambda_1| > |\lambda_2| \geq \cdots \geq |\lambda_n| \geq 0$. Assume that λ_1 has algebraic multiplicity 1. The power method approximates λ_1 and its eigenvector x_1.

ALGORITHM 4.1.

1 choose an initial guess $q^{(0)} \in \mathbb{C}^n$, $\left\|q^{(0)}\right\|_2 = 1$.

2 do j=1, until converge

3 $\quad z^{(j)} = Aq^{(j-1)}$

4 $\quad q^{(j)} = \frac{z^{(j)}}{\left\|z^{(j)}\right\|_2}$ $\qquad j \to \infty,\ q^{(j)} \to x_1$

5 $\quad \nu^{(j)} = (q^{(j)})^* A q^{(j)}$ $\qquad j \to \infty,\ \nu^{(j)} \to \lambda_1$

6 enddo

The following is an example of power method program.

F90 Program 4.1: Gaussian elimination without pivoting

```
1  program powermethod
2  !Power method to approximate the maximum eigenvalue and vector
3  !by D. Sheen http://www.nasc.snu.ac.kr
4    implicit none
5
6    real(8), allocatable:: a(:,:), q(:), prev_q(:), z(:), eig_vec(:)
```

```
7   real(8):: h,l2norm, diff_vec, eps, lambda, pi
8   integer(4):: n, j, iter, max_iter
9
10  write(6,*) "What are the dimension of the matrix n and max_iter?"
11  read(5,*) n, max_iter
12
13  allocate(a(1:n-1,1:n-1), q(1:n-1), prev_q(1:n-1), z(1:n-1), eig_vec(1:n-1) )
14  h = 1.d0/real(n)
15  pi = atan(1.d0)*4.d0
16
17  do j = 1, n-1
18     eig_vec(j) = sin(j*pi*(n-1)/n)
19  end do
20  eig_vec = eig_vec/sqrt(dot_product(eig_vec,eig_vec))
21
22  do j = 1, n-1
23     a(j,j) = 2.d0/(h*h)
24  end do
25  do j = 1, n-2
26     a(j,j+1) = -1.d0/(h*h)
27     a(j+1,j) = -1.d0/(h*h)
28  end do
29
30  iter = 0;   eps = 1.0d-15
31
32  q = 0.d0; q(5) = 1.d0
33  q = q/sqrt(dot_product(q,q))
34  prev_q = q
35  print*, "Initial approximation of q = ", q
36
37  do iter = 1,  max_iter
38    q = matmul(A, q)
39    q = q/sqrt(dot_product(q,q))
40    lambda = dot_product(q, matmul(A,q))
41
42    diff_vec = sqrt(dot_product(q - prev_q, q - prev_q) )
43    prev_q = q ! save current vector to use at the next iteration
44
45    if(diff_vec < eps ) then
46       write(6,91) iter, lambda, diff_vec
47       print*, "Coverged at iter = ", iter
48       print*, "Exact maximum eigenvalue = ", 2.d0*(1. - cos((n-1)*pi/n))/(h*h)
49       print*, "The exact eigenvector is as follows "
50       do j = 1, n/5+1
51          write(6,92) eig_vec((j-1)*5+1: min(j*5,n-1))
52       end do
53       print*, "Computed maximum eigenvalue = ", lambda
54       print*, "The computed eigenvector is as follows "
55       do j = 1, n/5+1
56          write(6,92) q((j-1)*5+1: min(j*5,n-1))
57       end do
58       print*, "|| Exact eig_vec - comput_vec ||_2 = ", &
59            sqrt(dot_product(eig_vec-q,eig_vec-q))
60       exit
61    endif
62    write(6,91) iter, lambda, diff_vec
```

```
63 !     write(6,93) iter, lambda, q, diff_vec
64   end do
65
66 91     format("iter=",i6,";  lambda=",g20.10,";  Diff_vec=",g12.3)
67 92     format(5g12.3)
68 !93     format("iter=",i6,";  lambda=",g15.3,";  q=",10g15.3, ";  Diff_vec=",g12.3)
69 end program powermethod
```

Exercise 4.11. *Let x and λ be an eigenvector and its associated eigenvalue of a nonsingular $n \times n$ matrix A. If T is a nonsingular $n \times n$ matrix, then $y = Tx$ and λ are an eigenvector and its associated eigenvalue of TAT^{-1}*

4.2.1 Convergence analysis

Note that

$$q^{(j)} = \frac{A^j q^{(0)}}{\left\| A^j q^{(0)} \right\|_2}. \tag{4.25}$$

Since A is diagonalizable, we have $X^{-1}AX = \Lambda$, where the components of $\Lambda = \text{diag}\{\lambda_1, \cdots, \lambda_n\}$ and $x_k, k = 1, \cdots, n$, are associated eigenpairs of A such that $Ax_k = \lambda_k x_k$, $\forall k$. Moreover, $x_k, k = 1, \cdots, n$, are linearly independent. Let $q^{(0)} = \sum_{k=1}^n \alpha_k x_k$, $\alpha_k \in \mathbb{C}$. Then

$$\begin{aligned} A^j q^{(0)} &= \textstyle\sum_{k=1}^n \alpha_k \lambda_k^j x_k \\ &= \alpha_1 \lambda_1^j \left[x_1 + \textstyle\sum_{k=2}^n \frac{\alpha_k}{\alpha_1} \left(\frac{\lambda_k}{\lambda_1} \right)^j x_k \right] \\ &=: \alpha_1 \lambda_1^j \tilde{q}^{(j)}. \end{aligned} \tag{4.26}$$

Set

$$q^{(j)} = \frac{\alpha_1 \lambda_1^j \tilde{q}^{(j)}}{\|\alpha_1 \lambda_1^j \tilde{q}^{(j)}\|} = \frac{\alpha_1 \lambda_1^j}{|\alpha_1 \lambda_1^j|} \frac{\tilde{q}^{(j)}}{\|\tilde{q}^{(j)}\|}.$$

Notice that $\tilde{q}^{(j)}$ converges to x_1 since $\left| \frac{\lambda_k}{\lambda_1} \right| < 1$ for $k = 2, \cdots, n$. Hence, $q^{(j)}$ converges to $\text{Span}\{x_1\}$. Recalling that $A^j q^{(0)} = \alpha_1 \lambda_1^j \tilde{q}^{(j)}$ and $q^{(j)} = \frac{\tilde{q}^{(j)}}{\left\| \tilde{q}^{(j)} \right\|_2}$, we have $\tilde{q}^{(j)} = \beta_j q^{(j)}$ with $\beta_j = \frac{\left\| A^j q^{(0)} \right\|_2}{\alpha_1 \lambda_1^j}$. From the above argument it follows that

$$\nu^{(j)} = \frac{(q^{(j)})^* A q^{(j)}}{(q^{(j)})^* q^{(j)}} = \frac{(\tilde{q}^{(j)})^* A \tilde{q}^{(j)}}{(\tilde{q}^{(j)})^* \tilde{q}^{(j)}} \to \frac{x_1^* A x_1}{x_1^* x_1} = \lambda_1.$$

Exercise 4.12.

$$A = \begin{pmatrix} -261. & 209. & -49. \\ -530. & 422. & -98. \\ -800. & 631. & -144. \end{pmatrix}, \quad v^{(0)} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$

Apply the power method to find the largest eigenvalue and eigenvector.

4.2.2 Inverse iteration

Given $\mu \notin \sigma(A)$, find an approximation of $\lambda \in \sigma(A)$ that is closest to μ, "μ" is called a *shift*.

Idea : Apply the power method to $M_\mu^{-1} = (A - \mu I)^{-1}$.

Note that $\sigma(M_\mu^{-1}) = \{(\lambda_j - \mu)^{-1} : \lambda_j \in \sigma(A)\}$. Indeed,

$$Ax = \lambda x \Leftrightarrow (A - \mu I)x = (\lambda - \mu)x \Leftrightarrow (\lambda - \mu)^{-1}x = (A - \mu I)^{-1}x = M_\mu^{-1}x.$$

Assume that $\exists$ an integer m such that $|\lambda_m - \mu| < |\lambda_j - \mu|$, for all $\lambda_j \in \sigma(A), \lambda_j \neq \lambda_m$. Then $\xi_m = (\lambda_m - \mu)^{-1}$ is the eigenvalue of M_μ^{-1} with largest modulus. If $\mu = 0$, λ_m is the eigenvalues with smallest modulus. $\lambda_m = \frac{1}{\xi_m} + \mu$.

ALGORITHM 4.2.

1 choose an initial guess $q^{(0)} \in \mathbb{C}^n$, $\left\|q^{(0)}\right\|_2 = 1$.

2 do j=1, max_j

3 Solve $(A - \mu I)z^{(j)} = q^{(j-1)}$ for $z^{(j)}$

4 $q^{(j)} = z^{(j)} / \left\|z^{(j)}\right\|_2$

5 $\sigma^{(j-1)} = (q^{(j-1)})^* z^{(j)}$

6 enddo

In implementing the line 3 of the above algorithm, one can use any direct method to solve the matrix $(A - \mu I)z^{(j)} = q^{(j-1)}$.

Notice that

$$\sigma^{(j-1)} = (q^{(j-1)})^*(A - \mu I)^{-1}q^{(j-1)} = {q^{(j-1)}}^* z^{(j)}.$$

Exercise 4.13.

$$A = \begin{pmatrix} -261. & 209. & -49. \\ -530. & 422. & -98. \\ -800. & 631. & -144. \end{pmatrix}, \quad v^{(0)} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$

Apply the power and inverse iteration methods to find all the eigenvalues and eigenvectors.

Exercise 4.14. *Apply the power method to find the largest and smallest eigenvalue and eigenvector for the matrix in Example 4.1.4. Consider* $n = 10, 100$.

4.2.3 Periodic boundary value problem

Let f be a square–integrable periodic function of period 1 such that

$$\int_0^1 f(x)\,dx = 0. \tag{4.27}$$

Consider the periodic problem

$$\begin{aligned} -u''(x) &= f(x), \quad x \in (0,1), && \text{(4.28a)}\\ &\quad \text{with the periodic boundary condition.} && \text{(4.28b)} \end{aligned}$$

Remark 4.2. *Notice that the periodicity of* u *(of period 1) implies that*

$$u(x) = u(x+1)\,\forall x,$$

which implies that

$$\begin{aligned} u(0) &= u(1), && \text{(4.29a)}\\ u'(0) &= u'(1). && \text{(4.29b)} \end{aligned}$$

Exercise 4.15. *Show that, for all nonnegative integer* k, $\frac{d^k u}{dx^k}(0) = \frac{d^k u}{dx^k}(1)$ *whenever* $\frac{d^k u}{dx^k}(0)$ *exists.*

Let us discretize (4.28) by the finite difference method using a uniform mesh $x_j, j = 0, 1, \cdot, N$, with $x_j = jh$ and $h = \frac{1}{N}$. Notice that we need to find the values $u(x_j), j = 0, \cdots, N$, such that $u(x_0) = u(x_N)$. Denoting by U_j and b_j the approximate values of $u(x_j)$ and $f(x_j)$ for $j = 1, \cdots, N$. (Indeed, b_j's are the exact values of $f(x_j)$.) Using (4.29a), we then have

$$\begin{aligned} \frac{-U_N + 2U_1 - U_2}{h^2} &= b_1, && \text{(4.30a)}\\ \frac{-u(x_{j-1}) + 2u(x_j) - u(x_{j+1})}{h^2} &= b_j,\ j = 2, \cdots, N-1, && \text{(4.30b)} \end{aligned}$$

which form an $N-1$ equations in N unknowns. In order to have an $N \times N$ matrix system, the condition (4.29b) may be used. Exploiting $U_0 = U_N$, let us first attempt to use the first–order forward difference approximation to it

$$\frac{U_1 - U_N}{h} = \frac{U_1 - U_N}{h}, \tag{4.31}$$

which is a tautology. The trial of using the second–order central difference approximation to it

$$\frac{U_1 - U_{N-1}}{2h} = \frac{U_1 - U_{N-1}}{2h} \tag{4.32}$$

will not contribute anything, either. Hence a reasonable additional condition may be the condition that the right limit of the derivative at x_0 equal to that of the right limit at x_N if there exists a derivative at x_0. In this case,

$$\frac{U_1 - U_N}{h} = \frac{U_N - U_{N-1}}{h}, \tag{4.33}$$

or equivalently,

$$-U_1 + 2U_N - U_{N-1} = 0. \tag{4.34}$$

However, in general u may not be differentiable at x_0. This implies the possibility of jump of the derivative at x_0 if the left and right derivatives exist. Then, (4.33) should be written as

$$\frac{U_1 - U_N}{h} - \frac{U_N - U_{N-1}}{h} = \alpha, \tag{4.35}$$

or equivalently,

$$\frac{-U_1 + 2U_N - U_{N-1}}{h^2} = -\frac{\alpha}{h}. \tag{4.36}$$

We will come back to this point of discussion of periodicity boundary condition in a moment.

By Gerschgorin's circles theorem, all eigenvalues are included in the disc $\{z \in \mathbb{C} : |z - \frac{2}{h^2}| \leq \frac{2}{h^2}\}$. Hence the least upper bound $|\lambda|$ for $\lambda \in \sigma(A_h)$ is less than or equal to $\frac{4}{h^2}$, while the greatest lower bound $|\lambda|$ for $\lambda \in \sigma(A_h)$ is larger than or may be equal to 0.

We proceed to investigate in more details. First, the eigenvalues and their associated eigenfunctions of the second-order elliptic operator $\mathcal{L} = -\frac{d^2}{dx^2}$:

$L^2(0,1) \to L^2(0,1)$, with the domain of $\mathcal{L}$ is given by

$$H^2_{per}(0,1) = \{v \in L^2(0,1) : \ v', v'' \in L^2(0,1); v(0) = v(1), v'(0) = v'(1)\},$$

whose eigenvalues and eigenfunctions are given by

$$\lambda = (2k\pi)^2, \quad \phi(x) = \sin(2k\pi x), \quad \text{for } k = 0, 1, 2, \cdots . \tag{4.37}$$

Collecting (4.30) and (4.36), we have the following linear system:

$$A\mathbf{U} = \mathbf{b}, \tag{4.38}$$

where A is an $N \times N$ matrix given by

$$A = \frac{1}{h^2}\begin{bmatrix} 2 & -1 & 0 & 0 & \cdots & 0 & 0 & 0 & -1 \\ -1 & 2 & -1 & 0 & \cdots & 0 & 0 & 0 & 0 \\ 0 & -1 & 2 & -1 & \cdots & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \ddots & \ddots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \ddots & \ddots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & -1 & 2 & -1 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & -1 & 2 & -1 \\ -1 & 0 & 0 & 0 & \cdots & 0 & 0 & -1 & 2 \end{bmatrix}, \tag{4.39}$$

and $\mathbf{U}$ and $\mathbf{b}$ are N–dimensional vectors such that $\mathbf{U} = (U_1, \cdots, U_N)^t$ and $\mathbf{b} = (f(x_1), \cdots, f(x_{N-1}), -\frac{\alpha}{h})^t$.

Notice that among these continuous eigenfunctions, there are only n number of them which can be approximated by the n-uniform mesh. They are explicitly given by

$$x^{(k)} = \left(\sin\frac{2k\pi}{n}, \sin\frac{4k\pi}{n}, \cdots, \sin\frac{2(n-1)k\pi}{n}\right)^t, \quad k = 0, 1, \cdots, n-1 \tag{4.40}$$

Here, the eigenfunction 0 corresponding to the eigenvalue 0 represents any nonzere constant function. If A_h acts on $x^{(k)}$, the j-th component of $A_h x^{(k)}$ can be calculated as follows:

$$\begin{aligned} & -\frac{1}{h^2}\sin\frac{2(j-1)k\pi}{n} + \frac{2}{h^2}\sin\frac{2jk\pi}{n} - \frac{1}{h^2}\sin\frac{2(j+1)k\pi}{n} \\ = \ & -\frac{2}{h^2}\sin\frac{2jk\pi}{n}\cos\frac{2k\pi}{n} + \frac{2}{h^2}\sin\frac{2jk\pi}{n} = \frac{2}{h^2}\left(1 - \cos\frac{2k\pi}{n}\right)\sin\frac{2jk\pi}{n}, \end{aligned}$$

owing to the elementary trigonometric formula $\sin(\theta_1+\theta_2)+\sin(\theta_1-\theta_2) = 2\sin\theta_1\cos\theta_2$. We therefore observe that (4.40) are eigenvectors of A_h associated with the eigenvalue $\frac{2}{h^2}\left(1-\cos\frac{2k\pi}{n}\right)$ for $k=0,1,\cdots,n-1$.

Remark 4.3. *Notice that the discrete eigenvectors given by* (4.40) *are nothing but the restrictions of the first* $n-1$ *continuous eigenvectors given by* (4.37) *at the interior nodes* $x_j=\frac{j}{n}, j=0,1,\cdots,n-1$. *However, the discrete eigenvalues* $\frac{2}{h^2}\left(1-\cos\frac{2k\pi}{n}\right)$ *are approximated from the continuous eigenvalues* $(k\pi)^2$ *since*

$$\frac{2}{h^2}\left(1-\cos\frac{2k\pi}{n}\right) = (2k\pi)^2+\mathcal{O}(h^2)$$

due to the expansion:

$$1-\cos\frac{2k\pi}{n} = \frac{1}{2!}\left(\frac{2k\pi}{n}\right)^2-\frac{1}{4!}\left(\frac{2k\pi}{n}\right)^4+\frac{1}{6!}\left(\frac{2k\pi}{n}\right)^6-\cdots.$$

Notice that the eigenvectors satisfy the orthogonality relationship:

$$(x^{(k)},x^{(l)}) = \delta_{kl}\frac{n}{2},$$

where δ_{kl} denotes the Kronecker delta.

We have the following immediate information on the eigenvalues:

Proposition 4.3. *The* N *eigenvevalues of* (4.39) *are given by* $\lambda_k=\frac{2}{h^2}(1-\cos\frac{2k\pi}{N}), k=0,1,\cdots,N-1$.

Using $\frac{2}{h^2}(1-\cos\frac{2k\pi}{N})=\frac{4}{h^2}\sin^2\frac{k\pi}{N}$, we have the following:

Corollary 4.1. *(i)* N *is odd:* $\lambda_0=0$ *is the minimal eigenvalue with algebraic multiplicity 1; ; also, we have* $\lambda_k=\lambda_{N-k}$, $k=1,\cdots,(N-1)/2$, *are the eigenvalues with algebraic multiplicity 2 with* $\lambda_0<\lambda_1<\cdots<\lambda_{(N-1)/2}=\lambda_{(N+1)/2}<\frac{4}{h^2}$.

(ii) N *is even:* $\lambda_0=0$ *and* $\lambda_{N/2}=\frac{4}{h^2}$ *are the minimal and maximal eigenvalues with algebraic multiplicity 1, respectively; also, we have* $\lambda_k=\lambda_{N-k}$, $k=1,\cdots,N/2-1$, *are the eigenvalues with algebraic multiplicity 2 with* $\lambda_0<\lambda_1<\cdots<\lambda_{N/2}$.

Exercise 4.16. *Show that* 0 *is an eigenvalue of the matrix* A *given by* (4.39). *Find an eigenvector associated with eigenvalue* 0.

Exercise 4.17. *Observe that the* $(N-1)\times(N-1)$ *matrix* A *given by* (2.29) *is the* $(N-1)\times(N-1)$ *principal submatrix of the* $N\times N$ *matrix* A *given by* (4.39). *Based on this, show that the first* $N-1$ *columns of* A *given by* (4.39) *are linearly independent.*

Owing to the above two exercises, one sees that the rank of A given by (4.39) is $N-1$.

Exercise 4.18. *Let* $N = 10$ *and find all eigenvalues of the matrix* A *given by* (4.39) *by the inverse iteration method.*

Now let us proceed to solve (**??**) which is singular and of rank 1 deficient. Moreover, one should observe that the summation of all rown in A results in a null vector. This leads to the *consistency condition* that the summation of the right hand side should vanish. That is,

$$\sum_{j=1}^{N-1} f(x_j) - \frac{\alpha}{h} = 0, \tag{4.41}$$

which implies that α should be chosen such that

$$\alpha = \frac{1}{N}\sum_{j=1}^{N-1} f(x_j) : \tag{4.42}$$

otherwise, the system (**??**) does not possess a solution. *Thus, from now on, we assume that* (4.42) *is imposed.* Hence, by eliminating α, the last component b_N of $\mathbf{b}$ is given by

$$b_N = -\sum_{j=1}^{N-1} f(x_j). \tag{4.43}$$

Since A is symmetric and the first $(N-1)$ columns are linearly independent, the first $(N-1)$ rows are linearly independent. This means that in order to solve (**??**) one can solve the subsystem

$$\tilde{A}\mathbf{U} = \tilde{\mathbf{b}}, \tag{4.44}$$

where $\tilde{A}$ and $\tilde{\mathbf{b}}$ consist of the $(N-1)\times N$ submatrix of A and the first $(N-1)$ rows of $\mathbf{b}$, respectively.

A (possibly) simplest way of solving (4.44) is to write it as an $(N-1)\times(N-1)$ matrix system by sending the last variable U_N to the right hand side so that

$$\widehat{A}\widehat{\mathbf{U}} = \widehat{\mathbf{b}}, \tag{4.45}$$

where $\widehat{\mathbf{U}} = (U_1, \cdots, U_{N-1})^t$, $\widehat{A}$ is the $(N-1)\times(N-1)$ matrix given by (2.29) and $\widehat{\mathbf{b}}$ is the modified vector from $\mathbf{b}$ in the first and last components such that $\widehat{b}_1 = f(x_1) + \frac{1}{h^2}U_N$ and $\widehat{b}_{N-1} = f(x_{N-1}) + \frac{1}{h^2}U_N$.

To summarize, (4.42) can be solved by finding a family of solutions to (4.45) by regarding U_N as a parameter.

Indeed, the differential equation (4.28) with the periodic boundary condition is unique up to an additive constant. In other words, if $u(x)$ is a solution, $u(x)+c$ is also a solution for any constant c. In order to ensure unique solvability, one can add an additional contraint to the governing equation. A popular additional comdition is

$$\int_0^1 u(x)\,dx = 0. \tag{4.46}$$

This condition can be implemented as a postprocessing: set $const = \frac{1}{N}\sum_{j=1}^{N} U_j$. Then, subtract the average $const$ from all components of $U_j, j = 1, \cdots, N$.

Exercise 4.19. *Define a periodic function f of period 1 by*

$$f(x) = \begin{cases} -x + \frac{1}{4}, & x \in [0, 1/2] \\ x - \frac{3}{4}, & x \in [1/2, 1] \end{cases}$$

with extension to $\mathbb{R}$ periodically. Verify that

$$u(x) = \begin{cases} \frac{x^3}{6} - \frac{x^2}{8} + \frac{1}{192}, & x \in [0, 1/2] \\ -\frac{x^3}{6} + \frac{3}{8}x^2 - \frac{x}{4} + \frac{3}{64}, & x \in [1/2, 1] \end{cases}$$

is a solution to (4.28),(4.43), and (4.46). Write a program to solve this problem.

Chapter 5

Orthogonal Decomposition Methods

5.1 Householder transformation

5.1.1 Motivation

Recall that each step in Gaussian elimination is to multiply the matrices G_j and P_j so that the resulting matrix $U = G_{n-1}P_{n-1}G_{n-2}P_{n-1}\cdots G_1P_1A$ is upper triangular. As before let $A^{(0)} = A$ and $A^{(j)} = G_jP_jA^{(j-1)}$. If $\varepsilon^{(j)}$ is the bound of round-off error arising from the multiplication G_jP_j to $A^{(j-1)}$, then the relative error in the final solution x is estimated by $\kappa(A^{(j)})\varepsilon^{(j)}$. Therefore

$$\frac{\|\Delta x\|}{\|x\|} \leq \sum_{j=0}^{n} \kappa(A^{(j)})\varepsilon^{(j)},$$

where $\varepsilon^{(0)}$ denotes the errors in the initial data A and b.

Lemma 5.1. *Let $\|\cdot\|$ be the $2-$norm, then for any $n \times n$ A and a unitary matrix U, $\kappa(UA) = \kappa(A)$.*

Proof. $\kappa(UA) = \|UA\|_2\|(UA)^{-1}\|_2 = \|A\|_2\|A^{-1}U^*\|_2 = \|A\|_2\|A^{-1}\|_2 = \kappa(A)$. □

This motivates to try to find an alternative way to multiply $A^{(j-1)}$ to the left by a matrix $H^{(j)}$ to obtain $A^{(j)}$, recursively, in order to obtain a resulting matrix

$$R := A^{(n-1)} = H^{(n-1)}H^{(n-2)}\cdots H^{(1)}A^{(0)} \tag{5.1}$$

to be a matrix which is easy to be inverted. For instance, if $A^{(n-1)}$ is triangular matrix, it is easy to be inverted as in the Gaussian elimination. Notice that

if follows from (5.1) that

$$A = \left[H^{(n-1)}H^{(n-2)}\cdots H^{(1)}\right]^{-1} A^{(n-1)} =: H^{-1}R. \tag{5.2}$$

Having in mind that a unitary matrix is also as easily inverted as triangular matrices, and a multiple of several unitary matrices is also unitary, one can try to find each $H^{(j)}$ to be unitary. But the problem is how to systematically obtain such unitary matrices. The next two sections are due to the original ideas of Householder and Givens.

5.1.2 Householder transformation

Let $H_v : x \in \mathbb{R}^n \longmapsto H_v x \in \mathbb{R}^n$ be the reflection of x to the hyperplane orthogonal to v spanned by $\{v\}$. Then, as in the following figure,

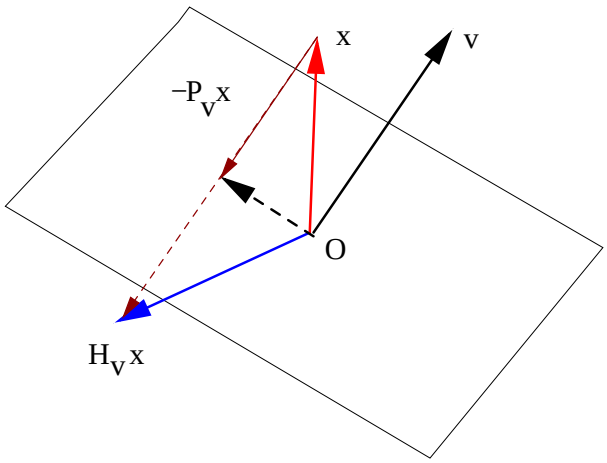


Figure 5.1.1: Householder transform of x which is the reflection of x to the plane passing through the origin O and orthogonal to v: $H_v x = x - \frac{vv^*}{v^*v}x$

$$(H_v x - x)/\!\!/ v, \longrightarrow H_v x - x = -2\frac{(v\cdot x)}{v\cdot v}v.$$

Therefore,

$$H_v x = \left(I - 2\frac{vv^t}{v^t v}\right)x$$

This transformation can be extended to the vector space over the complex field.

Definition 5.1. *Given $v \in \mathbb{C}^n$, $H_v = I - 2\frac{vv^*}{v^*v}$ is called the Householder transformation (or Householder matrix) with respect to v.*

Properties 5.1. *The Householder matrix H_v has the following properties.*

1. *H_v is Hermitian : $H_v^* = H_v (H_v^* = H_v)$*
2. *H_v is involving $(H_v^2 = I)$*
3. *H_v is unitary.*
4. *$H_v v = -v$*
5. *$H_v x = x$ if $(x, v) = 0$.*

Proof $H_v^* H_v = (I - 2\frac{vv^*}{v^*v})^*(I - \frac{2vv^*}{v^*v}) = I - \frac{4vv^*}{v^*v} + \frac{4vv^*vv^*}{(v^*v)^2} = I$ $\square$.
Due to the above properties, given a vector x, its Householder transform $H_v x$ under v has the same 2-norm as v. That is,

$$\|H_v x\|_2 = \|x\|_2$$

Exercise 5.1. *Notice that in the definition of Householder matrix $H_v = I - 2\frac{vv^*}{v^*v}$, the vector v can be normalized: Thus if $\|v\|_2 = 1$, $H_v = I - 2vv^*$. Assume that v is normalized and show the following statements:*

1. *For any pair of vectors $x, y \in \mathbb{C}^n, \|x\|_2 = \|y\|_2, x \neq y, x^*y = y^*x$, find a unit vector v such that $H_v x = y$. Indeed, if $H_v x = y$, then one can verify that $x^*y = y^*x$.*

5.1.3 Parallelization to e_1 of a vector using a Householder matrix

Let us study more properties of the Householder transformation:

$$\begin{aligned} H_v : \mathbb{C}^n &\to \mathbb{C}^n \\ x &\mapsto H_v x = (I - \frac{2vv^*}{v^*v})x. \end{aligned}$$

Given $x \neq 0, \quad x \in \mathbb{C}^n$, we will construct a $v \in \mathbb{C}^n$ such that $H_v x = \alpha e_1$ for some $\alpha \in \mathbb{C}$. For this, we need to have that

$$H_v x = (I - 2\frac{vv^*}{v^*v})x = x - 2\frac{v^*x}{v^*v}v = \alpha e_1. \tag{5.3}$$

This means

$$v \in Span\{x, e_1\} \text{ i.e. } v = \beta' x + \beta e_1, \text{ for some } \beta, \beta' \in \mathbb{C}.$$

Since we are only interested in the direction of v, we may assume $\beta' = 1$ Thus, we start with

$$v = x + \beta e_1. \tag{5.4}$$

We then have

$$\begin{aligned} v^* x &= (x^* + \overline{\beta} e_1^t)x = \|x\|^2 + \overline{\beta} x_1 \in \mathbb{C}, && \text{(5.5a)} \\ v^* v &= (x^* + \overline{\beta} e_1^t)(x + \beta e_1) = \|x\|_2^2 + |\beta|^2 + \beta\overline{x_1} + \overline{\beta} x_1. && \text{(5.5b)} \end{aligned}$$

Notice that

$$\beta\overline{x_1} + \overline{\beta} x_1 = 2\,\mathrm{Re}\,(\beta\overline{x_1}) = 2\,\mathrm{Re}\,(\overline{\beta} x_1).$$

It follows from (5.3) that

$$H_v x = x - 2\frac{v^* x}{v^* v}(x + \beta e_1) = (1 - 2\frac{v^* x}{v^* v})x - 2\beta\frac{v^* x}{v^* v} e_1 = \alpha e_1,$$

which implies that

$$1 - 2\frac{v^* x}{v^* v} = 0,$$

Thus, using (5.5), one has

$$\|x\|_2^2 + |\beta|^2 + \beta\overline{x_1} + \overline{\beta} x_1 - 2(\|x\|_2^2 + \overline{\beta} x_1) = 0.$$

Since

$$-\|x\|_2^2 + |\beta|^2 + \beta\overline{x_1} - \overline{\beta} x_1 = 0.$$

Here, $-\|x\|_2^2 + |\beta|^2$ and $-(\beta\overline{x_1} - \overline{\beta} x_1)$ are equal, and therefore the latter must be real. Since $\beta\overline{x_1} - \overline{\beta} x_1 = 2i\,\mathrm{Im}\,(\beta\overline{x_1})$ is also complex, it is equal to zero. Thus,

$$\beta\overline{x_1} = \overline{\beta} x_1 \in \mathbb{R}. \tag{5.6}$$

and $|\beta|^2 = \|x\|_2^2$, or $\beta = e^{i\sigma}\|x\|_2$ for some $\sigma \in \mathbb{R}$. Thus choose v such that

$$v = x + e^{i\sigma}\|x\|_2 e_1.$$

In particular, we choose a σ such that

$$\|v\|_2 \geq \|x\|_2. \tag{5.7}$$

This will enhance the stability of calculating and simplify the actual computation in (5.9). Thus if $x_1 = |x_1|e^{i\omega}$, then (5.6) implies that

$$\begin{aligned} \overline{\beta}x_1 &= e^{i(\omega-\sigma)}|x_1|\|x\|_2 \qquad \text{which is real} \\ &= \pm|x_1|\|x\|_2 \end{aligned}$$

and thus

$$\overline{\beta} = \pm\frac{|x_1|}{x_1}\|x\|_2, \quad x_1 \neq 0.$$

Hence

$$\beta = \pm\frac{x_1|x_1|}{x_1\overline{x_1}}\|x\|_2 = \pm\frac{x_1}{|x_1|}\|x\|_2.$$

The condition (5.7) is then equivalent to

$$\|v\|_2 = \|x + \beta e_1\|_2 = \left\|x \pm \frac{x_1}{|x_1|}\|x\|_2 e_1\right\|_2 \geq \|x\|_2.$$

Thus finally choose β and v such that

$$\beta = \frac{x_1}{|x_1|}\|x\|_2; \quad v = x + \beta e_1. \tag{5.8}$$

With this choice of β, (5.5) yields

$$v^*x = \|x\|_2^2 + \frac{\overline{x_1}}{|x_1|}\|x\|_2 x_1 = \|x\|_2(\|x\|_2 + |x_1|), \tag{5.9a}$$

$$v^*v = \|x\|_2^2 + \left|\frac{x_1}{|x_1|}\|x\|_2\right|^2 + \beta\overline{x_1} + \overline{\beta}x_1 \tag{5.9b}$$

$$= \|x\|_2^2 + \|x\|_2^2 + 2\|x\|_2|x_1| = 2\|x\|_2(\|x\|_2 + |x_1|). \tag{5.9c}$$

Thus, using these we have from (5.4) and (5.3) that

$$H_v x = x - 2\frac{v^*x}{v^*v}v = -\beta e_1 = -\frac{x_1}{|x_1|}\|x\|_2 e_1. \tag{5.10a}$$

5.1.4 Annihilating certain blocks in a vector using a Householder matrix

Householder transformation(matrices) can be used to make any contiguous block of vector components zeros. In particular we wish to annihilate the $k+1$st to jth components of of a given vector. Given $x = (x_1, \cdots, x_n)^T$, let

$$v = (0, \cdots, 0, x_k + \beta, x_{k+1}, \cdots, x_j, 0, \cdots, 0)^T;\ \beta = \frac{x_k}{|x_k|}\sqrt{|x_k|^2 + \cdots + |x_j|^2}.$$

Then

$$H_v x = (I - 2\frac{vv^*}{v^*v})x = (x_1, \cdots, x_{k-1}, -\beta, 0, \cdots, 0, x_{j+1}, \cdots, x_n)^T.$$

5.2 Givens rotation

Theorem 5.1. *If $A \in M(2,2)$ is orthogonal, i.e. $A^*A = I$, then either A is a Householder matrix or A is a Givens rotation.*

Proof. Let

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}.$$

Then since $A^*A = I$,

$$A^*A = \begin{pmatrix} \overline{a} & \overline{c} \\ \overline{b} & \overline{d} \end{pmatrix} \begin{pmatrix} a & b \\ c & d \end{pmatrix} = \begin{pmatrix} |a|^2 + |c|^2 & \overline{a}b + \overline{c}d \\ a\overline{b} + c\overline{d} & |b|^2 + |d|^2 \end{pmatrix}.$$

Thus we have

$$\left\| \begin{pmatrix} a \\ c \end{pmatrix} \right\|_2 = 1, \quad \left\| \begin{pmatrix} b \\ d \end{pmatrix} \right\|_2 = 1, \quad \begin{pmatrix} a \\ c \end{pmatrix}^* \begin{pmatrix} b \\ d \end{pmatrix} = 0.$$

We therefore have the two cases depending on the choice of signature from $\pm$.

$$\begin{pmatrix} a \\ c \end{pmatrix} = \begin{pmatrix} \cos\theta \\ \sin\theta \end{pmatrix}, \begin{pmatrix} b \\ d \end{pmatrix} = \begin{pmatrix} \cos\left(\theta \pm \frac{\pi}{2}\right) \\ \sin\left(\theta \pm \frac{\pi}{2}\right) \end{pmatrix}$$

for some θ.

1. $ad - bc = 1 :\Longrightarrow \begin{pmatrix} a & b \\ c & d \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} = \begin{pmatrix} \cos(-\theta) & \sin(-\theta) \\ -\sin(-\theta) & \cos(-\theta) \end{pmatrix}$, which is a Givens rotation.
2. $ad - bc = -1 :\Longrightarrow \begin{pmatrix} a & b \\ c & d \end{pmatrix} = \begin{pmatrix} \cos\theta & \sin\theta \\ \sin\theta & -\cos\theta \end{pmatrix}$, which is a Householder reflection with respect to $v = (-\sin\frac{\theta}{2}, \cos\frac{\theta}{2})$. Indeed,

$$H_v = I - 2\frac{vv^*}{v^*v} = \begin{pmatrix} 1 - 2\sin^2\frac{\theta}{2} & 2\sin\frac{\theta}{2}\cos\frac{\theta}{2} \\ 2\sin\frac{\theta}{2}\cos\frac{\theta}{2} & 1 - 2\cos^2\frac{\theta}{2} \end{pmatrix} = \begin{pmatrix} \cos\theta & \sin\theta \\ \sin\theta & -\cos\theta \end{pmatrix}.$$

□

Let us write the Givens rotation of angle θ on the plane of the k-th and ℓ-th coordiates as follows:

$$G(k,\ell,\theta) = \begin{pmatrix} I & \vdots & 0 & \vdots & 0 \\ \cdots & c & \cdots & s & \cdots \\ 0 & \vdots & I & \vdots & 0 \\ \cdots & -s & \cdots & c & \cdots \\ 0 & \vdots & 0 & \vdots & I \end{pmatrix} \quad H(k,\ell,\theta) = \begin{pmatrix} I & \vdots & 0 & \vdots & 0 \\ \cdots & c & \cdots & -s & \cdots \\ 0 & \vdots & I & \vdots & 0 \\ \cdots & -s & \cdots & -c & \cdots \\ 0 & \vdots & 0 & \vdots & I \end{pmatrix}.$$

The Givens rotation

$$G(k,\ell,\theta) : x \mapsto y = G(k,\ell,\theta)x$$

is useful to annihilate a specific component. Since

$$y_k = cx_k + sx_\ell,\ y_\ell = -sx_k + cx_\ell,\ y_l = x_l \text{ if } l \neq k,\ell, \tag{5.11}$$

in order to make $y_\ell = 0$, choose c and s such that

$$c = \frac{x_k}{\sqrt{x_k^2 + x_\ell^2}},\ s = \frac{x_\ell}{\sqrt{x_k^2 + x_\ell^2}}. \tag{5.12}$$

Thus, Givens rotations are used to zero a specific entry.

ALGORITHM 5.1. *Givens*(k,ℓ,θ,A,m,n), $A \in M(m,n) \to G(k,\ell,\theta)A$

1 $v(:) = a(k,:)$
2 $w(:) = a(\ell,:)$
3 $a(k,:) = \cos\theta * v(:) + \sin\theta * w(:)$
4 $a(\ell,:) = -\sin\theta * v(:) + \cos\theta * w(:)$

Some useful properties of Givens rotations are as follows:

1. Givens rotation is a rank-two perturbation of identity.
2. Since $G(k,\ell,\theta)^{-1} = G(k,\ell,\theta)^t = G(k,\ell,-\theta)$, Givens rotations are orthogonal:

$$G(k,\ell,\theta)^t G(k,\ell,\theta) = G(k,\ell,\theta)G(k,\ell,\theta)^t = I$$

5.3 QR-decomposition

Definition 5.2. *If there is an orthogonal (or unitary) matrix Q and an upper triangular matrix R such that*

$$A = QR, \tag{5.13}$$

A is said to have a QR-decomposition.

In general, QR-decomposition is not unique.

Since the determinant of an orthogonal (unitary) matix is equal to one,

$$\det A = \det R$$

for all square matrices A.

Hence, the linear system $Ax = b$ can be solved easily using a QR-decomposition as follows:

1. Multiply by Q^* to the both sides of $Ax = b$ to get

$$Rx = Q^*b. \tag{5.14}$$

2. Solve (5.14) by back-substitution.

We turn to investigate several methods to decompose a matrix into QR-form.

5.3.1 QR-decomposition by Householder transformation

A matrix $A \in M(n, n)$ can be reduced step by step using the unitary Householder matrices H_j defined as follows: Let A be a nonsingular matrix.

ALGORITHM 5.2.

```
1 A^(0) := A
2 do j=1, n-1
3    A^(j) = H_j A^(j-1)
4 enddo
```

The algorithm produces

$$A^{(n-1)} = H_{n-1}H_{n-2}\cdots H_1A^{(0)}.$$

Assume that a_{11} is not equal to zero. (If $a_{11} = 0$, a partial pivoting will do.) First, Let H_1 be determined by $H_1 A_1^{(0)} = \alpha e_1$ where $A_k^{(j)}$ is the kth-column of the matrix $A^{(j)}$. Here.

$$A^{(0)} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix}$$

and $H_1 = H_{v^{(1)}}$, with $v^{(1)}$ such that $H_{v^{(1)}} A_1^{(0)} = \alpha_1 e_1$

$$A^{(1)} = H_1 A^{(0)} = \begin{pmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \cdots & a_{1n}^{(1)} \\ 0 & a_{22}^{(1)} & \cdots & a_{2n}^{(1)} \\ \vdots & & \ddots & \vdots \\ 0 & a_{n2}^{(1)} & \cdots & a_{nn}^{(1)} \end{pmatrix}.$$

Next, find a vector $v^{(2)}$ such that $H_{v^{(2)}} A_2^{(1)}$ takes the form

$$\begin{pmatrix} a_{12}^{(1)} \\ \alpha_2 \\ 0 \\ \vdots \\ 0 \end{pmatrix}.$$

Set $H_2 := H_{v^{(2)}}$ and apply this Householder transformation to $A^{(1)}$ to obtain

$$A^{(2)} = H_2 A^{(1)} = \begin{pmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \cdots & a_{1n}^{(1)} \\ 0 & a_{22}^{(2)} & \cdots & a_{2n}^{(2)} \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & a_{nn}^{(2)} \end{pmatrix}.$$

Continue this iterative procedure to obtain

$$A^{(n-1)} = H_{n-1} A^{(n-2)} = H_{n-1} H_{n-2} \cdots H_1 A^{(0)} = \begin{pmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \cdots & a_{1n}^{(1)} \\ 0 & a_{22}^{(2)} & \cdots & a_{2n}^{(2)} \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & a_{nn}^{(n-1)} \end{pmatrix}.$$

Writing $A^{(n-1)} = R$, which is an upper triangular matrix, we have the QR decomposition

$$A = QR = (H_{n-1}H_{n-2}\cdots H_1)^{-1}R.$$

If $x_1 \neq 0$, by normalization, we may assume that $\frac{x_1}{|x_1|} = 1$. Then, with $v = x + \|x\|_2 e_1$, we have

$$\begin{aligned} H_v x &= (I - 2\frac{vv^*}{v^*v})x = x - 2\frac{(x + \|x\|_2 e_1)(x^*x + \|x\|_2 x_1)}{(x^* + \|x\|_2 e_1{}^t)(x + \|x\|_2 e_1)} \\ &= x - 2\frac{(x + \|x\|_2 e_1)\|x\|_2{}^2}{(\|x\|_2{}^2 + \|x\|_2{}^2)} = -\|x\|_2 e_1. \end{aligned}$$

Suppose

$$\begin{pmatrix} a_{11}^{(j-1)} & \cdots & a_{1,j-1}^{(j-1)} & a_{1,j}^{(j-1)} & \cdots & a_{1,n}^{(j-1)} \\ & \ddots & \vdots & \vdots & & \\ 0 & & a_{j-1,j-1}^{(j-1)} & a_{j-1,j}^{(j-1)} & \cdots & a_{j-1,n}^{(j-1)} \\ & & & a_{j,j}^{(j-1)} & \cdots & a_{j,n}^{(j-1)} \\ & & & \vdots & & \vdots \\ & & & a_{n,j}^{(j-1)} & \cdots & a_{n,n}^{(j-1)} \end{pmatrix}$$

Determine a Householder matrix $\hat{H}_j$ such that

$$\hat{H}_j \begin{pmatrix} a_{jj}^{(j-1)} \\ \vdots \\ a_{nj}^{(j-1)} \end{pmatrix} = \alpha_j \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \quad \text{for some } \alpha_j.$$

Then set

$$H_j = \begin{pmatrix} I_{j-1} & 0 \\ 0 & \hat{H}_j \end{pmatrix}$$

so that

$$A^{(j)} = H_j A^{(j-1)}, \cdots, A^{(n-1)} = H_{n-1}A^{(n-2)} = \text{Upper triangular matrix} = U(= R).$$

Summarizing,

$$H_{n-1}H_{n-2}\cdots H_1 A = R; \qquad A = H^*R = QR.$$

Exercise 5.2. *Write a program to solve $Ax = b$ using Householder transformations, where A is the matrix given by* (2.29) *with $b = e_1$ for $n = 100$.*

The following programs solve $Ax = b$ using Householder transformations.

F90 Program 5.1: Householder transform with partial pivoting, Lines 19-20 implement (5.8)**.**

```
1  ! Householder elimination with partial pivot
2  ! Written by D. Sheen 2004. 4. http://www.nasc.snu.ac.kr
3  subroutine householder_tran(a,b,n,j)
4    implicit none
5    interface
6       function house(v,x) result(h)! Householder transform overwites x = (I-2vv*/v*v)x
7         real(8), intent(in):: v(:), x(:)
8         real(8):: h(size(v,1))
9         real(8):: vstarv, vstarx
10      end function house
11   end interface
12
13   integer:: n, j, k, l, dim
14   real(8), intent(inout) :: a(j:n,j:n), b(j:n)
15   real(8), allocatable :: v(:)
16   real(8) :: m, beta
17
18   dim = n-j+1; allocate(v(1:dim) )
19   beta = a(j,j)/abs(a(j,j))*sqrt( dot_product(a(j:n,j),a(j:n,j)) )
20   v(1) = a(j,j) + beta; v(2:dim) = a(j+1:n,j)
21   a(j,j) = -beta;    a(j+1:n,j) = 0.d0 ! The jth columnn
22
23   do k = j+1, n! The kth columnn
24      a(j:n,k) = house(v,a(j:n,k))
25   enddo
26   b(j:n) = house(v,b(j:n))
27   deallocate(v)
28
29 end subroutine householder_tran
30
31 function house(v,x) result(h)! Householder transform overwites x = (I-2vv*/v*v)x
32   real(8), intent(in):: v(:), x(:)
33   real(8):: h(size(v,1))
34   real(8):: vstarv, vstarx
35   vstarv = dot_product(v,v); vstarx = dot_product(v,x)
36   h(:) = x(:) - 2.d0*vstarx/vstarv*v(:)
37 end function house
```

5.3.2 QR-decomposition by Givens rotationn

Instead of using Householder transformation, one may use Givens rotatons to have a QR-decomposition of A. The procedure is quite similar to that given in the previous subsection.

Use the formula (5.11) and (5.12) with $k = 1$ and $\ell = 2, \cdots, n$, and then with $k = 2$ and $\ell = 3, \cdots, n$, and so on, until with $k = n - 1$ and $\ell = n$. This will annihilate the lower part of A column by column. For instance, in order annihilate $a(j+1 : n, j)$, with suitable modification to $a(j : n, j : n)$ and $b(j : n)$, apply the following algorithm.

ALGORITHM 5.3. *Givens*$(k, \ell, \theta, A, m, n)$, $A \in M(m, n) \to G(k, \ell, \theta)A$

1 $v(:) = a(k, :)$
2 $w(:) = a(\ell, :)$
3 $a(k, :) = \cos\theta * v(:) + \sin\theta * w(:)$
4 $a(\ell, :) = -\sin\theta * v(:) + \cos\theta * w(:)$

Denote by $G_j = \text{Givens}(k(j), \ell(j), \theta(j), A, n, n)$ the jth Givens rotation in the above procedure. Then, we have

$$G_N \cdots G_2 G_1 A = R,$$

where R is an upper triangular matrix. Setting

$$Q = G_1^t \cdots G_N^t,$$

one gets the QR-decomposition

$$A = QR.$$

In order to store G_j in a compact manner for each jth roation, consider

$$G_j = \begin{pmatrix} & \vdots & & \vdots & \\ \cdots & c_j & & s_j & \cdots \\ & \vdots & & \vdots & \\ \cdots & -s_j & \cdots & c_j & \cdots \\ & \vdots & & \vdots & \end{pmatrix},$$

we try to store only the two indices $k(j)$ and $\ell(j)$ where the rotation is taken and the information about the amount of rotation $\rho(j)$ in the following form $(k(j), \ell(j), \rho(j))$. We wish to store only one of the information from c_j and s_j. Since it is more stable to compute $\sqrt{1 - s_j^2}$ than $\sqrt{1 - c_j^2}$ if $|s_j| < |c_j|$, it

is desirable to store the larger of the two $|c_j|$ and $|s_j|$. For this, set

$$\rho(j) = \begin{cases} 1 & \text{if } c_j = 0 \\ s_j/2 & \text{if } |s_j| < |c_j| \\ 2/c_j & \text{if } |c_j| \le |s_j| \end{cases}.$$

Then define $k'(j) = \text{sign}(c_j)k(j), \ell'(j) = \text{sign}(s_j)\ell(j)$, and store only $(k'(j), \ell'(j), \rho(j))$. In the process of recovery of G_j, use the following algorithm.

ALGORITHM 5.4.

1 `if` $(\rho(j) == 1)$ `then` $c_j = 0; s_j = \text{sign}\,\ell'(j)$

2 `else if` $(|\rho(j)| < 1)$ `then` $s_j = 2\rho(j); \quad c_j = \text{sign}(k'(j))\sqrt{1 - s_j^2}$

3 `else` $c_k = 2/\rho(j); \quad s_j = \text{sign}(l'(j))\sqrt{1 - c_j^2}$

4 `end if`

Exercise 5.3. *Write a program to solve $Ax = b$ using Givens transformations, where A is the matrix given by* (2.29) *with $b = e_1$ for $n = 100$.*

The following programs solve $Ax = b$ using Givens transformations.

F90 Program 5.2: Givens transform with partial pivoting, Lines 19-20 implement (5.8)**.**

```
1  ! Givens rotation with partial pivot
2  ! Written by D. Sheen 2004. 4. http://www.nasc.snu.ac.kr
3  subroutine givens_tran(a,b,n,j)
4    implicit none
5    integer, intent(in):: n, j
6    integer:: k, l
7    real(8), intent(inout) :: a(j:n,j:n), b(j:n)
8    real(8), allocatable :: v(:), w(:)
9    real(8) :: c, s
10
11   allocate(v(j:n), w(j:n)) ! row vectors
12   do l = j+1, n
13      c = a(j,j)/sqrt( a(j,j)**2 + a(l,j)**2 )
14      s = a(l,j)/sqrt( a(j,j)**2 + a(l,j)**2 )
15      v(j:n) = a(j,j:n);       w(j:n) = a(l,j:n)
16      a(j,j:n) = c*v + s*w;     a(l,j:n) = -s*v + c*w
17
18      v(j) = b(j);       w(j) = b(l)
19      b(j) = c*v(j) + s*w(j);     b(l) = -s*v(j) + c*w(j)
20   end do
21   deallocate(v, w)
22
23 end subroutine givens_tran
```

5.3.3 QR-decomposition by the Gram-Schmidt orthogonalization

Let us look at an algorithm to obtain a QR-decomposition $QA = \begin{pmatrix} R \\ 0 \end{pmatrix}$, $A = \begin{pmatrix} Q^*R \\ 0 \end{pmatrix}$. QR-decomposition of A can be also obtained by Gram-Schmidt orthogonalization. Suppose that the $m \times n$ matrix A has rank n. Then, we can write $A = [a_1, \ldots, a_n]$ with linearly independent m-dimensional vectors $a_1, \ldots, a_n$ and $Q = [q_1, \ldots, q_m]$ with orthonormal vectors $q_1, \cdots, q_m$.

$$[a_1, \ldots, a_n] = [q_1, \ldots, q_n \mathbb{O} \cdots \mathbb{O}] \begin{pmatrix} r_{11} & \cdots & \cdots & r_{1n} \\ & r_{22} & \cdots & r_{2n} \\ & & \ddots & \vdots \\ & & & r_{nn} \\ \mathbb{O} & \mathbb{O} & \cdots & \mathbb{O} \end{pmatrix}. \tag{5.15}$$

By multiplying Q^t to the left of both sides, one gets

$$q_j^t a_k = \begin{cases} r_{jk} & \text{for } 1 \le j \le k, \\ 0, & \text{for } k < j \le n \end{cases},$$

which implies that a_k has the (orthogonal) components r_{jk} of in the direction of q_j only for $1 \le j \le k$. Thus, for $1 \le k \le n$,

$$a_k = \sum_{j=1}^{k} \left(q_j^t a_k\right) q_j, \text{ or } \mathrm{Span}\{a_1, \ldots, a_k\} \subset \mathrm{Span}\{q_1, \ldots, q_k\}.$$

Since the dimension of both spaces are equal, $\mathrm{Span}\{a_1, \ldots, a_k\} = \mathrm{Span}\{q_1, \ldots, q_k\}$ for $1 \le k \le n$. Moreover, we have

$$R(A) = \mathrm{Span}\{q_1, \ldots, q_n\}, \tag{5.16a}$$
$$R(A)^{\perp} = \mathrm{Span}\{q_{n+1}, \ldots, q_m\}. \tag{5.16b}$$

From (5.15) it follows that

$$a_k = \sum_{j=1}^{k-1} r_{jk} q_j + r_{kk} q_k,$$

which yields

$$q_k = \frac{1}{r_{kk}} \left(a_k - \sum_{j=1}^{k-1} r_{jk} q_j \right), \; 1 \le k \le n. \tag{5.17}$$

We observe that since $\|q_1\|_2 = 1$, Equation (5.17) for $k = 1$ implies $r_{11} = \|a_1\|_2$ and $q_1 = \frac{1}{r_{11}} a_1$.

Then, for $k = 2$, Equation (5.17) is nothing but

$$q_2 = \frac{1}{r_{22}} \left(a_2 - r_{12} q_1 \right).$$

Multiplying by q_1^t from the left of both sides in the above equation, one gets

$$r_{12} = q_1^t a_2,$$

and again since $\|q_2\|_2 = 1$, Equation (5.17) for $k = 2$ gives

$$r_{22} = \|a_2 - r_{12} q_1\|_2.$$

This process continues to give the following algorithm.

ALGORITHM 5.5. *Gram-Schmidt orthogonalization: A is overwritten with Q*

```
1 do j=1,n
2    r_jj = sqrt(a_j^T a_j)
3    a_j = a_j / r_jj
4    do k=j+1,n
5       r_jk = a_j^T a_k
6       a_k = a_k - a_j * r_jk
7    enddo
8 enddo
```

The above algorithm produces the m-dimensional orthonormal vectors $q_1, \cdots, q_n$ and the upper triangular matrix R part.

Exercise 5.4. *Write a Fortran program for the above QR-decomposition by the Gram-Schmidt orthogolization. For this, let A be the usual tridiagonal matrix A (2.29) for $n = 100$ and find the Q and R matrices for A.*

5.4 Singular Value Decomposition(SVD)

Example 5.1. *Let* $A = \frac{1}{25}\begin{bmatrix} 24 & 43 \\ 57 & 24 \end{bmatrix}$. *and set* $E = \{y \in \mathbb{R}^2 : y = Ax, \|x\|_2 = 1\}$. *We will investigate in the image* E *of the unit sphere under the linear transformation* A. *Of course, since* $A^{-1} =$ *exists, one can write*

$$E = \{y \in \mathbb{R}^2 : \|A^{-1}y\|_2 = 1, \}$$

where $A^{-1} = \frac{1}{25}\begin{bmatrix} -8 & 43 \\ 19 & -8 \end{bmatrix}$. *Thus, we have* $(-8y_1 + 43y_2)^2 + (19y_1 - 8y_2)^2 = 25^2$, *or* $425y_1^2 - 496y_1y_2 + 1913y_2^2 = 25^2$ *which is an equation of ellipse. We now take a different approach for this problem. Notice that*

$$A = U\Sigma V^T = \begin{bmatrix} \frac{3}{5} & -\frac{4}{5} \\ \frac{4}{5} & \frac{3}{5} \end{bmatrix}\begin{bmatrix} 3 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} \frac{4}{5} & \frac{3}{5} \\ \frac{3}{5} & -\frac{4}{5} \end{bmatrix}^*. \tag{5.18}$$

where U *and* V *are orthogonal matrices. Notice that* U *is a rotation and* V *is a reflection. Thus, since* $\|V^Tx\|_2 = 1$, *we have*

$$\begin{aligned} E &= \{y \in \mathbb{R}^2 : y = U\Sigma V^T x \|x\|_2 = 1, \} \\ &= \{y \in \mathbb{R}^2 : \Sigma^{-1}U^Ty = V^Tx, \|x\|_2 = 1, \} \\ &= \{y \in \mathbb{R}^2 : \|\Sigma^{-1}U^Ty\|_2 = 1\} \\ &= \{y \in \mathbb{R}^2 : \|\Sigma^{-1}z\|_2 = 1,\ z = U^Ty\} \\ &= \{y \in \mathbb{R}^2 : \|\Sigma^{-1}z\|_2 = 1,\ y = Uz\}. \end{aligned}$$

Thus, the equation $\frac{z_1^2}{3} + z_2^2 = 1$ *describes an ellipse, and then* E *is the rotation of it with the angle* $\theta = \text{atan}(\frac{4}{3})$.

$$u_1 = \left(\frac{3}{5}, \frac{4}{5}\right)^t, \quad u_2 = \left(-\frac{4}{5}, \frac{3}{5}\right)^t$$

In general in n-dimensional vector space, a sphere is transformed by a nonsingular matrix A *into an ellipsoid with singular values being the semi-axes of* E.

Theorem 5.2. *For* $A \in \mathcal{M}(m,n)$, *there exist unitary matrices* $U = [u_1, \ldots, u_m] \in \mathcal{M}(m,m)$ *and* $V = [v_1, \ldots, v_n] \in \mathcal{M}(n,n)$ *such that*

$$U^*AV = \text{diag}(\sigma_1, \ldots, \sigma_p) =: \Sigma, \quad p = \min(m,n) \tag{5.19}$$
$$\text{where } \sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_p.$$

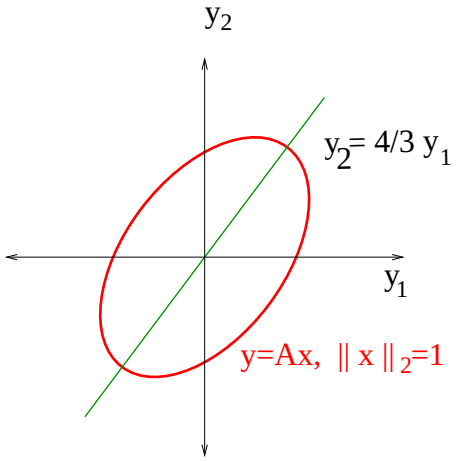


Figure 5.4.2: The ellipse E which is the image of the unit circle under the image of the matrix A

Proposition 5.1. $A = UU^*AVV^* = U\Sigma V^*$

1. *From* $AV = U\Sigma$, *one has* $Av_j = \sigma_j u_j, j = 1, \ldots, p$

   If $U = V$, *then this equation implies a eigenvalue problem. Hence it is called a generalized eigenvalue.*

2. *From* $U^*A = \Sigma V^*$, *one has* $A^*u_j = \sigma_j v_j, j = 1, \ldots, p$

σ_j : *jth singular value of* A.
u_j, v_j : *jth left and right singular vectors of* A.

$$\sigma_1 = \|A\|_2 = \sup_{x \neq 0} \frac{\|Ax\|_2}{\|x\|_2} = \sup_{\|x\|_2 = 1} \|Ax\|_2 \tag{5.20}$$

Proof of SVD Theorem 5.2. Let $\sigma_1 = \|A\|_2 = \max_{\|v\|_2=1} \|Av\|_2$. Then there exists a vector $v_1 \in \mathbb{C}^n$, with $\|v_1\|_2 = 1$ and $\|Av_1\|_2 = \sigma_1$. Put $y = Av_1 \in \mathbb{C}^m$ and $u_1 = \frac{y}{\|y\|_2} = \frac{Av_1}{\|Av_1\|_2} = \frac{Av_1}{\sigma_1} \in \mathbb{C}^m$; therefore, $Av_1 = \sigma_1 u_1 \in \mathbb{C}^m$.

Next choose, for instance, by using Gram-Schmidt orthogonalization, $U_1 \in \mathcal{M}(m, m-1)$ and $V_1 \in \mathcal{M}(n, n-1)$ such that $U = [u_1, U_1] \in \mathcal{M}(m, m)$ and $V = [v_1, V_1] \in \mathcal{M}(n, n)$ are unitary. Then, using orthonality,

$$\begin{aligned} A_1 &:= U^*AV = \begin{bmatrix} u_1^* \\ U_1^* \end{bmatrix} \begin{bmatrix} Av_1 & AV_1 \end{bmatrix} \\ &= \begin{bmatrix} u_1^*Av_1 & u_1^*AV_1 \\ U_1^*Av_1 & U_1^*AV_1 \end{bmatrix} = \begin{bmatrix} \sigma_1 & u_1^*AV_1 \\ 0 & U_1^*AV_1 \end{bmatrix}. \end{aligned} \tag{5.21}$$

Claim that $u_1^*AV_1 = \underline{0} \in \mathbb{C}^{m-1}$. Indeed, using (5.21) observe that

$$\left\| A_1 \begin{pmatrix} \sigma_1 \\ (u_1^*AV_1)^* \end{pmatrix} \right\|_2 = \left\| \begin{pmatrix} \sigma_1^2 + (u_1^*AV_1)(u_1^*AV_1)^* \\ (U_1^*AV_1)(u_1^*AV_1)^* \end{pmatrix} \right\|_2 \geq \sigma_1^2 + \|u_1^*AV_1\|_2^2,$$

which implies that

$$\|A_1\|_2 = \sup_{x \in \mathbb{C}^n} \frac{\|A_1 x\|_2}{\|x\|_2} \geq \frac{\sigma_1^2 + \|u_1^*AV_1\|_2^2}{\|(\sigma_1 \; u_1^*AV_1)\|_2} = \sqrt{\sigma_1^2 + \|u_1^*AV_1\|_2^2}.$$

Since $\|A_1\|_2 = \|U^*AV\|_2 = \|A\|_2 = \sigma_1$, we conclude that $u_1^*AV_1 = 0$. Therefore,

$$A_1 = U^*AV = \begin{pmatrix} \sigma_1 & 0 \\ 0 & U_1^*AV_1 \end{pmatrix}.$$

Observe that $\|U_1^*AV_1\|_2 \leq \|A_1\|_2 = \sigma_1$ since the matrix norm of a matrix is always greater than or equal to that of its principal submatrix (see Exercise 3.4).

Let us proceed with the $(m-1) \times (n-1)$ matrix $A^{(1)} := U_1^*AV_1$ for which there exists a vector $\widetilde{v}_2 \in \mathbb{C}^{n-1}$, with $\|\widetilde{v}_2\|_2 = 1$ and $\left\|A^{(1)}\widetilde{v}_2\right\|_2 = \sigma_2 \leq \sigma_1$. Put $\widetilde{y}_2 = A^{(1)}\widetilde{v}_2 \in \mathbb{C}^{m-1}$ and $\widetilde{u}_2 = \frac{\widetilde{y}_2}{\|\widetilde{y}_2\|_2} = \frac{A^{(1)}\widetilde{v}_2}{\left\|A^{(1)}\widetilde{v}_2\right\|_2} = \frac{A^{(1)}\widetilde{v}_2}{\sigma_2}$; therefore, $A^{(1)}\widetilde{v}_2 = \sigma_2\widetilde{u}_2$. Now, from $\widetilde{u}_2 \in \mathbb{C}^{m-1}$ and $\widetilde{v}_2 \in \mathbb{C}^{n-1}$, setting $u_2 = U_1\widetilde{u}_2 \in \mathbb{C}^m$ and $v_2 = V_1\widetilde{v}_2 \in \mathbb{C}^n$, we have $u_1^*u_2 = 0$ and $v_1^*v_2 = 0$ with $Av_2 = \sigma_2 u_2$. Choose then $U_2 \in \mathcal{M}(m, m-2)$ and $V_2 \in \mathcal{M}(n, n-2)$ such that $U^{(2)} = [u_1, u_2, U_2] \in \mathcal{M}(m,m)$ and $V^{(2)} = [v_1, v_2, V_2] \in \mathcal{M}(n,n)$ are unitary. Then, using the orthogonality and the same argument as in deriving $u_1^*AV_1 = 0$, one gets $u_j^*AV_2 = 0$ for $j = 1, 2$ (see Exercise 5.5), and therefore,

$$\begin{aligned} A_2 \;:=\; (U^{(2)})^*AV^{(2)} &= \begin{bmatrix} u_1^* \\ u_2^* \\ U_2^* \end{bmatrix} \begin{bmatrix} Av_1 & Av_2 & AV_2 \end{bmatrix} \\ &= \begin{bmatrix} u_1^*Av_1 & u_1^*Av_2 & u_1^*AV_2 \\ u_2^*Av_1 & u_2^*Av_2 & u_2^*AV_2 \\ U_2^*Av_1 & U_2^*Av_2 & U_2^*AV_2 \end{bmatrix} = \begin{bmatrix} u_1^*\sigma_1 u_1 & u_1^*\sigma u_2 & u_1^*AV_2 \\ u_2^*\sigma_1 u_1 & u_2^*\sigma_2 u_2 & u_2^*AV_2 \\ U_2^*\sigma_1 u_1 & U_2^*\sigma_2 u_2 & U_2^*AV_2 \end{bmatrix}. \\ &= \begin{bmatrix} \sigma_1 & 0 & u_1^*AV_2 \\ 0 & \sigma_2 & u_2^*AV_2 \\ 0 & 0 & U_2^*AV_2 \end{bmatrix} = \begin{bmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_2 & 0 \\ 0 & 0 & U_2^*AV_2 \end{bmatrix} \end{aligned}$$

Thus, by an induction argument

$$U^*AV = \begin{pmatrix} \sigma_1 & & & & 0 \\ & \sigma_2 & & & \\ & & \ddots & & \\ 0 & & & \sigma_p & 0 \cdots 0 \end{pmatrix}. \tag{5.22}$$

□

Exercise 5.5. *In the above proof show that* $u_j^* A V_2 = 0$ *for* $j = 1, 2$.

Corollary 5.1. *If* $A = U\Sigma V^*$ *is a SVD of* A *with* $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > \sigma_{r+1} = \cdots = \sigma_p = 0$, *the following properties hold:*

1. $A = \sum_{j=1}^{r} \sigma_j u_j v_j^* = U_r \Sigma_r V_r^*$. *Hence* $Av_k = \sigma_k u_k, k = 1, \cdots, r$.
   $U_r = [u_1, \ldots, u_r], \quad V_r = [v_1, \ldots, v_r], \quad \Sigma_r = \text{diag}\{\sigma_1, \ldots, \sigma_r\}$.
2. $\text{rank}\,(A) = r$.
3. $N(A) = \text{Span}\{v_{r+1}, \ldots, v_n\}$.
4. $R(A) = \text{Span}\{u_1, \ldots, u_r\}$.
5. $\|A\|_F = \sqrt{\sigma_1^2 + \cdots + \sigma_r^2}$.
6. $\|A\|_2 = \sigma_1$.
7. $\sigma_j = \sqrt{\lambda_j(A^*A)}, \quad j = 1, \ldots, p,$
   where $\lambda_j(A^*A)$ *is the* j*th largest eigenvalue of* A^*A.

Proof Since $v = [v_1, \cdots v_n]$ *is unitary and* $\text{span}\{v_1, \cdots, v_n\} = \mathbb{C}^n$, *we have* $x = \sum \alpha_j v_j$, *and therefore,*

$$\begin{aligned} \text{Range}(A) &= \{Ax \mid x \in \mathbb{C}^n\} \\ &= \{\sum_{j=1}^{r} \sigma_j u_j v_j^* x \mid x \in \mathbb{C}^n\} \\ &= \{\sum_{j=1}^{r} \sigma_j u_j v_j^* (\sum_{k=1}^{n} \alpha_k v_k), \alpha_k \in \mathbb{C}\} \\ &= \{\sum_{j=1}^{r} \sigma_j \alpha_j u_j \mid \alpha_j \in \mathbb{C}\}. \end{aligned}$$

2.

$$\begin{aligned}\text{Null}(A) &= \{x \in \mathbb{C}^n \mid Ax = 0\} \\ &= \{x \in \mathbb{C}^n \mid \sum_{j=1}^{r} \sigma_j u_j v_j^* x = 0\} \\ &= \{x \in \mathbb{C}^n \mid \sum_{j=1}^{r} \sigma_j u_j v_j^* \sum_{k=r+1}^{n} \alpha_k v_k = 0\} \\ &= \{x \in \mathbb{C}^n \mid x = \sum_{k=r+1}^{n} \alpha_k v_k\}\end{aligned}$$

where $x = \sum_{k=1}^{n} \alpha_k v_k$

5. $\|A\|_F = \|U_r \Sigma_r V_r^*\|_F = \|\Sigma_r\|_F = \sqrt{\sigma_1^2 + \cdots + \sigma_r^2}$ *where* u, v^* *: unitary.*

   note:: preserve F*-norm.*

Exercise 5.6. *Let* A *be an* $m \times n$ *matrix. Then show that the eigenvalues of* A^*A *are nonnegative with eigenvalues* $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n \geq 0$*. Also, denoting* $\lambda_j(A^*A) = \lambda_j, j = 1 \cdots, r$*, show that*

$$\sigma_j = \sqrt{\lambda_j(A^*A)}.$$

Exercise 5.7. *If* A *is an* $m \times n$ *matrix, find relationships between the eigenvalues and eigenvectors of*

$$B = \begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix}$$

and the singular values and singular vectors of A*.*

Exercise 5.8. *Show that if* A *is a positive-semidefinite matrix, then the singular values of* A *are the same as the eigenvalues of* A*.*

Exercise 5.9. *Prove that if* A *is a positive-definite matrix, and* $A = U\Sigma V^*$ *is a singular value decomposition of* A*, then* $U = V$*.*

Exercise 5.10. *Implement the SVD algorithm.*

Now we are ready to implement the SVD algorithm stated as in the proof of Theorem 5.2. We proceed as follows.

1. In order to compute the l^2-norm of a marix A, recall that

   $$\|A\|_2^2 = \sup_{\|x\|_2=1} \|Ax\|_2^2 = \sup_{\|x\|_2=1} (Ax, Ax) = \sup_{\|x\|_2=1} (x, A^tAx) = \sup_{\|\xi\|_2=1} (\xi, \lambda_{\max}(A^tA)\xi) =$$

   Hence, we identify the l^2-norm of A with $\sqrt{\lambda_{\max}(A^tA)}$.

2. $\lambda_{\max}(A^tA)$ can be approximated by the power iteration method.

3.

5.4.1 Shur Decomposition

Definition 5.3. *Two $t \times n$ matrices A and B are called "unitarily similar" to each other if there exist a unitary matrix U such that*

$$UA^{-1}U(= U^*AU) = B. \tag{5.23}$$

Theorem 5.3. *For $A \in \mathcal{M}(n, n)$, $\exists$ a unitary matrix U such that*

$$UA^{-1}U = U^*AU = \begin{pmatrix} \lambda_1 & & * \\ & \ddots & \\ 0 & & \lambda_n \end{pmatrix} = T; \text{ an upper triangular matrix.}$$

In other words, A and T are unitarily equivalent.

Proposition 5.2. *Every Hermitian matrix is unitarily similar to a diagonal matrix with real entries.*

Proof. Let $U^*AU = T$ be a Shur decomposition of A. Then

$$T = U^*AU = U^*A^*U = (U^*AU)^* = T^* \tag{5.24}$$

Since T is an upper-triangular matrix and $T = T^*$, T is diagonal with real entries. □

Thus, if $UA^{-1}U = \Lambda = \text{diag}\{\lambda_1, \ldots, \lambda_n\} \Longrightarrow AU = \Lambda U \Longrightarrow Au_j = \lambda_j u_j, \quad j = 1, \ldots, n$

For a general $m \times n$ matrix A, the singular value decomposition $A = U\Sigma V^*$ implies that

$$A^*A = (U\Sigma V^*)^*(U\Sigma V^*) = V\Sigma U^*U\Sigma V^*$$

and therefore,

$$A^*A = V\Sigma^2V^*; \qquad A^*AV = V\Sigma^2 \Longrightarrow A^*Av_j = \sigma_j^2 v_j.$$

Chapter 6

Least squares problems

Consider the overdetermined system from a curve fitting problem with the given data points

$$(x_k, y_k), \quad k = 1, 2, \cdots, m.$$

Let $\boldsymbol{\alpha} = (\alpha_1, \cdots, \alpha_n)^T \in \mathbb{R}^n$ be a parameter vector. Consider a function $y = f(x; \boldsymbol{\alpha})$ depending of n number of parameters of the form

$$f(x; \boldsymbol{\alpha}) = \sum_{j=1}^{n} \alpha_j \phi_j(x), \quad \text{with } \phi_j(x) \text{ are given functions.} \tag{6.1}$$

Here, $\phi_j(x), j = 1, \cdots, n$, are given such that they form a basis for an n-dimensional function space. For example, $\phi_j(x) = x^{j-1}, j = 1, \cdots$. or $\phi_j(x) = e^{(j-1)x}, j = 1, \cdots$. In usual data–fitting problems, the number of data points is larger than that of parameter, *i.e.* $m > n$.

The aim of parameter estimation is to choose a best parameter set as follows:

$$f(x_k; \boldsymbol{\alpha}) = \sum_{j=1}^{n} \alpha_j \phi_j(x_k) \sim y_k, \quad k = 1, \cdots, m, \tag{6.2}$$

which can be put in matrix notation as follows:

$$A\boldsymbol{\alpha} \simeq \mathbf{y}, \tag{6.3}$$

where $\mathbf{y} = (y_1, \cdots, y_m)^t \in \mathbb{R}^m$, and $A_{kj} = \phi_j(x_k)$. Since the $m \times n$ matrix A is usually a non-square matrix with $m > n$, one cannot expect to find $\boldsymbol{\alpha}$ which makes (6.3) as $A\boldsymbol{\alpha} = \mathbf{y}$. Instead one often expects to make the difference vector $\mathbf{y} - A\boldsymbol{\alpha}$ as small as possible in certain vector norm. When the vector norm is chosen as ℓ^2-norm, it is called a "least squares problem."

Define an object function $J(\boldsymbol{\alpha})$ by

$$J(\boldsymbol{\alpha}) = \sum_{k=1}^{m} |f(x_k;\boldsymbol{\alpha}) - y_k|^2 = \sum_{k=1}^{m} \left| \sum_{j=1}^{n} \alpha_j \phi_j(x_k) - y_k \right|^2 .$$

We try to find parameters $\alpha_1^*, \cdots, \alpha_n^*$, or a vector $\boldsymbol{\alpha}^* = (\alpha_1^*, \cdots, \alpha_n^*)^T \in \mathbb{R}^n$ such that

$$J(\boldsymbol{\alpha}^*) = \min_{\boldsymbol{\alpha}\in\mathbb{R}^n} J(\boldsymbol{\alpha}). \tag{6.4}$$

This is a classical least squares minimization problem. In order to solve this, differentiate $J(\boldsymbol{\alpha})$ with respect to $\boldsymbol{\alpha}$. Then,

$$\underline{0} = \nabla_{\boldsymbol{\alpha}} J(\boldsymbol{\alpha}), \tag{6.5}$$

whose components are given as follows:

$$\frac{\partial J}{\partial \alpha_\ell}(\boldsymbol{\alpha}) = 2\sum_{k=1}^{m} \left(\sum_{j=1}^{n} \alpha_j \phi_j(x_k) - y_k \right) \phi_\ell(x_k) = 0, \ell = 1, \cdots, n. \tag{6.6}$$

This is equivalent to

$$\sum_{j=1}^{n} \left(\sum_{k=1}^{m} \phi_j(x_k)\phi_\ell(x_k) \right) \alpha_j = \sum_{k=1}^{m} y_k \phi_\ell(x_k), \ell = 1, \cdots, n. \tag{6.7}$$

Thus, writing $S_{\ell j} = \sum_{k=1}^{m} \phi_j(x_k)\phi_\ell(x_k), j = 1, \cdots, n$, and $b_\ell = \sum_{k=1}^{m} y_k \phi_\ell(x_k), \ell = 1, \cdots, n$, we obtain the following linear system

$$S\boldsymbol{\alpha} = \mathbf{b}. \tag{6.8}$$

Notice that S is symmetric. Moreover, we have

$$\begin{aligned}
\boldsymbol{\alpha}^T S \boldsymbol{\alpha} &= \sum_{j=1}^{n}\sum_{\ell=1}^{n}\sum_{k=1}^{m} \phi_j(x_k)\phi_\ell(x_k)\alpha_\ell\alpha_j \\
&= \sum_{k=1}^{m}\sum_{j=1}^{n} \alpha_j\phi_j(x_k) \sum_{\ell=1}^{n} \alpha_\ell\phi_\ell(x_k) \\
&= \sum_{k=1}^{m} \left(\sum_{j=1}^{n} \alpha_j\phi_j(x_k) \right)^2 \\
&= \sum_{k=1}^{m} \left(f(x_k;\boldsymbol{\alpha}) \right)^2 ,
\end{aligned}$$

which vanishes if and only if

$$f(x_k; \boldsymbol{\alpha}) = 0 \quad \forall k = 1, \cdots, m.$$

Notice that if $f(x; \boldsymbol{\alpha}) = \sum_{j=1}^{n} \alpha_j \phi_j(x)$ vanishes at the $m, m \geq n$, distinct points x_k's, $f \equiv 0$, which implies that $\boldsymbol{\alpha} = \underline{0}$. This proves that S is postive-definite. We summarise the above as in the following theorem.

Theorem 6.1. *Suppose that $m \geq n$. Assume that $x_k, k = 1, \cdots, m$, are distinct. Assume that $\phi_j(x), j = 1, \cdots, n$, are given such that they form a basis for an n-dimensional function space. Let $S_{\ell j} = \sum_{k=1}^{m} \phi_j(x_k)\phi_\ell(x_k), \ell, j = 1, \cdots, n$. The matrix S in (6.8) is symmetric positive-definite.*

Using the matrix notations as given in (**??**) and (6.5), we have

$$\nabla_{\boldsymbol{\alpha}} J(\boldsymbol{\alpha}) = A^t A\boldsymbol{\alpha} - A^t \mathbf{y},$$

and hence we have the square matrix equation:

$$S\boldsymbol{\alpha} = A^t \mathbf{y}.$$

Example 6.1. *Consider that $(x_k, y_k), k = 1, \cdots, m$ are given data. Let us try to fit these data by a linear polynomial $y = \alpha_1 + \alpha_2 x$. This leads to the system of m linear equations in two unknowns α_1 and α_2 as follows:*

$$\begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_m \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}.$$

In general, consider

$$Ax = b, \quad A \in \mathbb{C}^{m \times n}, b \in \mathbb{C}^m \tag{6.9}$$

If $m > n$ (m equations in n unknowns), usually there does not exist an exact solution; thus we seek $\phi(x_{opt}) = \min_{x \in \mathbb{C}^n} \|Ax - b\|_2^2$.

Example 6.2.

$$A = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, \quad b_1 = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}, \quad b_1 \leq b_2 \leq b_3$$

1. In $\|\cdot\|_1$-norm,

$$\|Ax-b\|_1 = \left\| \begin{pmatrix} x-b_1 \\ x-b_2 \\ x-b_3 \end{pmatrix} \right\|_1 = |x-b_1| + |x-b_2| + |x-b_3|.$$

$$\therefore x = b_2.$$

2. In $\|\cdot\|_\infty$-norm (mini-max problem),

$$\min_x \|Ax-b\|_\infty = \min_x \left[\max_{j=1,2,3} |x-b_j| \right]$$

$$\therefore x = \frac{b_1+b_3}{2}$$

note:: b_1 and b_3 are end points.

3. In $\|\cdot\|_2$-norm (least squares problem),

$$\min_x \|Ax-b\|_2^2 = \min_x \sum_{j=1}^{3} (x-b_j)^2 \qquad \textit{differentiate in } x$$

$$\therefore x = \frac{b_1+b_2+b_3}{3}.$$

From now on, we consider only the least squares problems. Then,

$$\phi(x) = \|Ax-b\|_2^2 = (Ax-b)^*(Ax-b) = x^*A^*Ax - x^*A^*b - b^*Ax + \|b\|_2^2.$$

$$\nabla\phi(x) = 2[A^*Ax - A^*b] = 0.$$

We have the *normal equation*:

$$A^*(Ax-b) = 0, \tag{6.10}$$

which has a unique solution if and only if rank $(A) = n$. *

Set $\chi = \{x \in \mathbb{C}^n | \|Ax-b\|_2 = \min_{y\in\mathbb{C}^n} \|Ay-b\|_2\}$. Then the following holds:

1. $x \in \chi \Longleftrightarrow A^*(Ax-b) = 0.$

*rank $(BA) \le$ rank (A)

2. χ is a convex set, $x, y \in \chi \Longrightarrow (1-\alpha)x + \alpha y \in \chi$.
3. χ has a unique element x_{LS} with minimal 2-norm.
4. $\chi = \{x_{LS}\} \Longleftrightarrow \text{rank}\,(A) = n$, Denote

$$\rho_{LS} = \|Ax_{LS} - b\|_2.$$

Remark ::

- Optimization Problems ::[linear, nonlinear],[unconstrained,constrained].
- Linear Programming = *Constrained* linear optimization.

First, we consider the least squares problems with an $m \times n$ matrix A of rank n with $m > n$. In this case, the usual Cholesky method can solve the normal equation successfully and find the unique minimizer x_{LS}.

ALGORITHM 6.1. *Find x_{LS} and ρ_{LS}*

$A \in \mathcal{M}(m,n),\ \text{rank}\,(A) = n,\ b \in \mathbb{C}^n$

1 $C = A^*A$ $A \in \mathcal{M}(m,n)$

2 $d = A^*b$

COMPUTE THE CHOLESKY DECOMPOSITION

3 $C = GG^*$ $G \in \mathcal{M}(n,n)$ LOWER TRIANGULAR

4 Solve for y : $Gy = d$

5 Solve for x : $G^*x = y$

(The number of flops) $\simeq \frac{n^2}{2}\left(m + \frac{n}{3}\right)$

Exercise 6.1. *Consider the following least squares problem. Suppose we have the following data*

$$\{(t_j, y_j),\ y_j = \cos(t_j)\ , t_j = \frac{j}{10}\pi,\ j = 1, \cdots, 10\},$$

and we want to fit the data in the least squares sense using a 6th-order polynomial

$$y = \sum_{j=0}^{6} a_j t^j.$$

First, formulate a linear algebraic least squares problem

$$\min_{0\neq x\in\mathbb{R}^7} \|Ax-b\|_2^2,$$

where $x=(a_0,a_1,\cdots,a_6)^t$. *Then, using the above Cholesky algorithm write a program and find the coefficients* $a_0,a_1,\cdots,a_6$.

Next, we consider the least squares problems with an $m\times n$ matrix A of rank $< n$ with $m > n$. In this case, $A*A$ is not invertible and the normal equation (6.10) is not solvable. However, such a least squares problem can be solved by using the SVD as stated in the following theorem.

Theorem 6.2. *Let* $A=U\Sigma V^*$ *be a SVD of* A.

$$\begin{aligned}
\text{rank}\,(A) &= r && (6.11)\\
A &= \sum_{j=1}^{r}\sigma_j u_j v_j^* && (6.12)\\
x_{LS} &= \sum_{j=1}^{r}\frac{u_j^*b}{\sigma_j}v_j && (6.13)\\
\rho_{LS} &= \sqrt{\sum_{j=r+1}^{m}\left|u_j^*b\right|^2} && (6.14)
\end{aligned}$$

Proof. Since $U^*AV=\Sigma$ is diagonal, writing $\alpha=V^*x$

$$\begin{aligned}
\|Ax-b\|_2^2 &= \|U^*(Ax-b)\|_2^2\\
&= \|U^*AV\alpha-U^*b\|_2^2\\
&= \|(\sigma_1\alpha_1-u_1^*b,\cdots,\sigma_r\alpha_r-u_r^*b,-u_{r+1}^*b,\cdots,-u_m^*b)^t\|_2^2\\
&= \textstyle\sum_{j=1}^{r}\left|\sigma_j\alpha_j-u_j^*b\right|^2+\sum_{j=r+1}^{m}\left|u_j^*b\right|^2,
\end{aligned}$$

which has a minimum if $\alpha_j=\frac{u_j^*b}{\sigma_j}$, $j=1,\cdots,r$, and 0 for $j=r+1,\cdots,n$. with Equations 6.13 and 6.14 since $x=V\alpha=\alpha_1v_1+\cdots+\alpha_rv_r$. □

6.1 Moore-Penrose pseudo inverse $A^\dagger$ of $A\in\mathcal{M}(m,n)$.

The SVD can be used to find a pseudo inverse of a matrix which is not of square size. The Moore-Penrose pseudo inverse $A^\dagger\in\mathcal{M}(n,m)$ of $A\mathcal{M}(m,n)$ is the unique solution $\chi\in\mathcal{M}(n,m)$ such that

1. $A\chi A = A$.
2. $\chi A\chi = \chi$
3. $(A\chi)^* = A\chi \quad \in \mathcal{M}(m,m)$
4. $(\chi A)^* = \chi A \quad \in \mathcal{M}(n,n)$

Example 6.3. *If $A = U\Sigma V^*$ is a SVD of A, then $A^\dagger = V\Sigma^\dagger U^*$ where*

$$\Sigma^\dagger = \begin{pmatrix} 1/\sigma_1 & & & & \\ & 1/\sigma_2 & & & \\ & & \ddots & & \\ & & & 1/\sigma_r & \\ & & & & 0 \end{pmatrix}. \tag{6.15}$$

Definition 6.1. *Let $W_1, \cdots, W_J$ be subspaces of V. We say that $W_1, \cdots, W_J$ are independent if*

$$\sum_{j=1}^{J} v_j = 0, \quad v_j \in W_j,$$

implies that $v_j = 0$ for all $j = 1, \cdots, J$.

Lemma 6.1. *Let V be a finite dimensional vector space and $W_j, j = 1, \cdots, J$, be subspaces of V. Then $W_j, j = 1, \cdots, J$, are independent if and only if*

$$W_{j_k} \cap (W_{j_1} + \cdots, W_{j_{k-1}}) = \{0\}, \quad \forall k \le J,$$

where $\{j_1, \cdots, j_k\}$ are k distinct integers from $\{1, \cdots, J\}$. If, $\mathcal{B}_j$, is an ordered basis for W_j, for $j = 1, \cdots, J$, then $(\mathcal{B}_1, \cdots, \mathcal{B}_J)$ is an ordered basis for $W_1 + \cdots + W_J$.

Exercise 6.2. *Prove the above Lemma 6.1.*

Definition 6.2. *If V is an n-dimensional vector space, an ordered basis $\mathcal{B}$ for V is a basis for V with ordering. To denote an ordered basis, we shall abuse notation*

$$\mathcal{B} = \{\xi_1, \cdots, \xi_n\},$$

implicitly assuming that the vectors are written in order.

Proposition 6.1. *Let $\mathcal{B}_V = \{\xi_1, \cdots, \xi_n\}$ and $\mathcal{B}_W$ be ordered bases for V and W, which are n and m-dimensional vector spaces. Then there exists an one-to-one correspondence between a linear transformation $T : V \to W$ and an $m \times n$ matrix A_T and with respect the ordered bases $\mathcal{B}_V$ and $\mathcal{B}_W$. Moreover, the (j,k)th component of A_T is the coefficient of the jth component of $T\xi_k$ with respect to $\mathcal{B}_W$.*

Lemma 6.2. *If, $\mathcal{B}_j$, is an ordered basis for W_j, for $j = 1, \cdots, J$, then $(\mathcal{B}_1, \cdots, \mathcal{B}_J)$ is an ordered basis for $W_1 + \cdots + W_J$ if and only if $W_j, j = 1, \cdots, J$, are independent.*

Exercise 6.3. *Prove the above Lemma 6.2.*

Definition 6.3. *If any of the conditions of Lemma 6.1 hold, W is called the direct sum of $W_1, \cdots, W_J$, denoted by*

$$W = W_1 \oplus \cdots \oplus W_J.$$

Exercise 6.4. *Let T be a linear operator on a finite dimensional vector space V. If $W_j, j = 1, \cdots, J$, are eigenspaces associated with all the distinct eigenvectos $\alpha_j, j = 1, \cdots, J$, of T, show that $W_j, j = 1, \cdots, J$, are independent. Moreover, show that if T is diagonalizable, then $V = W_1 \oplus \cdots \oplus W_J$.*

Exercise 6.5. *If T is an idempotent linear operator on a vector space, show that $I + T$ is invertible and find the inverse.*

Theorem 6.3 ([4] p. 284)**.** *Let W be a subspace of an inner product space V. For any $x \in V$, the following hold:*

1. *$\tilde{x} \in W$ is a best approximation to x by vectors in W if and only if*

$$(\tilde{x} - x, w) = 0 \quad \forall w \in W;$$

2. *If a best approximation to x by vectors in W exists, it is unique;*
3. *If $\dim(W) = m$ and $\{\xi_1, \cdots, \xi_m\}$ is a basis for W, then the best approximation to x is given by*

$$\tilde{x} = \sum_{j=1}^{m} \frac{(x, \xi_j)}{\|\xi\|^2} \xi_j.$$

Definition 6.4 ([4] p. 285)**.** *Whenever the vector $\tilde{x}$ in Theorem 6.3 exists it is called the orthogonal projection of x on W. If every vector in V has an orthogonal projection on W, the mapping that assigns to each vector in V its orthogonal projection on W is called the orthogonal projection of V on W.*

Definition 6.5. *A linear transformation $T : X \to Y$ is a projection if $T^2 = T$. A projection $T : X \to Y$ is an orthogonal projection if $T^* = T$. Similarly an $n \times n$ matrix P can be identified as a projection if P satisfies $P^2 = P$, and as an orthogonal projection if it has the additional property $P^* = P$.*

Notice that $T^2 = T$ is meaningful only if the range $T(X)$ of T is included in X. This occurs if $Y \subset X$. The notion of orthogonal projection is that the difference $x - Px$ is orthogonal to the range space of P for all $x \in X$. In other words, for all $x, y \in X$,

$$0 = \langle x - Px, Py \rangle = \langle P^*(x - Px), y \rangle = \langle (P^* - P^*P)x, y \rangle ,$$

which implies that $(P^* - P^*P)x = 0$ for all $x \in X$. This is equivalent to

$$P^* = P^*P,$$

from which it follows that

$$P = (P^*)* = (P^*P)* = P^*P = P^*$$

since P^*P is Hermitian.

Let $P : \mathbb{C}^n \to N(A)^\perp$ and $\overline{P} : \mathbb{C}^m \to R(A)$ are orthogonal projections. Then we have

$$\mathbb{C}^n = N(A) \oplus N(A)^\perp = N(A) \oplus R(P),$$

and

$$\mathbb{C}^n = R(A) \oplus R(A)^\perp = R(\overline{P}) \oplus R(A)^\perp.$$

Thus,

$$P = P^* = P^2; Px = 0 \text{ if and only if } x \in N(A),$$
$$\overline{P} = \overline{P}^* = \overline{P}^2; \overline{P}x = 0 \text{ if and only if } x \in R(A)^\perp.$$

Property 6.1. *Then $A^\dagger A = P$ and $AA^\dagger = \overline{P}$.*

efore we begin to prove we explain the concept of pseudoinverse $A^\dagger$ in more details following [7]. Notice that for each $b \in R(A)$ there exists a

unique $x_b \in N(A)^\perp$ such that $Ax_b = b$. This defines a well-defined mapping $f : R(A) \to N(A)^\perp$ such that

$$Af(y) = y,\ f(y) \in N(A)^\perp \qquad \forall y \in R(A).$$

Indeed, if $b \in R(A)$, then there is an x such that $b = Ax$; thus

$$b = A(Px + (I-P)x) = APx + A(I-P)x = APx, \text{ since } (I-P)x \in N(A),$$

with $Px \in N(A)^\perp$. Hence if $x_1, x_2 \in N(A)^\perp, Ax_1 = Ax_2 = b$, we have

$$x_1 - x_2 \in N(A) \cap N(A)^\perp = \{0\}.$$

Thus $x_1 = x_2$.

Notice that the mapping $f : R(A) \to N(A)^\perp$ defined above is one-to-one onto function.

Using this observation, the pseudoinverse $A^\dagger$ can be understood as follows:

$$A^\dagger y = \begin{cases} f(y) & \text{for } y \forall y \in R(A), \\ 0 & \text{for } y \forall y \in R(A)^\perp. \end{cases} \tag{6.16}$$

Exercise 6.6. *Prove that the mapping $f : R(A) \to \mathbb{C}^n$ such that*

$$Af(y) = y,\ f(y) \in N(A)^\perp \qquad \forall y \in R(A).$$

is linear mapping.

Some useful and interesting properties of the pseudo-inverse are listed as follows.

Properties 6.1. *1.* $A^{\dagger\dagger} = A$.

2. $(A^\dagger)^* = (A^*)^\dagger$.

3. $\|Ax - b\|_2 \geq \left\|AA^\dagger b - b\right\|_2 \qquad x \in \mathbb{C}^n$.

4. $\|Ax - b\|_2 = \left\|AA^\dagger b - b\right\|_2$ *and* $x \neq A^\dagger b \qquad \Longrightarrow \qquad \|x\|_2 > \left\|A^\dagger b\right\|, x \neq x_{LS}$. *Thus $A^\dagger b$ is the solution of the least squares problem.*

Exercise 6.7. *Show that Properties 6.1 hold.*

Theorem 6.4. *Let $A = U\Sigma V^*$ be a SVD of $A \in \mathcal{M}(m,n)$ with $r = \text{rank}\,(A)$. Then for $k < r$,*

$$\min_{\text{rank}\,(B)=k, B\in\mathcal{M}(m,n)} \|A - B\|_2 = \left\| A - \sum_{j=1}^{k} \sigma_j u_j v_j^* \right\|_2 \tag{6.17}$$

We recall the following Dimension Theorem from Linear Algebra.

Theorem 6.5 (Dimension theorem)**.** *Let B be an $m \times n$ matrix. Then we have*

$$n = \text{rank}\,(B) + \text{nullity}(B) = \dim(R(B)) + \dim(N(B)). \tag{6.18}$$

In general, if $T : V \longrightarrow W$ is a linear transformation from a finite dimensional vector space V into a vector space W, then we have

$$n = \text{rank}\,(T) + \text{nullity}(T) = \dim(R(T)) + \dim(N(T)). \tag{6.19}$$

Proof. Suppose $\text{rank}\,(B) = k$. Then $\dim N(B) = n - k$. Let $x_1, \ldots, x_{n-k}$ be orthogonal vectors of $N(B)$. Then $\text{Span}\{x_1, \ldots, x_{n-k}\} \cap \text{Span}\{v_1, \ldots, v_{k+1}\} \neq \{0\}$ thus there exists $z \in \mathbb{C}^n$ with $\|z\|_2 = 1$, $Bz = 0$ and $z = \sum_{j=1}^{k+1} \beta_j v_j$. Hence $Az = \sum_{j=1}^{k+1} \beta_j A v_j = \sum_{j=1}^{k+1} \beta_j \left(\sum_{l=1}^{r} \sigma_l u_l v_l^*\right) v_j = \sum_{j=1}^{k+1} \beta_j \sigma_j u_j$. Then

$$\begin{aligned}
\|A - B\|_2^2 &\geq \|(A - B)z\|_2^2 = \|Az\|_2^2 \\
&= \left\|\sum_{j=1}^{k+1} \beta_j \sigma_j u_j\right\|_2^2 \\
&= \sum_{j=1}^{k+1} \sigma_j^2 \|\beta_j u_j\|_2^2 \qquad u_j\text{'s are orthogonal} \\
&\geq \sigma_{k+1}^2 \sum_{j=1}^{k+1} \|\beta_j u_j\|_2^2 \\
&= \sigma_{k+1}^2 \sum_{j=1}^{k+1} \beta_j^2 \\
&= \sigma_{k+1}^2,
\end{aligned}$$

since $\sum_{j=1}^{k+1} \beta_j^2 = 1$. Consequently, we have

$$\|A - B\|_2 \geq \sigma_{k+1},$$

and the equality holds if we take $B = \sum_{j=1}^{k} \sigma_j u_j v_j^*$. This proves our theorem. □

Consider

$$\min_{x \in \mathbb{R}^n} \|Ax - b\|, \quad A \in \mathcal{M}(m, n), \quad b \in \mathbb{R}^m \tag{6.20}$$

$$rank(A) = n, \quad m \geq n \tag{6.21}$$

Recall that the Householder matrices (see (5.3)):

$$H_j = I - 2\frac{v_j v_j^t}{v_j^t v_j}$$

such that the following QR-decomposition holds:

$$\underbrace{H_n H_{n-1} \cdots H_1}_{Q^t} A = \begin{pmatrix} \ddots & & & * \\ & \ddots & & \\ & & \ddots & 0 \\ 0 & \cdots & \cdots & 0 \end{pmatrix} = \begin{pmatrix} R \\ 0 \end{pmatrix}$$

from the algorithm

Algorithm 6.2.

```
1 A^(0) = A; b^(0) = b
2 do j=1,n
3    A^(j) = H_j A^(j-1)
4    b^(j) = H_j b^(j-1)
5 enddo
```

We then have

$$A^{(n)} = \begin{pmatrix} R \\ \mathbb{O} \end{pmatrix} \text{ with } R \in \mathcal{M}(m,m), \mathbb{O} \in \mathcal{M}(m-n,n) \tag{6.22a}$$

$$b^{(n)} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \text{ with } b_1 \in \mathbb{R}^n, b_2 \in \mathbb{R}^{m-n}. \tag{6.22b}$$

Since Q is an orthogonal matrix, we have

$$\begin{aligned} \|Ax-b\|_2 &= \left\|Q^t(Ax-b)\right\|_2 \\ &= \left\| \begin{pmatrix} R \\ \mathbb{O} \end{pmatrix} x - \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \right\|_2 \\ &= \left\| \begin{pmatrix} Rx \\ \mathbb{O} \end{pmatrix} - \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \right\|_2 \\ &= \sqrt{\|Rx-b_1\|_2^2 + \|b_2\|_2^2} \end{aligned}$$

Hence as the solution of the least squares problem, one has

$$x = x_{LS} = Rb_1^{-1}, \tag{6.23}$$

$$\rho_{LS} = \|b_2\|_2. \tag{6.24}$$

Notice that the $n \times n$ matrix R is invertible since $\text{rank}\,(A) = \text{rank}\,(R) = n$.

Chapter 7

Richardson-type iterative methods for linear systems

7.1 The (Gauss-) Jacobi method and Seidel method

7.1.1 The Jacobi method

Given an $n \times n$ matrix A, let

$$\begin{aligned} P &= D = \mathrm{diag}\{a_{11}, a_{22}, \ldots, a_{nn}\} && \text{(7.1a)} \\ N &= -(A - D) && \text{(7.1b)} \end{aligned}$$

so that $A = P - N$ or $N = P - A$. In order to solve $Ax = b$, one can write $Px = Nx + b$. Then,

$$\begin{aligned} x &= P^{-1}(Nx + b) = P^{-1}((P - A)x + b) \\ &= (I - P^{-1}A)x + P^{-1}b && \text{(7.2a)} \\ &= x + P^{-1}(b - Ax) = x + P^{-1}r, && \text{(7.2b)} \end{aligned}$$

where $r = b - Ax$ denotes the residual. From this several iterative schemes can be generated by

$$\begin{aligned} x^{(j+1)} &= P^{-1}(Nx^{(j)} + b) \\ &= P^{-1}((P - A)x^{(j)} + b) && \text{(7.3a)} \\ &= (I - P^{-1}A)x^{(j)} + P^{-1}b && \text{(7.3b)} \\ &= x^{(j)} + P^{-1}(b - Ax^{(j)}) = x^{(j)} + P^{-1}r^{(j)}, && \text{(7.3c)} \end{aligned}$$

depending on the choice of P and N

The Jacobi method is an iterative method with initial approximation $x^{(0)}$.

Set $A = L + D + U$ where L and U are the strictly lower and upper triangular parts of the matrix A and D the diagonal part of the matrix A. Let $r^{(j)} = b - Ax^{(j)}$ be the jth residual. We assume that D is invertible and certain conditions for A such that the Jacobi iteration scheme converges. Some necessary and sufficient conditions to ensure convergence will be discussed later.

ALGORITHM 7.1. *Jacobi scheme*

```
1 do j=1, max_iter
2    x^(j) = x^(j-1) + P^{-1}(b - Ax^(j-1))        Dx^(j) + (L + U)x^(j-1) = b
3 enddo
```

In practice the above algorithm can be implemented as follows:

ALGORITHM 7.2. *practical Jacobi scheme*

```
1 Initial approximation x(:)
2 do j=1, max_iter
3    do k=1, n
4       x_next(k) = (b(k) - sum_{l=1}^{k-1} a(k,l)x(l) - sum_{l=k+1}^{n} a(k,l)x(l)) / a(k,k)
5    enddo
6    x = x_next
7 enddo
```

In Fortran 90/95 version,

ALGORITHM 7.3. *Jacobi scheme F90/95 versrion*

```
1 Initial approximation x(:)
2 do j=1, max_iter
3    do k=1, n
4       x_next(k) = (b(k) - dot_product(a(k,1:k-1),x(1:k-1)) - dot_product(a(k,k+1:n),x(k+1:n))) / a(k,k)
5    enddo
6    x = x_next
7 enddo
```

F90 Program 7.1: Jacobi method

```
!Jacobi iteration scheme to find the solution of -u" = f, (0,1); u(0)=u(1)=0
!The RHS is given such that the solution is u(x_j) = (n+1-j)*h, j = 1, n
! where h is the mesh size and
! D. Sheen http://www.nasc.snu.ac.kr
program jacobi
  implicit none
  real(8),allocatable, dimension(:):: x, xn, b, err
  real(8)::l2norm
  integer:: n, j, iter, max_iter

  write(6,*) "What is the dimension of the vector?"
  read(5,*) n
  write(6,*) "What is the maximum number of iteration?"
  read(5,*) max_iter

  allocate( x(0:n), xn(0:n), b(1:n-1), err(1:n-1) )

  b=0.d0; b(1)=real(n) ! the exact solution should be n, n-1, n-2,\cdots, 2,1
  x = 1.d0
  x(0)=0.d0; x(n)=0.d0
  xn(0)=0.d0; xn(n)=0.d0

  do iter = 1, max_iter
     do j = 1, n-1
        xn(j) = (b(j) + x(j-1) + x(j+1) )/2.d0
     enddo
     x = xn ! update for the next iteration

     if ( mod(iter, 10) == 0) then
        do j = 1, n-1
           err(j) = xn(j) - real(n-j) ! use x(j) for the error calculation
        end do
        l2norm=dot_product(err(1:n-1),err(1:n-1)); l2norm = sqrt(l2norm)
        write(6,921) iter, l2norm
        if (l2norm < 1.d-6) then
           write(6,*) "The Jacobi iteration converged at iter < ", iter
           exit
        end if
     end if
  enddo
  print*, "The solution x(0:n) is given as follows:"
  print*, xn
921  format("iter =",i6, ";   l2norm =", f25.15)
end program jacobi
```

Exercise 7.1. *Implement the above Jacobi method and solve the two-dimensional elliptic problem Example 2.5.*

7.1.2 Seidel method

Seidel method is given in the following decomposition with P and N as follows:

$$P = D + L \tag{7.4a}$$
$$N = -(A - P) = -U \tag{7.4b}$$

ALGORITHM 7.4. *Seidel scheme*

1 $x^{(0)}$ is initial approximation
2 do j=1, max_iter
3 do k=1, n
4 $x_k{}^{(j)} = \left(b_k - \sum_{l=1}^{k-1} a_{kl} x_l{}^{(j)} - \sum_{l=k+1}^{n} a_{kl} x_l{}^{(j-1)}\right) / a_{kk}$ $(D+L)x^{(j)} + Ux^{(j-1)} = b$
5 enddo
6 enddo

Notice that in the evaluation of the kth component at jth iteration step, the 1th, 2nd, $\cdots$, $k-1$th components that updated at the current iteration step are immediately used. Then the kth component is used successively in updating $k+1$th, $k+2$nd, $\cdots$, nth components. Based on this observation, Seidel algorithm is usually implemented as follows:

ALGORITHM 7.5.

1 Initial approximation $x(:)$
2 do j=1, max_iter
3 do k=1, n
4 $x(k) = \frac{b(k) - \sum_{l=1}^{k-1} a(k,l)x(l) - \sum_{l=k+1}^{n} a(k,l)x(l)}{a(k,k)}$
5 enddo
6 enddo

In Fortran 90/95 version,

ALGORITHM 7.6.

1 Initial approximation $x(:)$

2 do j=1, max_iter

3 do k=1, n

4 $x(k) = \frac{b(k) - dot\_product(a(k,1{:}k-1), x(1{:}k-1)) - dot\_product(a(k,k+1{:}n), x(k+1{:}n))}{a(k,k)}$

5 enddo

6 enddo

F90 Program 7.2: Seidel method

```
!Symmetric Seidel iteration scheme to find the solution of -u" = f, (0,1); u(0)=u(1)=0
!The RHS is given such that the solution is u(x_j) = (n+1-j)*h, j = 1, n
! where h is the mesh size and
! D. Sheen http://www.nasc.snu.ac.kr
program seidel
  implicit none
  real(8),allocatable, dimension(:):: x, b, err
  real(8)::l2norm
  integer:: n, j, iter, max_iter

  write(6,*) "What is the dimension of the vector?"
  read(5,*) n
  write(6,*) "What is the maximum number of iteration?"
  read(5,*) max_iter

  allocate( x(0:n), b(1:n-1), err(1:n-1))

  b=0.d0; b(1)=real(n) ! the exact solution should be n, n-1, n-2,\cdots, 2,1
  x = 1.d0
  x(0)=0.d0; x(n)=0.d0

  do iter = 1, max_iter
     do j = 1, n-1
        x(j) = (b(j) + x(j-1) + x(j+1) )/2.d0
     enddo
     do j = n-1, 1, -1
        x(j) = (b(j) + x(j-1) + x(j+1) )/2.d0
     enddo

     if ( mod(iter, 10) == 0) then
        do j = 1, n-1
           err(j) = x(j) - real(n-j) ! use x(j) for the error calculation
        end do
        l2norm=dot_product(err(1:n-1),err(1:n-1)); l2norm = sqrt(l2norm)
        write(6,921) 2*iter, l2norm
        if (l2norm < 1.d-6) then
           write(6,*) "The Seidel iteration converged at iter < ", 2*iter
           exit
        end if
     end if
  end do
  print*, "The solution x(0:n) is given as follows:"
  print*, x

```

```
45 921  format("iter =",i6, ";   l2norm =", f25.15)
46
47 end program seidel
```

Exercise 7.2. *Implement the above Seidel method and solve the two-dimensional elliptic problem Example 2.5.*

7.1.3 SOR: Successive Overrelaxation Scheme

With the choice of P we have the jth iterative vector $x^{(j)} = (I - P^{-1}A)x^{(j-1)} + P^{-1}b$. A different choice of P will result in significantly different convergence depending on A. Also a modification of iterative vector may provide significant improvement in convergence.

With a parameter ω, let us consider the following modification of the jth iterative vector

$$x_k{}^{(j+1)} = \omega \left(b_k - \sum_{l=1}^{k-1} a_{kl} x_l{}^{(j+1)} - \sum_{l=k+1}^{n} a_{kl} x_l{}^{(j)} \right) / a_{kk} + (1-\omega) x_k{}^{(j)} \quad (7.5)$$

Notice that this can be written as

$$a_{kk} x_k{}^{(j+1)} + \omega \sum_{l=1}^{k-1} a_{kl} x_l{}^{(j+1)} = \omega \left(b_k - \sum_{l=k+1}^{n} a_{kl} x_l{}^{(j)} \right) + (1-\omega) a_{kk} x_k{}^{(j)}.$$

In vector form, we then have

$$(D + \omega L)x^{(j+1)} = w(b - Ux^{(j)}) + (1-\omega)Dx^{(j)},$$

or equivalently,

$$P_\omega x^{(j+1)} = N_\omega x^{(j)} + b,$$

with

$$P_\omega = \frac{1}{\omega}(D + \omega L), \quad N_\omega = \frac{1}{\omega}\{(1-\omega)D - \omega U\}$$

Notice that the scheme is

$$\begin{cases} \text{Seidel scheme} & \text{if } \omega = 1, \\ \text{stationary} & \text{if } \omega = 0, \\ \text{relaxed scheme,} & \text{otherwise.} \end{cases}$$

F90 Program 7.3: SOR method

```
!Sor iteration scheme to find the solution of -u" = f, (0,1); u(0)=u(1)=0
!The RHS is given such that the solution is u(x_j) = (n+1-j)*h, j = 1, n
! where h is the mesh size and
! D. Sheen http://www.nasc.snu.ac.kr
program sor

  implicit none
  real(8), allocatable, dimension(:):: b, x, xt, err
  real(8):: res, omega, eps, pi, l2norm
  integer:: n, j, iter, max_iter
  logical:: ssor, optimal, residual

  ssor = .false.;   optimal = .true.;   residual = .true.

  pi = atan(1.d0)*4.d0
  write(6,*) "What is the dimension of the vector"
  read(5,*) n
  write(6,*) "What is the max_iter?"
  read(5,*) max_iter
  write(6,*) "What is the stopping residual?"
  read(5,*) eps
  write(6,*) "What is omega?"
  read(5,*) omega
  ! omega = 2/ (1 + sqrt( 1-\rho(M_J)**2));
  ! \rho(M_J) = 1 - \frac12 ( \rho(A))
  ! \rho (A) = 2.d0* (1.d0 - (cos( real(n-1)*pi/real(n))))
  omega = 2.d0/ (1.d0 + sqrt(1.d0 - (cos( real(n-1)*pi/real(n)))**2) )
  write(6,*) "Optimal omega = ", omega

  allocate(b(1:n-1), x(0:n), xt(0:n), err(1:n-1))

  x = 0.d0; xt = 0.d0
  b = 0.d0; b(1) = dble(n)

  x = 1.d0
  x(0)=0.d0; x(n)=0.d0

  do iter = 1, max_iter
     do j = 1, n-1
        x(j) = omega*(b(j) + x(j-1) + x(j+1) ) / 2.d0 + (1.d0-omega)*x(j)
     enddo
     if (ssor) then
        do j = n-1, 1, -1
           x(j) = omega*(b(j) + x(j-1) + x(j+1) ) / 2.d0 + (1.d0-omega)*x(j)
        enddo
     end if
     if ( mod(iter, 10) == 0) then
        if(residual) then
           res = 0.d0
           do j = 1, n-1
              res = max(res, abs( b(j) + x(j-1) - 2.d0 * x(j) + x(j+1) ) )
           enddo
           if (ssor) then
              print*, "iter = ", 2*iter,  "; maximum  residual = ", res
           else
```

```
56                print*, "iter = ", iter,   ";     maximum residual = ", res
57             endif
58             if (res .lt. eps) stop "Converged"
59          else !Calculate l^2-error
60             do j = 1, n-1
61                err(j) = x(j) - real(n-j) ! use x(j) for the error calculation
62             end do
63             l2norm=dot_product(err(1:n-1),err(1:n-1)); l2norm = sqrt(l2norm)
64             write(6,921) 2*iter, l2norm
65             if (l2norm < 1.d-6) then
66                write(6,*) "The Seidel iteration converged at iter < ", 2*iter
67                exit
68             end if
69          end if
70       end if
71    enddo
72
73 921   format("iter =",i6, ";    l2norm =", f25.15)
74
75 end program sor
```

7.1.4 SSOR: Symmetrized Successive Overrelaxation Scheme

SSOR is the Symmetrized SOR in the following fashion. Fist, update the vector x, from the first to last components. In the second step update it from the last to first components. With a parameter ω, let us consider the following modification of the jth iterative vector

ALGORITHM 7.7.

1 do $j = 0, max\_iter$

2 do k=1,n

3 $x_k = \omega \left(b_k - \sum_{l=1}^{k-1} a_{kl} x_l - \sum_{l=k+1}^{n} a_{kl} x_l \right) / a_{kk} + (1-\omega) x_k$

4 enddo

5 do k=n,1,-1

6 $x_k = \omega \left(b_k - \sum_{l=1}^{k-1} a_{kl} x_l - \sum_{l=k+1}^{n} a_{kl} x_l \right) / a_{kk} + (1-\omega) x_k$

7 enddo

8 enddo

Exercise 7.3 (Successive overrelaxation (SOR) method and Symmetrized

Successive overrelaxation (SSOR) method).

$$A = \begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & & \ddots & & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{pmatrix}$$

$n = 100, 1000, 10000$, $\omega = 1.7, 1$, ${x_j}^{(0)} = (-1)^j$. *Check the change of residual using* $\|\cdot\|_\infty$.

7.2 Richardson iterative methods

For some easily invertible matrix P, one has the following Richardson iterative scheme:

$$x^{(j)} = x^{(j-1)} + \alpha_j P^{-1} r^{(j-1)} \qquad j\text{th iterative step.} \tag{7.6}$$

The iterative schemes that have been introduced can be regarded as Richardson iterative scheme as follows:

Jacobi method $\alpha_j = 1 \quad \forall j \qquad P = D$

Seidel method $\alpha_j = 1 \quad \forall j \qquad P = D + L$

JOR* method $\alpha_j = \omega \quad \forall j \qquad P = D$

SOR method $\alpha_j = \omega \quad \forall j \qquad P = D + L$

If $\alpha_j =$ constant $\forall j$, the scheme is called "stationary Richardson method," otherwise it is called an "dynamic Richardson method."

Turn to discuss the convergence condition for the Richardson iterative method. Set the error at the jth step iteration

$$e^{(j)} = x^{(j)} - x.$$

Then,

$$\begin{array}{rrcccl} & x^{(j)} & = & x^{(j-1)} & + & \alpha_j P^{-1}(b - Ax^{(j-1)}) \\ -) & x & = & x & + & \alpha_j P^{-1}(b - Ax) \\ & e^{(j)} & = & e^{(j-1)} & - & \alpha_j P^{-1} A e^{(j-1)} \qquad = (I - \alpha_j P^{-1} A) e^{(j-1)} \end{array}$$

Thus, $e^{(j)} = (I - \alpha_j P^{-1}A)e^{(j-1)} = (I - \alpha_j P^{-1}A)(I - \alpha_{j-1}P^{-1}A)\cdots(I - \alpha_1 P^{-1}A)e^{(0)}$, and hence,

$$\left\|e^{(j)}\right\| \leq \prod_{k=1}^{j} \left\|I - \alpha_k P^{-1}A\right\| \left\|e^{(0)}\right\|.$$

Since $\left\|I - \alpha_j P^{-1}A\right\| < 1$ iff $\rho(I - \alpha_j P^{-1}A) < 1$, we have the following theorem.

Theorem 7.1. *The Richardson iterative scheme is convergence if* $\left\|I - \alpha_j P^{-1}A\right\| < 1$ *for all* j *or if* $\rho(I - \alpha_j P^{-1}A) < 1$ *for all* j. *In particular, the stationary iterative scheme is convergence if* $\left\|I - \alpha P^{-1}A\right\| < 1$ *or if* $\rho(I - \alpha P^{-1}A) < 1$.

We therefore have the following theorem:

Theorem 7.2. *The stationary Richardson method is convergent iff*

$$2\alpha \frac{\mathrm{Re}\,(\lambda)}{\alpha^2|\lambda|^2} > 1 \qquad \lambda \in \sigma(P^{-1}A) \tag{7.7}$$

Proof.

$$\begin{aligned}
&\rho(I - \alpha P^{-1}A) < 1 \\
&\iff (1 - \alpha\,\mathrm{Re}\,(\lambda))^2 + (\alpha\,\mathrm{Im}\,(\lambda)^2 < 1 \qquad \forall \lambda \in \sigma(P^{-1}A) \\
&\iff \alpha^2|\lambda|^2 < 2\alpha\,\mathrm{Re}\,(\lambda) \qquad \forall \lambda \in \sigma(P^{-1}A) \\
&\iff \frac{2\alpha\,\mathrm{Re}\,(\lambda)}{\alpha^2|\lambda|^2} > 1 \qquad \forall \lambda \in \sigma(P^{-1}A)
\end{aligned}$$

□

Theorem 7.3. *Assume that* $P^{-1}A$ *has positive real eigenvalues* $0 < \lambda_1 \leq \cdots \leq \lambda_N$. *Then the stationary Richardson method converges iff*

$$0 < \alpha < \frac{2}{\lambda_N}. \tag{7.8}$$

Moreover, $\rho(I - \alpha P^{-1}A) < 1$ *is minimized if* $\alpha = \frac{2}{\lambda_1 + \lambda_N}$ *with the value* $\frac{\lambda_N - \lambda_1}{\lambda_N + \lambda_1}$.

Proof. The eigenvalues of $I - \alpha P^{-1}A$ are $1 - \alpha\lambda_j,\ j = 1, \ldots, N$. In order to have $|1 - \alpha\lambda_j| < 1,\ \forall j$, one has the necessary and sufficient condition

$$\alpha < \frac{2}{\lambda_j}, \qquad \forall j$$

Thus the method converges if and only if $\alpha < 2/\lambda_N$.

$\rho(I - \alpha P^{-1}A)$ is minimized if $1 - \alpha\lambda_N = -(1 - \alpha\lambda_1)$ or $\alpha = 2/(\lambda_1 + \lambda_N)$. In this case, $\rho(I - \alpha P^{-1}A) = 1 - \frac{2}{\lambda_1+\lambda_N}\lambda_1 = \frac{\lambda_N-\lambda_1}{\lambda_N+\lambda_1}$ □

Definition 7.1. *An $n \times n$ matrix A is called diagonally dominant if $|a_{jj}| \geq \sum_{k\neq j} |a_{jk}|,\ \forall j$ and there exists at least one row j such that $|a_{jj}| > \sum_{k\neq j} |a_{jk}|$.*

Definition 7.2. *An $n \times n$ matrix A is called strictly diagonally dominant if $|a_{jj}| > \sum_{k\neq j} |a_{jk}|,\ \forall j$.*

Proposition 7.1. *For an $n \times n$ matrix A, the following are equivalent.*

1. *A is strictly diagonally dominant.*
2. *$|a_{jj}| > \sum_{k\neq j} |a_{jk}|,\ \forall j$.*
3. *$\left\|I - D^{-1}A\right\|_\infty < 1$.*

Proof. The first two assertions are nothing but the definition. Notice that

$$I - P^{-1}A = I - D^{-1}A = \begin{pmatrix} 0 & \frac{a_{12}}{a_{11}} & \cdots & \frac{a_{1n}}{a_{11}} \\ \frac{a_{21}}{a_{22}} & 0 & \cdots & \frac{a_{2n}}{a_{22}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{a_{n1}}{a_{nn}} & \frac{a_{n2}}{a_{nn}} & \cdots & 0 \end{pmatrix}.$$

From this and the definition of $\|\cdot\|_\infty$, one has $\left\|I - P^{-1}A\right\|_\infty = \max_j \sum_{k\neq j} \left|\frac{a_{jk}}{a_{jj}}\right| < 1$ as an equivalent condition for the strictly diagonally dominance of A. □

Theorem 7.4. *If A is strictly diagonally dominant, then the Jacobi method converges.*

Proof. $P = D$, $\alpha_j = \alpha = 1$. Thus the Jacobi method converges. □

Theorem 7.5. *The Seidel method converges if A is strictly diagonally dominant.*

Proof. Suppose that A is strictly diagonally dominant such that $|a_{jj}| > \sum_{k\neq j} |a_{jk}|,\ \forall j$.

Since we have the Seidel method gives the jth step error

$$e_k{}^{(j)} = -\sum_{l=1}^{k-1} \frac{a_{kl}}{a_{kk}} e_l{}^{(j)} - \sum_{l=k+1}^{n} \frac{a_{kl}}{a_{kk}} e_l{}^{(j-1)} \qquad k = 1, \ldots, n,$$

we have

$$|e_k{}^{(j)}| \le \sum_{l=1}^{k-1} \left|\frac{a_{kl}}{a_{kk}}\right| \ |e_l{}^{(j)}| + \sum_{l=k+1}^{n} \left|\frac{a_{kl}}{a_{kk}}\right| \ |e_l{}^{(j-1)}| \qquad k = 1, \ldots, n$$

Denote $r = \|I - D^{-1}A\|_\infty < 1$. Thus we need to show that

$$\left\|e^{(j)}\right\|_\infty \le r \left\|e^{(j-1)}\right\|_\infty \qquad \forall k. \tag{7.9}$$

In order to show this for each kth component of $e^{(j)}$, we start with $k = 1$.

$$\begin{aligned} |e_1{}^{(j)}| &= \left| -\sum_{l=2}^{n} \frac{a_{1l}}{a_{11}} e_l{}^{(j-1)} \right| \le \sum_{l=2}^{n} \left|\frac{a_{1l}}{a_{11}}\right| \left\|e^{(j-1)}\right\|_\infty \\ &\le \|I - D^{-1}A\|_\infty \left\|e^{(j-1)}\right\|_\infty = r \left\|e^{(j-1)}\right\|_\infty \end{aligned}$$

Inductively, assume that $|e_l{}^{(j)}| \le r\left\|e^{(j-1)}\right\|_\infty$ for $l = 1, \ldots, k-1$. Then, using $0 \le r < 1$,

$$\begin{aligned} |e_k{}^{(j)}| &\le \sum_{l=1}^{k-1} \left|\frac{a_{kl}}{a_{kk}}\right| |e_l{}^{(j)}| + \sum_{l=k+1}^{n} \left|\frac{a_{kl}}{a_{kk}}\right| |e_l{}^{(j-1)}| \\ &\le \left(\sum_{l=1}^{k-1} \left|\frac{a_{kl}}{a_{kk}}\right|\right) r \left\|e^{(j-1)}\right\|_\infty + \left(\sum_{l=k+1}^{n} \left|\frac{a_{kl}}{a_{kk}}\right|\right) \left\|e^{(j-1)}\right\|_\infty \\ &\le \left\|I - D^{-1}A\right\|_\infty \left\|e^{(j-1)}\right\|_\infty. \end{aligned}$$

This then shows inductively that, for all $k = 1, \cdots, n$,

$$|e_k{}^{(j)}| \le \left\|I - D^{-1}A\right\|_\infty \left\|e^{(j-1)}\right\|_\infty.$$

In other word

$$\left\|e^{(j)}\right\|_\infty \le r \left\|e^{(j-1)}\right\|_\infty. \tag{7.10}$$

Therefore the Seidel scheme is convergent. □

Indeed, we have the following theorem.

Theorem 7.6. *If A is diagonally dominant, then both Jacobi and Seidel method are convergent.*

Theorem 7.7. *Let A be a Hermitian matrix with positive diagonal entries, then the Seidel method is convergent. $\Longleftrightarrow$ A is positive-definite.*

Theorem 7.8. *Let A be a symmetric matrix with positive diagonals. The SOR is convergent iff A is positive-definite and $\omega \in (0, 2)$.*

Chapter 8

Projection methods

In this chapter, recalling that the residual is defined as $r^{(j)} := b - Ax^{(j)}$, we will follow [6].

8.1 General theory

Notice that a two-term iterative scheme can be written always as

$$x^{(j+1)} = x^{(j)} + d^{(j)},$$

where $d^{(j)}$ may be understood as the difference or direction vector from the current search to the next search. Accordingly, the residual can be written as a two-term iterative formula:

$$r^{(j+1)} = r^{(j)} - Ad^{(j)},$$

since

$$r^{(j+1)} = b - Ax^{(j+1)} = b - A\left(x^{(j)} + d^{(j)}\right) = r^{(j)} - Ad^{(j)}, \tag{8.1}$$

Let K and L be two m-dimensional subspaces of $\mathbb{R}^n$. A projection method can be described as a two-term iterative method: given the previous iterate $x^{(j)}$, search the new iterate $x^{(j+1)} = x^{(j)} + d^{(j)}$ such that

$$d^{(j)} \in K \text{ and } r^{(j+1)} \perp L. \tag{8.2a}$$

Let $v_1, \cdots, v_m$ and $w_1, \cdots, w_m$ be the bases for $K \subset \mathbb{R}^n$ and $L \subset \mathbb{R}^n$, respectively. Define then the $n \times m$ matrices $V := [v_1. \cdots, v_m]$ and $W := [w_1. \cdots, w_m]$. Then the projection method with a search vector in K can be written in the form

$$x^{(j+1)} = x^{(j)} + Vy^{(j)}, \tag{8.3}$$

where the coefficients of $y^{(j)} \in \mathbb{R}^m$ are suitable coefficients for $v_1, \cdots, v_m$ such that

$$\sum_{k=1}^{m} y_k^{(j)} v_k = d^{(j)} \in K.$$

The condition that the residual vector $r^{(j+1)} = b - Ax^{(j+1)} = b - Ax^{(j)} - AVy^{(j)} = r^{(j)} - AVy^{(j)}$ be orthogonal to L is equivalent to $(r^{(j+1)}, w) = 0$ for all $w \in L$, which can be written in matrix form as follows:

$$0 = W^T r^{(j+1)} = W^T (r^{(j)} - AVy^{(j)}). \tag{8.4}$$

Hence if $W^T AV$ is invertible, the coefficient $y^{(j)}$ in (8.3) can be found by the following formula:

$$y^{(j)} = (W^T AV)^{-1} W^T r^{(j)}. \tag{8.5}$$

8.1.1 Error projection method

Error projection methods are obtained for SPD matrices A with $L = K$. Assume that A is SPD and $K = L$. Then *an error projection method* is a two-term iterative scheme such that

$$x^{(j+1)} = x^{(j)} + d^{(j)},$$

with

$$d^{(j)} \in K, \quad r^{(j+1)} \perp K.$$

We proceed with the following proposition.

Proposition 8.1. *Suppose that A is positive-definite and $K = L$. Then $B = W^T AV$ is nonsingular.*

Proof. Recalling $V = [v_1, \cdots, v_m], W = [w_1. \cdots, w_m]$, $K = L$ implies that there exist g_{kj}'s such that $w_j = \sum_{k=1}^{m} v_k g_{kj}$. Writing $G = g_{kj}$, we have $W = VG$.. Obviously the $m \times m$ matrix G is nonsingular. Since A is positive-definite and V is of full rank m, $V^T AV$ is an $m \times m$ positive-definite matrix. Thus $B = W^T AV = G^T (V^T AV)$ is invertible. □

The above Proposition 8.1 implies that in the error projection method the vector $d^{(j)}$ can be updated by (8.3) and (8.5).

Definition 8.1. *If A is SPD, the $A-$inner product and $A-$norm are defined by*

$$\langle x, y\rangle_A = (Ax, y), \qquad \|x\|_A = \sqrt{\langle x, x\rangle_A}.$$

Also, the A-orthognality is denoted by $\perp_A$.

Next, observe that the condition $r^{(j+1)} \perp K$ implies that

$$(r^{(j+1)}, v) = 0 \quad \forall v \in K,$$

which is equivalent to

$$\left\langle (x - x^{(j+1)}), v \right\rangle_A = 0 \quad \forall v \in K \tag{8.6}$$

since $b = Ax$. Thus,

$$r^{(j+1)} \perp K \Leftrightarrow e^{(j+1)} \perp_A K.$$

Hence the error $e^{(j+1)} = x^{(j+1)} - x$ is orthogonally projected onto AK. That is,

$$(e^{(j+1)}, Av) = \left\langle e^{(j+1)}, v \right\rangle_A = 0 \quad \forall v \in K. \tag{8.7}$$

In this sense, the method is called an "error projection method."

Proposition 8.2. *Assume that A is SPD and $K = L$. Then the $x^{(j+1)} = x^{(j)} + d^{(j)}$ with $d^{(j)} \in K$ solves the error projection method onto K iff $x^{(j+1)}$ minimizes the A-norm of the error over $x^{(j)} + K$.*

Proof. Consider for any $v \in K$,

$$J(v) = \|x - (x^{(j)} + v)\|_A^2.$$

Notice that $d^{(j)} \in K, r^{(j+1)} \perp K$ is equivalent to

$$(r^{(j+1)}, v - d^{(j)}) = (A(x - x^{(j+1)}), v - d^{(j)}) = \left\langle x - x^{(j+1)}), v - d^{(j)} \right\rangle_A = 0 \, \forall v \in K.$$

Using this, $x^{(j+1)} = x^{(j)} + d^{(j)}$, the A-orthogonality (8.6), we have

$$\begin{aligned} J(v) &= \big(A(x - x^{(j)} - v), (x - x^{(j)} - v)\big) \\ &= \left\langle (x - x^{(j+1)}) - (v - d^{(j)}, (x - x^{(j+1)}) - (v - d^{(j)}) \right\rangle_A \\ &= \left\langle (x - x^{(j+1)}), (x - x^{(j+1)})\big) + \big(A(v - d^{(j)}), (v - d^{(j)})\right\rangle_A, \end{aligned}$$

which will be minimized if $v = d^{(j)}$. In other words, the minimum of $J(v)$ is taken as $\|x - x^{(j+1)}\|_A^2$ with $v = d^{(j)}$. □

8.1.2 Residual projection method

Instead, if A is nonsingular, by setting $L = AK$, residual projection methods are derived.

Proposition 8.3. *Suppose that A is nonsingular and $L = AK$. Then $B = W^T AV$ is nonsingular.*

Proof. Notice that $L = AK$ is equivalent to

$$[w_1, \cdots, w_m] = A\,[v_1. \cdots, v_m] = [Av_1, \cdots, Av_m]\,.$$

Similarly to the proof of Proposition 8.1, there exists an $m \times m$ invertible matrix G such that $W = AVG$. Since A is an $n \times n$ invertible matrix and V consists of an $n \times m$ matrix with m linearly independent columns, AV is an $n \times m$ matrix of rank m. Thus $(AV)^T AV$ is a nonsingular $m \times m$ matrix. Hence $B = W^T AV = G^T (AV)^T (AV)$ is nonsingular. □

Let P_K and Q_K^L be the orthogonal projection onto the space K, and the (oblique) projection to K orthogonal to L, respectively. They are defined by

$$P_K x \in K, \qquad x - P_K x \perp K, \tag{8.8a}$$
$$Q_K^L x \in K, \qquad x - Q_K^L x \perp L. \tag{8.8b}$$

Explicit formula for P_K and Q_K are given as follows: Let $P_K x = \sum_{j=1}^n \alpha_j v_j$ and $Q_K^L x = \sum_{j=1}^n \beta_j v_j$. Then, notice that the orthogonalities

$$(x - P_K x, v_k) = 0, \quad (x - Q_K^L x, Av_k) = 0, \quad \forall k = 1, \cdots, m,$$

are equivalent to

$$(x - \sum_{j=1}^n \alpha_j v_j, v_k) = 0, \quad (x - \sum_{j=1}^n \beta_j v_j, Av_k) = 0, \quad \forall k = 1, \cdots, m.$$

Set $p_{kj} = (v_j, v_k), q_{kj} = (v_j, Av_k)$ for $j, k = 1, \cdots, m$, and $c_k = (x, v_k), d_k = (x, Av_k)$ for $k = 1, \cdots, m$, and P and Q be $m \times m$ matrices with components p_{jk}'s and q_{jk}'s and c and d are m-dimensional vectors with components c_j's and d_j's, respectively. Then the cofficients α_js and β_j's are given by

$$(\alpha_1, \cdots, \alpha_m)^t = P^{-1} c, \tag{8.9a}$$
$$(\beta_1, \cdots, \beta_m)^t = Q^{-1} d. \tag{8.9b}$$

Remark 8.1. *If the bases are orthogonal, one has much simpler formula*

$$P_K x = \sum_{j=1}^{m} \frac{(x, v_j)}{(v_j, v_j)} v_j, \text{ if } v_j's \text{ are orthogonal,} \tag{8.10a}$$

$$Q_K^L x = \sum_{j=1}^{m} \frac{(x, Av_j)}{(v_j, Av_j)} v_j, \text{ if } v_j's \text{ are } A\text{-orthogonal.} \tag{8.10b}$$

Proposition 8.4. *Assume that $r^{(j)} = b - Ax^{(j)} \in K$ and K is invariant under A. Then the solution $x^{(j+1)}$ obtained by*

$$Q_K^L(b - Ax^{(j+1)}) = 0$$

is the exact solution of $Ax = b$.

Proof. Let $x^{(j+1)} = x^{(j)} + d^{(j)}$ with $d^{(j)} \in K$. Then $Ad^{(j)} \in AK = L$, which together with $r^{(j)} \in K$ yields

$$\begin{aligned} 0 &= Q_K^L(b - Ax^{(j+1)}) = Q_K^L(b - Ax^{(j)} - Ad^{(j)}) = Q_K^L(r^{(j)} - Ad^{(j)}) \\ &= r^{(j)} - Ad^{(j)} = b - Ax^{(j)} - Ad^{(j)} = b - A(x^{(j)} + d^{(j)}) = b - Ax^{(j+1)}. \end{aligned}$$

□

8.2 Krylov subspace methods

8.2.1 Krylov subspaces

Definition 8.2. *Let $v \in \mathbb{R}^n$ be given. Then the Krylov subspaces $K_m = K_m(A, v)$ are defined by*

$$K_m = K_m(A, v) = \text{Span}\{v, Av, \cdots, A^{m-1}v\}.$$

The grade $\mu(A, v)$ of v with respect to A is the lowest degree of polynomial p such that

$$p(A)v = 0.$$

Observe that

$$K_m(A, v) = \{p(A)v : \ p \in P_{m-1}, \}$$

where P_k denotes the set of all polynomial of degree $\leq k$. Also, notice that

$$\dim K_m(A, v) = \mu(A, v).$$

We have the following propositions.

Proposition 8.5. *Let $\mu(A, v)$ be the grade of v with respect to A. Then $\mathcal{K}_\mu$ is invariant under A and $\mathcal{K}_m = \mathcal{K}_\mu$ for all $m \geq \mu(A, v)$.*

Proof. Since the grade of v is $\mu(A, v)$, there exists a polynomial of degree $\mu(A, v)$ such that $p(A)v = 0$. Suppose $w \in \mathcal{K}$. Then $w = q(A)v$ for some polynomial of degree $\mu(A, v)-1$. We have $tq(t) = ap(t)+b(t)$ with a constant a and a polynomial b of degree $\mu(A, v) - 1$. Thus we have $Aw = Aq(A)v = ap(A)v + b(A)v = b(A)v \in \mathcal{K}_\mu(A, v)$. This shows that $\mathcal{K}_\mu$ is invariant under A. Moreover, this implies that $\mathcal{K}_{\mu+1} = \mathcal{K}_\mu$, which obviously leads to $\mathcal{K}_m = \mathcal{K}_\mu$ for all $m \geq \mu(A, v)$. □

Proposition 8.6 (p. 152, [6])**.**

$$\dim K_m(A, v) = m \text{ if and only if } \mu(A, v) \geq m. \tag{8.11}$$

We will consider a Krylov subspace projection method such that

$$d^{(j)} \in K_{j+1}; \quad r^{(j+1)} \perp L_{j+1}(= K_{j+1})$$

in the next subsection.

8.2.2 The Conjugate Gradient Method

In case *A is a SPD (symmetric positive definite) matrix*, the (CG) Conjugate Gradient method is one of the most widely used iterative schemes to solve a sparse linear system $Ax = b$.

Denoting $d^{(j)} = \alpha^{(j)}p^{(j)}$, we begin by stating the CG Algorithm as in the following form:

ALGORITHM 8.1. *CG: Conjugate Gradient Algorithm*

1 Choose an initial approximation $x^{(0)}$
2 $r^{(0)} = b - Ax^{(0)}; p^{(0)} = r^{(0)}$
3 do j=0, max_iter
4 $\quad q^{(j)} = Ap^{(j)}$
5 $\quad \alpha^{(j)} = (p^{(j)}, r^{(j)})/(q^{(j)}, p^{(j)})$
6 $\quad x^{(j+1)} = x^{(j)} + \alpha^{(j)}p^{(j)}$
7 $\quad r^{(j+1)} = r^{(j)} - \alpha^{(j)}q^{(j)}$

8 $\beta^{(j)} = (r^{(j+1)}, r^{(j+1)})/(r^{(j)}, r^{(j)})$

9 $p^{(j+1)} = r^{(j+1)} + \beta^{(j)} p^{(j)}$

10 `enddo`

Remark 8.2. *Notice that in the above algorithm, the expensive part which requires the only 1 matrix-vector multiplication $Ap^{(j)}$ in the line 1 and 3 vector inner product in the lines 5 and 8. Thus this version of CG algorithm requires*

$$1 \text{ matrix-vector multiplication } + 3 \text{ vector inner product.}$$

8.2.3 The derivaton of the conjugate gradient method

The idea of derivation is to keep the following properties:

1. For $\ell = 0, 1, \cdots$, we have
$$K_\ell(A, r^{(0)}) = \text{Span}\{p^{(0)}, \cdots, p^{(\ell-1)}\} = \text{Span}\{r^{(0)}, \cdots, r^{(\ell-1)}\}$$
2. The search directions $p^{(j)}$ are pairwise A-conjugate, i.e,
$$\left\langle p^{(j)}, p^{(k)} \right\rangle_A = 0, \quad j \neq k.$$
3. Furthermore, the residuals $r^{(j)}$ are orthogonal, *i.e,*
$$\left\langle r^{(j)}, r^{(k)} \right\rangle = 0, \quad j \neq k.$$

$$\begin{aligned}
x^{(1)} &= x^{(0)} + \alpha^{(0)} p^{(0)} \in K_1 \\
r^{(1)} &= r^{(0)} - \alpha^{(0)} A p^{(0)} \in K_2 \\
p^{(1)} &= r^{(1)} + \beta^{(0)} p^{(0)} \in K_2 = r^{(0)} - \alpha^{(0)} A p^{(0)} + \beta^{(0)} p^{(0)} \in K_2 \\
x^{(2)} &= x^{(1)} + \alpha^{(1)} p^{(1)} \in K_2 \\
r^{(2)} &= r^{(1)} - \alpha^{(1)} A p^{(1)} \in K_3
\end{aligned}$$

$$r^{(0)} \in K_1, r^{(1)} \in K_2, r^{(1)} \perp K_1,$$
$$r^{(1)} = r^{(0)} - \alpha^{(0)} A p^{(0)} \perp r^{(0)}$$

implies that

$$\alpha^{(j)} = (p^{(j)}, r^{(j)})/(q^{(j)}, p^{(j)}) \quad \text{for } j = 0.$$

Assume that A is a symmetric positive-definite matrix. Recall that quadratic function $J : \mathbb{R}^n \to \mathbb{R}$ defined by

$$J(\eta) = \frac{1}{2}\eta^t A\eta - b^t\eta,$$

whose minimization solution is the solution to linear system $Ax = b$.

The conjugate gradient method is minimizing the functional J such that

1. the step length $\alpha^{(j)}$ is optimal;
2. the search directions $p^{(j)}$ are conjugate in the sense that

$$p^{(j)} \cdot Ap^{(k)} = 0 \text{ for all } k < j.$$

Denote by $\langle \cdot, \cdot \rangle_A$

$$\langle \zeta, \eta \rangle_A = \zeta \cdot A\eta, \quad \zeta, \eta \in \mathbb{R}^n.$$

Since A is symmetric, positive-definite, $\langle \cdot, \cdot \rangle_A$ defines an inner product for $\mathbb{R}^n$.

Exercise 8.1. *Suppose that A is symmetric, positive-definite. Then show that $\langle \cdot, \cdot \rangle_A$ defines an inner product for $\mathbb{R}^n$.*

With the definition of the scalar product $\langle \cdot, \cdot \rangle_A$ define the jth residual

$$r^{(j)} := -\nabla J(\chi^{(j)}) = b - A\chi^{(j)}.$$

Consider an algorithm to find the solution and search direction vectors $x^{(j)}$ and $p^{(j)}$ such that

$$x^{(j+1)} = x^{(j)} + \alpha^{(j)}p^{(j)}, \tag{8.12a}$$
$$p^{(j+1)} = r^{(j+1)} + \beta^{(j)}p^{(j)}, \tag{8.12b}$$

with the residual $r^{(j+1)}$ orthogonal to the direction $p^{(j)}$ and the directions $p^{(j+1)}$ and $p^{(j)}$ being A-conjugate. The $\beta^{(j)}$ is chosen such that $x^{(j+1)}$ is minimized in the search direction $p^{(j)}$.

Notice that

$$-\nabla J\left(x^{(j)} + \alpha p^{(j)}\right) = b - A\left(x^{(j)} + \alpha p^{(j)}\right) = r^{(j)} - \alpha Ap^{(j)}. \tag{8.13}$$

Therefore, we again define

$$\phi(\alpha) := J\left(x^{(j)} + \alpha p^{(j)}\right).$$

Then, the minimum of ϕ can be calculated as follows:

$$\frac{d\phi}{d\alpha}(\alpha^{(j)}) = \left[\nabla J\left(x^{(j)} + \alpha^{(j)} p^{(j)}\right)\right]^T p^{(j)} = 0, \tag{8.14}$$

which implies, owing to (8.13), that

$$\alpha^{(j)} \left[p^{(j)}\right]^T A p^{(j)} - \left[r^{(j)}\right]^T p^{(j)} = 0.$$

We therefore have

$$\alpha^{(j)} = \frac{\left[p^{(j)}\right]^T r^{(j)}}{\left[p^{(j)}\right]^T A p^{(j)}} = \frac{r^{(j)} \cdot p^{(j)}}{\left\langle p^{(j)}, p^{(j)} \right\rangle_A}. \tag{8.15}$$

Next, since $p^{(j+1)}$ and $p^{(j)}$ are A-conjugate, (8.12b) yields

$$0 = \left\langle p^{(j+1)}, p^{(j)} \right\rangle_A = \left\langle r^{(j+1)} + \beta^{(j)} p^{(j)}, p^{(j)} \right\rangle_A,$$

and therefore,

$$\beta^{(j)} = -\frac{\left\langle r^{(j+1)}, p^{(j)} \right\rangle_A}{\left\langle p^{(j)}, p^{(j)} \right\rangle_A}. \tag{8.16}$$

The resulting algorithm is the conjugate gradient method as follows: given $x^{(0)} \in \mathbb{R}^n$ and $p^{(0)} = r^{(0)}$, compute iteratively $x^{(j)}$ and $p^{(j)}$ by

$$x^{(j+1)} = x^{(j)} + \alpha^{(j)} p^{(j)}, \quad \alpha^{(j)} = \frac{r^{(j)} \cdot p^{(j)}}{\left\langle p^{(j)}, p^{(j)} \right\rangle_A},$$
$$p^{(j+1)} = r^{(j+1)} + \beta^{(j)} p^{(j)}, \quad \beta^{(j)} = -\frac{\left\langle r^{(j+1)}, p^{(j)} \right\rangle_A}{\left\langle p^{(j)}, p^{(j)} \right\rangle_A}.$$

We then have the following lemmata.

Lemma 8.1. *For $\ell = 0, 1, \cdots$, we have*

$$K_\ell(A, r^{(0)}) = \mathrm{Span}\{p^{(0)}, \cdots, p^{(\ell-1)}\} = \mathrm{Span}\{r^{(0)}, \cdots, r^{(\ell-1)}\}$$

Lemma 8.2. *The search directions $p^{(j)}$ are pairwise A-conjugate, i.e,*

$$\left\langle p^{(j)}, p^{(k)} \right\rangle_A = 0, \quad j \neq k.$$

Furthermore, the gradients $r^{(j)}$ are orthogonal, i.e,

$$\left\langle r^{(j)}, r^{(k)} \right\rangle = 0, \quad j \neq k.$$

Based on the above lemmata, we have

Theorem 8.1. *For some $j \leq n$, we have $Ax^{(j)} = b$.*

Due to Lemma 8.1 it follows that

$$p^{(j-1)} \in K_j = K_j(A, r^{(0)}), \text{ and } \left\langle r^{(j)}, p^{(k)} \right\rangle = 0 \quad \text{for all } k = 0, \cdots, j-1 \tag{8.17}$$

In other words,

$$p^{(j-1)} \in K_j = K_j(A, r^{(0)}), \text{ and } r^{(j)} \perp K_j = K_j(A, r^{(0)}). \tag{8.18}$$

Therefore, the CG scheme can be regarded as a Krylov subspace projection method.

We now summarize the algorithm as follows with fewer flops calculations than Algorithm 8.2.2.

ALGORITHM 8.2.

1 Choose an initial approximation $x^{(0)}$
2 $r^{(0)} = b - Ax^{(0)}; p^{(0)} = r^{(0)}$
3 do j=0, max_iter
4 $\quad \gamma^{(j)} = \left\langle p^{(j)}, p^{(j)} \right\rangle_A$
5 $\quad \alpha^{(j)} = (p^{(j)}, r^{(j)})/\gamma^{(j)} = (r^{(j)}, r^{(j)})/\gamma^{(j)}$
6 $\quad x^{(j+1)} = x^{(j)} + \alpha^{(j)} p^{(j)}$
7 $\quad r^{(j+1)} = b - Ax^{(j+1)}$ or $r^{(j+1)} = r^{(j)} - \alpha^{(j)} Ap^{(j)}$
8 $\quad \beta^{(j)} = -\left\langle r^{(j+1)}, p^{(j)} \right\rangle_A / \gamma^{(j)} = (r^{(j+1)}, r^{(j+1)})/(\alpha^{(j)}\gamma^{(j)})$
9 $\quad p^{(j+1)} = r^{(j+1)} + \beta^{(j)} p^{(j)}$
10 enddo

Remark 8.3. *Notice that in the above algorithm, the expensive part which requires the matrix-vector multiplication $Ap^{(j)}$ in the lines 1, 7, and 8 can be computed only once and used repeatedly. Thus, each step of CG algorithm requires*

$$1 \text{ matrix-vector multiplication } + 3 \text{ vector inner product.}$$

Exercise 8.2. *Implement the above conjugate gradient method and solve the two-dimensional elliptic problem Example 2.5.*

Exercise 8.3. *Consider the two-dimensional elliptic problem to solve*

$$-\Delta u = f, \ \Omega = (0,1)^2, \tag{8.19}$$
$$u = 0, \ \partial\Omega, \tag{8.20}$$

with $f(x,y) = 13\pi^2 \sin(2\pi x)\sin(3\pi y)$ with the exact solution $u(x,y) = \sin(2\pi x)\sin(3\pi y$ Show that with the initial approximation $u_h(x,y) = 0$ in the CG scheme, the iteration converges at one iteration.

We show a two-dimensional elliptic solver by CG method as follows.

F90 Program 8.1: Conjugate Gradient Method

```
1  !June 7, 2007, Written by D.Sheen http://www.nasc.snu.ac.kr
2  ! This is a two-dimension elliptic problem solver with Dirichlet BC u=0.
3  ! - Delta u = f; f = 2 sin(pi y) + pi^2 x(1-x) sin(pi y)
4  ! exact sol u = x(1-x) sin(pi y)
5  ! try with n = 100
6  program cg
7    implicit none
8    integer:: k, j, jk, max_iter, n, nm12, iter
9    real(8):: tol_F, alpha, beta, app, res, h, pi, xj, yk
10   real(8), dimension(:), allocatable:: x, p, r, b, ap
11   real(8), dimension(:,:), allocatable:: a
12
13   interface
14      function matrix_mul(a,x,n)
15        integer:: j,k,n
16        real(8):: a(1:(n-1)**2,-2:2), x(1:(n-1)**2), y(0:n,0:n), b(1:(n-1)**2)&
17        , matrix_mul(1:(n-1)**2)
18
19      end function matrix_mul
20
21      function ind(j,k,n)
22        integer:: j,k,n,ind
23      end function ind
24
25      function l2err(h,n,x)
26        integer:: n, j, k, ind
27        real(8) :: l2err, h, u, xj, yk
```

```
28          real(8) :: x(1:(n-1)**2)
29        end function l2err
30
31        function u(x,y)
32          real(8):: x,y,u, pi
33        end function u
34
35        function f(x,y)
36          real(8):: x,y,f, pi
37        end function f
38      end interface
39
40      print*, "n = "
41      read(5,*) n
42      nm12 = (n-1)**2
43      h = 1.d0/dble(n);   pi = 4.d0 * atan(1.0d0)
44      allocate(x(1:nm12), b(1:nm12), p(1:nm12), r(1:nm12), ap(1:nm12)&
45            , a(1:nm12,-2:2))
46      print*, "size(x) =", size(x)
47
48      a(:,0) = 4.d0
49      a(:,-1) = -1.d0;    a(:,1) = -1.d0
50      a(:,-2) = -1.d0;    a(:,2) = -1.d0
51
52      do j = 1, n-1
53         a( (j-1)*(n-1)+1,-1) = 0.d0;   a(j*(n-1),1) = 0.d0
54      end do
55
56      do j = 1, n-1
57         a(j,-2) = 0.d0;         a( nm12+1-j, 2) = 0.d0;
58      end do
59
60    94 format(2i4, 5g11.3)
61
62      b = 0.d0 ; !    b(1) = dble(n)
63      do k = 1, n-1
64         yk = k*h
65         do j = 1, n-1
66            xj = j*h
67            b(ind(j,k,n)) = h**2 * f(xj,yk)
68         end do
69      end do
70
71      max_iter = 1000;  tol_F = 1.d-12
72
73      write(6,90)
74
75    90 format(3x,"k",17x,"res")
76    91 format(i6,5x,g11.3)
77    92 format(5g11.3)
78
79      x = 0.d0
80      r = b - matrix_mul(a,x,n); p= r
81      do iter = 1, max_iter
82         res = sqrt(dot_product(r,r))
83              write(6,91) iter, res
```

```
84        !      write(6,92) x(1:5)
85        if( res < tol_F) then
86           write(6,*) 'res = ', res
87           !         write(6,*) x
88           write(6,*) 'l2-err with n ', n, '   = ',  l2err(h,n,x)
89           print*, "Converged at ", iter, " iteration."
90           stop 999
91        end if
92        ap = matrix_mul(a,p,n)
93        app = dot_product(ap,p)
94        alpha =  dot_product(r,p) /app
95        x = x + alpha*p
96        r = r - alpha*ap ; beta = -dot_product(matrix_mul(a,r,n),p)/app
97        p = r + beta*p
98     enddo
99
100 end program cg
101
102 function matrix_mul(a,x,n)
103    implicit none
104    integer:: j,k,n,ind,jk
105    real(8):: a(1:(n-1)**2,-2:2), x(1:(n-1)**2), y(0:n,0:n), b(1:(n-1)**2) &
106          , matrix_mul(1:(n-1)**2)
107    real(8):: ac, ae, aw, as, an
108
109    y = 0.d0
110    do k = 1, n-1
111       do j = 1, n-1
112          jk = ind(j,k,n)
113          y(j,k) = x(jk)
114       end do
115    end do
116
117    do k = 1, n-1
118       do j = 1, n-1
119          jk = ind(j,k,n)
120          ae=0.d0;          aw=0.d0;          as=0.d0;          an=0.d0
121          ac = a(jk,0)
122          ae = a(jk,-1);          aw = a(jk,1)
123          as = a(jk,-2);          an = a(jk,2)
124          b(jk) = ac*y(j,k) + ae*y(j,k-1) +aw*y(j,k+1) + as*y(j-1,k) + an*y(j+1,k)
125       end do
126    end do
127    matrix_mul = b
128 end function matrix_mul
129
130 function ind(j,k,n)
131    implicit none
132    integer:: j,k,n,ind
133    ind =  (j-1)*(n-1) + k
134 end function ind
135
136 function l2err(h,n,x)
137    implicit none
138    integer:: n, j, k, ind
139    real(8) :: l2err, h, u, xj, yk
```

```
140   real(8) :: x(1:(n-1)**2)
141
142   l2err = 0.d0
143   do k = 1, n-1
144      yk = k*h
145      do j =1, n-1
146         xj = j*h
147         l2err = l2err +  (x(ind(j,k,n)) -  u(xj,yk) )**2
148      end do
149   end do
150   l2err = sqrt(l2err)*h * dble(n)/ ( dble(n)-1.d0)
151 end function l2err
152
153 function u(x,y)
154   implicit none
155   real(8):: x,y,u, pi
156   pi = 4.d0 * atan(1.d0)
157   u = x*(1.d0-x) * sin(pi*y)
158 end function u
159
160 function f(x,y)
161   implicit none
162   real(8):: x,y,f, pi
163   pi = 4.d0 * atan(1.d0)
164   f = 2.d0*sin(pi*y) + pi**2*x*(1.d0-x)* sin(pi*y)
165 end function f
```

8.2.4 Preconditioned Conjugate Gradient Method

Suppose A is a $n \times n$ symmetric positive-definite matrix. Let $M = LL^T$ be a suitable preconditioner (implicitly assuming that M is an $n \times n$ SPD matrix) for A. Then it is straightforward to see that

$$\left\langle M^{-1}Ax, y \right\rangle_M = \left\langle x, M^{-1}Ay \right\rangle_M \quad \text{for all } x, y \in \mathbb{R}^n, \tag{8.21}$$

which implies that $M^{-1}A$ is symmetric positive-definite with respect to the inner-product $\langle \cdot, \cdot \rangle_M$.

Exercise 8.4. *Show that $M^{-1}A$ is symmetric positive-definite with respect to the inner-product $(\cdot, \cdot)_A$.*

The idea of PCG is to apply the idea of CG to

$$M^{-1}Ax = M^{-1}b \tag{8.22}$$

In this case, the residual $z^{(j)} = M^{-1}(b - Ax^{(j)})$ for this system is related with the residual $r^{(j)} = b - Ax^{(j)}$ in the form

$$z^{(j)} = M^{-1}r^{(j)}. \tag{8.23}$$

We then have the following properties:

Theorem 8.2.

$$(z^{(j)}, z^{(k)})_M = 0, \quad j \neq k, \tag{8.24a}$$
$$(M^{-1}Ap^{(j)}, p^{(k)})_M = 0, \quad j \neq k. \tag{8.24b}$$

Then the strategy to update at each iterate is as follows: find

$$x^{(j+1)} = x^{(j)} + \alpha^{(j)} p^{(j)}, \tag{8.25a}$$
$$r^{(j+1)} = r^{(j)} - \alpha^{(j)} Ap^{(j)}, \tag{8.25b}$$
$$z^{(j+1)} = M^{-1}r^{(j+1)} = z^{(j)} - \alpha^{(j)} M^{-1}Ap^{(j)}, \tag{8.25c}$$
$$p^{(j+1)} = z^{(j+1)} + \beta^{(j)} p^{(j)}, \tag{8.25d}$$

where $\alpha^{(j)}$ and $\beta^{(j)}$ are determined by the properties (8.24). Indeed, due to (8.24a) and (8.25c),

$$0 = (z^{(j+1)}, z^{(j)})_M = (z^{(j)} - \alpha^{(j)} M^{-1}Ap^{(j)}, z^{(j)})_M,$$

from which an application of (8.25d) and (8.24b) implies that

$$\begin{aligned} \alpha^{(j)} &= \frac{(z^{(j)}, z^{(j)})_M}{(M^{-1}Ap^{(j)}, z^{(j)})_M} = \frac{(z^{(j)}, z^{(j)})_M}{(M^{-1}Ap^{(j)}, p^{(j)} - \beta^{(j-1)}p^{(j-1)})_M} \\ &= \frac{(z^{(j)}, z^{(j)})_M}{(M^{-1}Ap^{(j)}, p^{(j)})_M}. \end{aligned}$$

Thus,

$$\alpha^{(j)} = \frac{(z^{(j)}, z^{(j)})_M}{(M^{-1}Ap^{(j)}, p^{(j)})_M}. \tag{8.26}$$

Next, due to (8.24b) and (8.25d),

$$0 = (p^{(j+1)}, M^{-1}Ap^{(j)})_M = (z^{(j+1)} + \beta^{(j)}p^{(j)}, M^{-1}Ap^{(j)})_M.$$

Hence, by using (8.25c), (8.24a), and (8.26),

$$\begin{aligned} \beta^{(j)} &= -\frac{(z^{(j+1)}, M^{-1}Ap^{(j)})_M}{(p^{(j)}, M^{-1}Ap^{(j)})_M} = \frac{(z^{(j+1)}, \frac{1}{\alpha^{(j)}}(z^{(j+1)} - z^{(j)})_M}{(p^{(j)}, M^{-1}Ap^{(j)})_M} \\ &= \frac{1}{\alpha^{(j)}} \frac{(z^{(j+1)}, z^{(j+1)})_M}{(p^{(j)}, M^{-1}Ap^{(j)})_M} = \frac{(M^{-1}Ap^{(j)}, p^{(j)})_M}{(z^{(j)}, z^{(j)})_M} \frac{(z^{(j+1)}, z^{(j+1)})_M}{(p^{(j)}, M^{-1}Ap^{(j)})_M} \\ &= \frac{(z^{(j+1)}, z^{(j+1)})_M}{(z^{(j)}, z^{(j)})_M}. \end{aligned}$$

$$\beta^{(j)} = \frac{(z^{(j+1)}, z^{(j+1)})_M}{(z^{(j)}, z^{(j)})_M}. \tag{8.27}$$

ALGORITHM 8.3.

1 Choose an initial approximation $x^{(0)}$
2 $r^{(0)} = b - Ax^{(0)}; z^{(0)} = M^{-1}r^{(0)}; p^{(0)} = z^{(0)}$
3 do j=0, max_iter
4 $\quad \alpha^{(j)} = \frac{(z^{(j)},z^{(j)})_M}{(M^{-1}Ap^{(j)},p^{(j)})_M} = \frac{(r^{(j)},z^{(j)})}{(Ap^{(j)},p^{(j)})}$
5 $\quad x^{(j+1)} = x^{(j)} + \alpha^{(j)}p^{(j)}$
6 $\quad r^{(j+1)} = r^{(j)} - \alpha^{(j)}Ap^{(j)}$
7 $\quad z^{(j+1)} = M^{-1}r^{(j+1)}$
8 $\quad \beta^{(j)} = \frac{(z^{(j+1)},z^{(j+1)})_M}{(z^{(j)},z^{(j)})_M} = \frac{(r^{(j+1)},z^{(j+1)})}{(r^{(j)},z^{(j)})}$
9 $\quad p^{(j+1)} = z^{(j+1)} + \beta^{(j)}p^{(j)}$
10 enddo

8.2.5 The Conjugate Residual Scheme: A is Hermitian

Assume that A is Hermitian. The CR scheme is derived with the following A-conjugacy and orthogonality conditions on the residuals and search vectors, respectively:

$$(r^{(j+1)}, Ar^{(j)}) = 0, \tag{8.28a}$$
$$(Ap^{(j)}, Ap^{(j+1)}) = 0, \tag{8.28b}$$

Recall that

$$x^{(j+1)} = x^{(j)} + \alpha^{(j)}p^{(j)}, \tag{8.29a}$$
$$r^{(j+1)} = r^{(j)} - \alpha^{(j)}Ap^{(j)}, \tag{8.29b}$$
$$p^{(j+1)} = r^{(j+1)} + \beta^{(j)}p^{(j)}. \tag{8.29c}$$

Then (8.28a) leads to

$$0 = (r^{(j+1)}, Ar^{(j)}) = (r^{(j)} - \alpha^{(j)}Ap^{(j)}, Ar^{(j)}) = (r^{(j)}, Ar^{(j)}) - \alpha^{(j)}(Ap^{(j)}, Ar^{(j)}).$$

Thus,

$$\alpha^{(j)} = (r^{(j)}, Ar^{(j)})/(Ap^{(j)}, Ar^{(j)}).$$

Notice that (8.28b) implies that

$$\begin{aligned}(Ap^{(j)}, Ap^{(j)}) - (Ap^{(j)}, Ar^{(j)}) &= \left(Ap^{(j)}, A(p^{(j)} - r^{(j)})\right) \quad (8.30)\\ &= \left(Ap^{(j)}, A\beta^{(j-1)}p^{(j-1)}\right) \quad (8.31)\\ &= \beta^{(j-1)}\left(Ap^{(j)}, Ap^{(j-1)}\right) = 0\end{aligned}$$

Hence

$$\alpha^{(j)} = (r^{(j)}, Ar^{(j)})/(Ap^{(j)}, Ap^{(j)}). \quad (8.32)$$

Due to again from (8.28b),

$$\begin{aligned}0 &= (Ap^{(j)}, Ap^{(j+1)}) = \left(Ap^{(j)}, Ar^{(j+1)} + \beta^{(j)}Ap^{(j)}\right)\\ &= \left(Ap^{(j)}, Ar^{(j+1)}\right) + \beta^{(j)}\left(Ap^{(j)}, Ap^{(j)}\right),\end{aligned}$$

and thus

$$\beta^{(j)} = -\left(Ap^{(j)}, Ar^{(j+1)}\right) / \left(Ap^{(j)}, Ap^{(j)}\right).$$

It follows from (8.28a), (8.29b) and (8.30) that

$$\left(Ap^{(j)}, Ap^{(j)}\right) = \left(Ap^{(j)}, Ar^{(j)}\right) = \left(\frac{1}{\alpha^{(j)}}(r^{(j)} - r^{(j+1)}), Ar^{(j)}\right) = \frac{1}{\alpha^{(j)}}\left(r^{(j)}, Ar^{(j)}\right),$$

and that

$$-\left(Ap^{(j)}, Ar^{(j+1)}\right) = -\left(\frac{1}{\alpha^{(j)}}(r^{(j)} - r^{(j+1)}), Ar^{(j+1)}\right) = \frac{1}{\alpha^{(j)}}\left(r^{(j+1)}, Ar^{(j+1)}\right).$$

Hence

$$\beta^{(j)} = \left(r^{(j+1)}, Ar^{(j+1)}\right) / \left(r^{(j)}, Ar^{(j)}\right). \quad (8.33)$$

Summarizing the above, in the the following Conjugate Residual Algorithm has been derived:

ALGORITHM 8.4. *CR: Conjugate Residual Algorithm*

1 Choose an initial approximation $x^{(0)}$

2 $r^{(0)} = b - Ax^{(0)}; p^{(0)} = r^{(0)}$

3 do j=0, max_iter

4 $\alpha^{(j)} = (r^{(j)}, Ar^{(j)})/(Ap^{(j)}, Ap^{(j)})$
5 $x^{(j+1)} = x^{(j)} + \alpha^{(j)} p^{(j)}$
6 $r^{(j+1)} = r^{(j)} - \alpha^{(j)} Ap^{(j)}$
7 $\beta^{(j)} = (r^{(j+1)}, Ar^{(j+1)})/(r^{(j)}, Ar^{(j)})$
8 $p^{(j+1)} = r^{(j+1)} + \beta^{(j)} p^{(j)}$
9 Compute $Ap^{(j+1)} = Ar^{(j+1)} + \beta^{(j)} Ad^{(j)}$
10 `enddo`

Remark 8.4. *Notice that minimizing residual scheme is related with least squares.*

8.3 The steepest descent method - the gradient method

Let A be an $n \times n$ invertible matrix. Define a quadratic function $f : \mathbb{R}^n \to \mathbb{R}$ defined by

$$f(\eta) = \|A\eta - b\|_2^2$$

Then, the solution of

$$Ax = b \tag{8.34}$$

is identical to that of the minimization problem

$$\min_{\eta \in \mathbb{R}^n} f(\eta).$$

In order to solve the above minimization problem iteratively, it is natural to search the next iterate $x^{(j+1)}$ from the current iterate $x^{(j)}$ to the steepest descent direction to reduce the function values maximally. Notice that the gradient of $f(\eta)$ is given by

$$\nabla f(\eta) = A^T(A\eta - b). \tag{8.35}$$

The gradient method is based on the this idea. Let $x^{(0)}$ be an initial approximation to (8.34). Then iteratively one finds the next iterate $x^{(j+1)}$ from $x^{(j)}$

in the negative direction to the gradient, *i.e.*,

$$\begin{aligned} x^{(j+1)} &= x^{(j)} + s^{(j)} \left(-\nabla f(x^{(j)}) \right) \\ &= x^{(j)} + s^{(j)} \left(-A^T (Ax^{(j)} - b) \right) \\ &= x^{(j)} + s^{(j)} A^T r^{(j)}, \end{aligned} \tag{8.36}$$

where $r^{(j)} = b - Ax^{(j)}$ is the residual at the j-th iterate. Notice that (8.36) can be regarded as a kind of dynamic Richardson iterative method (7.6) with the parameter $s^{(j)}$ and $P = A^{-T}$. It therefore follows that $s^{(j)}$ needs to be chosen sufficiently small such that the spectral radius $\rho(s^{(j)} A^T) < 1$ but not too small in order to make the value $f(x^{(j+1)})$ decrease as much as possible. The idea of steepest descent method is to choose the step length $s^{(j)} > 0$ to minimize

$$\phi(s) := f \left(x^{(j)} + s A^T r^{(j)} \right). \tag{8.37}$$

Since f is quadratic in its argument, the scalar-valued function $\phi(s)$ is also a quadratic function in s. Hence the minimizer of $\phi(s)$ is obtained by finding $s^{(j)}$ such that $\frac{d\phi}{ds}(s^{(j)}) = 0$. This means that

$$\frac{d\phi}{ds}(s) = \left[\nabla f \left(x^{(j)} + s A^T r^{(j)} \right) \right]^T A^T r^{(j)} = 0. \tag{8.38}$$

With the aid of (8.35) that

$$\begin{aligned} 0 &= \left[A^T \left(A \left(x^{(j)} + s A^T r^{(j)} \right) - b \right) \right]^T A^T r^{(j)} \\ &= \left[A^T \left(s A A^T r^{(j)} - r^{(j)} \right) \right]^T A^T r^{(j)} \\ &= \left(s \left[r^{(j)} \right]^T A A^T - \left[r^{(j)} \right]^T \right) A A^T r^{(j)} \\ &= s \left[r^{(j)} \right]^T A A^T A A^T r^{(j)} - \left[r^{(j)} \right]^T A A^T r^{(j)} \\ &= s [A\xi^{(j)}]^T [A\xi^{(j)}] - [\xi^{(j)}]^T \xi^{(j)}. \end{aligned}$$

where

$$\xi^{(j)} = A^T r^{(j)} = A^T (b - Ax^{(j)}).$$

We therefore have that $s^{(j)}$ is given by

$$s^{(j)} = \frac{\|\xi^{(j)}\|_2^2}{\|A\xi^{(j)}\|_2^2}. \tag{8.39}$$

We thus have the following algorithm.

ALGORITHM 8.5. *Gradient Method 1*

1 $x^{(1)}$ is an initial approximation
2 do j=1, max_iter
3 $r^{(j)} = b - Ax^{(j)}$
4 $\xi^{(j)} = A^T r^{(j)}$
5 $s^{(j)} = \frac{\|\xi^{(j)}\|_2^2}{\|A\xi^{(j)}\|_2^2}$
6 $x^{(j+1)} = x^{(j)} + s^{(j)}\xi^{(j)}$
7 enddo

Let us look at the flops to calculate each iterative step, let $\nu(A)$ be the nonzero component of the matrix A. Then, in the calculation of the residual $r^{(j)}$, one needs exactly $\nu(A)$ flops. Again in computing $\xi^{(j)}$, one needs additional $\nu(A)$ flops. In finding $s^{(j)}$, the number of flops to compute the multiplication of matrix A and $\xi^{(j)}$ is $\nu(A)$, and that to compute the inner products $\|\xi^{(j)}\|_2^2$ and $\|A\xi^{(j)}\|_2^2$ is $2n$. Finally to update the iterate to $x^{(j+1)}$, one has to add only n flops. Summing up, for each step of the algorithm, the flops required should be $3\nu(A) + 3n$.

8.3.1 Another gradient method: A is symmetric

Let A be an $n \times n$ invertible matrix. We now consider another type of gradient method by changing the quadratic function $f : \mathbb{R}^n \to \mathbb{R}$ into the one defined by

$$f(\eta) = \frac{1}{2}\eta^T A\eta - \eta^T b$$

Then, we hope that the solution of the linear system (8.34). is identical to that of the minimization problem

$$\min_{\eta \in \mathbb{R}^n} f(\eta).$$

However, this does not hold unless A is symmetric since

$$\nabla f(\eta) = \frac{A + A^T}{2}\eta - b. \tag{8.40}$$

We thus restrict to the case of symmetric matrices: let A be a symmetric $n \times n$ invertible matrix. In this case, we have

$$\nabla f(\eta) = A\eta - b, \tag{8.41}$$

which is the negative of the residual.

In order to solve the above minimization problem iteratively, it is natural to search the next iterate $x^{(j+1)}$ from the current iterate $x^{(j)}$ to the steepest descent direction to reduce the function values maximally.

The gradient method is based on the this idea. Let $x^{(0)}$ be an initial approximation to (8.34). Then iteratively one finds the next iterate $x^{(j+1)}$ from $x^{(j)}$ in the negative direction to the gradient, *i.e.*,

$$\begin{aligned} x^{(j+1)} &= x^{(j)} + s^{(j)} \left(-\nabla f(x^{(j)}) \right) \\ &= x^{(j)} + s^{(j)} \left(-(Ax^{(j)} - b) \right) \\ &= x^{(j)} + s^{(j)} r^{(j)}, \end{aligned} \tag{8.42}$$

where $r^{(j)} = b - Ax^{(j)}$ is the residual at the j-th iterate. Notice that (8.36) can be regarded as a kind of dynamic Richardson iterative method (7.6) with the parameter $s^{(j)}$ and $P = I$. It therefore follows that $s^{(j)}$ needs to be chosen sufficiently small such that the spectral radius $\rho(s^{(j)}) < 1$ but not too small in order to make the value $f(x^{(j+1)})$ decrease as much as possible. The idea of steepest descent method is to choose the step length $s^{(j)} > 0$ to minimize

$$\phi(s) := f\left(x^{(j)} + s r^{(j)} \right). \tag{8.43}$$

Since f is quadratic in its argument, the scalar-valued function $\phi(s)$ is also a quadratic function in s. Hence the minimizer of $\phi(s)$ is obtained by finding $s^{(j)}$ such that $\frac{d\phi}{ds}(s^{(j)}) = 0$. This means that

$$\frac{d\phi}{ds}(s) = \left[\nabla f\left(x^{(j)} + s r^{(j)} \right) \right]^T r^{(j)} = 0. \tag{8.44}$$

With the aid of (8.35) that

$$\begin{aligned} 0 &= \left[\left(A\left(x^{(j)}+sr^{(j)}\right)-b\right)\right]^T r^{(j)} \\ &= \left[\left(sAr^{(j)}-r^{(j)}\right)\right]^T r^{(j)} \\ &= s\left[r^{(j)}\right]^T Ar^{(j)} - \left[r^{(j)}\right]^T r^{(j)}. \end{aligned}$$

We therefore have that $s^{(j)}$ is given by

$$s^{(j)} = \frac{\|r^{(j)}\|_2^2}{\left[r^{(j)}\right]^T Ar^{(j)}}. \tag{8.45}$$

We thus have the following algorithm.

ALGORITHM 8.6. *Gradient Method 2*

1 $x^{(1)}$ is an initial approximation
2 do j=1, max_iter
3 $r^{(j)} = b - Ax^{(j)}$
4 $s^{(j)} = \frac{(r^{(j)},r^{(j)})}{(r^{(j)},Ar^{(j)})}$
5 $x^{(j+1)} = x^{(j)} + s^{(j)}r^{(j)}$
6 enddo

Let us look at the flops to calculate each iterative step, let $\nu(A)$ be the nonzero component of the matrix A. Then, in the calculation of the residual $r^{(j)}$, one needs exactly $\nu(A)$ flops. Again in computing $Ar^{(j)}$, one needs additional $\nu(A)$ flops; therefore, the evaluation of $\left[r^{(j)}\right]^T Ar^{(j)}$ will require $\nu(A)+n$ flops. Hence, the total flops to calculate $s^{(j)}$ is $\nu(A)+2n$. Finally to update the iterate to $x^{(j+1)}$, one has to add only n flops. Summing up, for each step of the algorithm, the total number of flops required is $2\nu(A)+3n$. Notice that the difference between the flops per each iterative step for [Gradient Method 1] and [Gradient Method 2] is the number of nonzero components in A. Therefore, to approximate the solution of $Ax = b$ using [Gradient Method 1] and [Gradient Method 2] up to certain tolerance, it should be said compatible if the iteration number for [Gradient Method 1] is about 2/3 that for [Gradient Method 2].

Exercise 8.5. *Implement [Gradient Algorithm 1] and [Gradient Algorithm 2] and compare the convergence behaviors to solve with the matrix given in* (2.29).

Exercise 8.6. *Explain why the [Gradient Algorithm 2] converges usually faster than [Gradient Algorithm 1].*

Bibliography

[1] J. Backus. Specifications for the IBM Mathematical FORmula TRANslating System, FORTRAN. Technical report, International Business Machinces Cooperation, New York, Nov. 1954. Preliminary Report, Programming Research Group, Applied Science Division.

[2] B. I. Clasen. Sur une nouvelle méthode de résolution des équations linéaires et sur l'application de cette méthode au calcul des déterminants. *Ann. Soc. Sci. Bruxelles*, 12:251–281, 1888.

[3] G. H. Golub and C. F. van Loan. *Matrix Computations*. Johns Hopkins Univ. Press, Baltimore, MD, third edition, 1996.

[4] K. Hoffman and R. Kunze. *Linear Algebra*. Prentice-Hall, Inc, Englewood Cliffs, New Jersey, second edition, 1971.

[5] W. Jordan. *Handbuch der Vermessungskunde, Erster Band, Dritte verbesserte Auflage, Metzler,*. Stuttgart, 1888.

[6] Y. Saad. *Iteretive Methods for Sparse Linear Systems*. SIAM, Philadelphia, second edition, 2003.

[7] J. Stoer and R. Bulirsch. *Introduction to Numerical Analysis*. Springer-Verlag, New York, second edition, 1993.